Elementos de Estatística Computacional Usando Plataformas de Software Livre/Gratuito

Publicações Matemáticas

Elementos de Estatística Computacional Usando Plataformas de Software Livre/Gratuito

Alejandro C. Frery
UFAL

Francisco Cribari-Neto
UFPE

impa

25º Colóquio Brasileiro de Matemática

Impresso no Brasil / Printed in Brazil

Capa: Noni Geiger / Sérgio R. Vaz

25º Colóquio Brasileiro de Matemática

- A Short Introduction to Numerical Analysis of Stochastic Differential Equations - Luis José Roman
- An Introduction to Gauge Theory and its Applications - Marcos Jardim
- Aplicações da Análise Combinatória à Mecânica Estatística - Domingos H. U. Marchetti
- Dynamics of Infinite-dimensional Groups and Ramsey-type Phenomena - Vladimir Pestov
- **Elementos de Estatística Computacional usando Plataformas de Software Livre/Gratuito - Alejandro C. Frery e Francisco Cribari-Neto**
- Espaços de Hardy no Disco Unitário - Gustavo Hoepfner e Jorge Hounie
- Fotografia 3D - Paulo Cezar Carvalho, Luiz Velho, Anselmo Antunes Montenegro, Adelailson Peixoto, Asla Sá e Esdras Soares
- Introdução à Teoria da Escolha - Luciano I. de Castro e José Heleno Faro
- Introdução à Dinâmica de Aplicações do Tipo Twist - Clodoaldo G. Ragazzo, Mário J. Dias Carneiro e Salvador Addas-Zanata
- Schubert Calculus: an Algebraic Introduction - Letterio Gatto
- Surface Subgroups and Subgroup Separability in 3-manifold Topology - Darren Long and Alan W. Reid
- Tópicos em Processos Estocásticos: Eventos Raros, Tempos Exponenciais e Metaestabilidade - Adilson Simonis e Cláudia Peixoto
- Topics in Inverse Problems - Johann Baumeister and Antonio Leitão
- Um Primeiro Curso sobre Teoria Ergódica com Aplicações - Krerley Oliveira
- Uma Introdução à Simetrização em Análise e Geometria - Renato H. L. Pedrosa

Distribuição:
IMPA
Estrada Dona Castorina, 110
22460-320 Rio de Janeiro, RJ
E-mail: ddic@impa.br - http://www.impa.br
ISBN: 85-244-0232-6

Sumário

Capítulo 1

Introdução

O propósito destas notas é introduzir o leitor ao uso de duas plataformas computacionais apropriadas para computação científica, notadamente para simulação estocástica, análise estatística de dados e produção de gráficos. Essas plataformas são de grande valia para o trabalho cotidiano de estatísticos, matemáticos aplicados, físicos, químicos, engenheiros, economistas e profissionais de áreas afins. Elas devem ser vistas como complementares e não como substitutas, dado que cada uma tem vantagens relativas bem definidas. Diferentemente de outras plataformas muito disseminadas (ver a referência [36] para um exemplo ilustre), tanto `R` quanto `Ox` são numericamente confiáveis e, portanto, recomendáveis até para aplicações consideradas críticas.

A linguagem de programação `Ox`, a primeira das plataformas abordadas, é uma linguagem matricial de programação com orientação a objetos que foi desenvolvida por Jurgen Doornik (Nuffield College, University of Oxford); ver `http://www.doornik.com`. Sua sintaxe é similar à da linguagem `C`, como será ilustrado através de exemplo. Ela contém uma ampla lista de implementações numéricas de grande utilidade e é distribuída gratuitamente para uso acadêmico, havendo uma versão comercial para uso não-acadêmico. Uma de suas vantagens mais marcantes é a sua eficiência. Programas bem escritos em `Ox` às vezes chegam a ser competitivos, em termos de tempo de execução, com programas escritos em linguagens de mais baixo nível, como, e.g., `C` e `FORTRAN`. A principal utilidade da linguagem

Ox, em nosso entender, reside em utilizações computacionalmente intensivas, como, e.g., simulações de Monte Carlo. Essas simulações são de grande valia na avaliação do desempenho de procedimentos estatísticos de estimação e teste em amostras de tamanho típico. Em particular, são úteis para avaliações de robustez e da qualidade de aproximações, notadamente aproximações assintóticas.

A plataforma R, por sua vez, é um ambiente para análise de dados, programação e gráficos; ver http://www.r-project.org. Ela é distribuída gratuitamente mesmo para uso não-acadêmico e seu código fonte encontra-se disponível para inspeção e alteração, se desejável. Ela é semelhante à plataforma comercial S-PLUS (http://www.insightful.com/splus), ambas sendo baseadas na linguagem S de programação, que foi desenvolvida por John Chambers e colaboradores. Sua maior utilidade, a nosso ver, reside na análise de dados e na produção de gráficos com qualidade de publicação. Uma outra virtude de R é que, por ser uma plataforma muito utilizada no meio acadêmico, existe uma grande variedade de pacotes desenvolvidos para as mais diversas aplicações; o repositório oficial destes pacotes, bem como do software, é http://www.cran.org.

Uma diferença entre as duas plataformas consideradas reside em suas formas de distribuição. Ox é distribuída gratuitamente apenas para uso acadêmico, e seu código fonte não se encontrando publicamente disponível. Por outro lado, R é *software livre*.

"Software livre" é um conceito importante no mundo da computação. Quando o software é livre, seu código fonte está universalmente disponível e pode ser livremente alterado para adaptá-lo a necessidades específicas. Assim sendo, o software livre é de fato gratuito, porém não se deve usar esta denominação para referir-se a plataformas computacionais sem custo.

O software gratuito (*freeware*) pode ser usado sem necessidade de compra ou pagamento, porém não oferece necessariamente acesso ao código fonte, por isso não pode ser alterado nem ter tal código estudado; unicamente pode-se utilizá-lo tal como foi disponibilizado. Fica, assim, estabelecida a diferença entre software livre e software gratuito.

As plataformas R e Ox, utilizadas no presente texto, são, respectivamente, software livre e gratuito só para fins acadêmicos, como mencionado acima.

Capítulo 2

Estatística Descritiva Uni- e Multivariada

2.1 Introdução ao R

R é uma linguagem e um ambiente para computação estatística e para preparação de gráficos de alta qualidade. É um projeto GNU similar à linguagem e ambiente S-PLUS e, ainda que haja diferenças significativas entre eles, grande parte do código desenvolvido para um funciona no outro.

R oferece uma grande variedade de técnicas estatísticas (modelos lineares e não-lineares, testes estatísticos clássicos, modelos de séries temporais, classificação e agrupamento, entre outros) e gráficas, e é altamente extensível.

R é uma coleção integrada de facilidades de software para manipulação de dados, realização de cálculos e preparação de gráficos, que inclui

- tratamento efetivo de dados e facilidades de armazenamento;
- operadores para cálculos em matrizes multidimensionais;
- ferramentas de diversos níveis para análises de dados;
- facilidades gráficas para análise de dados;

- uma linguagem de programação bem definida, simples e eficaz que inclui expressões condicionais, laços, funções recursivas definidas pelo usuário e recursos de entrada e saída.

Antes de começar a usar `R`, recordemos que estão disponíveis em `http://www.r-project.org`, tanto o código fonte para compilação como os executáveis já compilados para diversos sistemas operacionais. Nesse sítio encontram-se também disponíveis textos e tutoriais. Como leituras subseqüentes a este curso recomendamos os livros [16, 32, 48, 49, 50] e os artigos [14, 40]. A versão que utilizaremos para o desenvolvimento destas notas é a 2.0.1 para Linux, disponibilizada em novembro de 2004.

Uma vez que o programa esteja instalado e em execução, para sair do ambiente é necessário fornecer o comando `q()`; o sistema de ajuda em HTML é ativado com o comando `help.start()`. Se desejarmos indicar o navegador a ser utilizado (que deverá ter capacidade para processar Java), por exemplo Mozilla, poderemos fazê-lo da seguinte forma:

```
> help.start(browser="mozilla")
```

2.1.1 Primeiros passos

`R` pode ser usado como uma calculadora de grande capacidade. Vamos à seguinte sessão

```
$ R
> 2
[1] 2
> 2+2
[1] 4
> sqrt(2)
[1] 1.414214
> exp(sqrt(2))
[1] 4.11325
> sin(exp(sqrt(2)))
[1] -0.8258217
> sinh(exp(sqrt(2)))
[1] 30.56439
```

```
14 > sinh(exp(sqrt(2 - 1i*2)))
15 [1] -20.96102-6.575177i
16  > q()
17 Save workspace image? [y/n/c]: n
```

Iniciamos uma sessão (em Linux) chamando, a partir de qualquer caminho, o sistema R (linha 1). Entre as linhas 1 e 2, teremos uma saída com informações da versão do R, sua data de lançamento e outros dados (aqui omitidos). A linha 2 passa ao R uma entrada constante e R a imprime (linha 3); a saída de dados numéricos é precedida por defecto pelo indicador da linha, neste caso `[1]`, já que R supõe que pode haver mais de uma linha de dados. Na linha 4 pedimos ao R que calcule $2 + 2$, e o resultado é impresso na linha 5. Nas linhas 6, 8, 10 e 12 solicitamos a realização de outros cálculos, e seus respectivos resultados são exibidos nas linhas 7, 9, 11 e 13. R trabalha com números complexos; a unidade complexa $\sqrt{-1}$ é denotada na entrada por `1i`, e as linhas 14 e 15 mostram uma operação com complexos e seu resultado, respectivamente. Ao terminar uma sessão (linha 16) R nos perguntará se desejamos guardar as variáveis e funções definidas (linha 17) para uso futuro; se assim o fizermos, salvaremos também os comandos que foram emitidos na sessão. Se desejamos exportar os comandos para um arquivo de texto, podemos fazê-lo com `savehistory(file = "arquivo.txt")`, para depois recuperá-los com `loadhistory(file = "arquivo.txt")`.

Para ter uma idéia da capacidade gráfica do R podemos usar os seguintes comandos, que ativarão as demonstrações incluídas na distribuição básica:

```
1 > demo("graphics")
2 > demo("image")
3 > demo("persp")
4 > demo("recursion")
```

A linha 1 ativa a demonstração de algumas capacidades gráficas do R, incluindo o uso de cores. A linha 2 ativa a demonstração dos recursos de uso de imagens para visualização de dados multidimensionais. A linha 3 mostra alguns recursos do R para visualização de funções multidimensionais em perspectiva. A linha 4 mostra como R implementa um método adaptativo para calcular integrais numéricas.

2.1.2 Bibliotecas

O sistema `R` utiliza diversas bibliotecas de funções e conjuntos de dados adicionais, que são carregadas com a auxílio da função `library()`, tal como mostrado a seguir.

```
> library(cluster)
```

Essa função carrega bibliotecas já instaladas localmente. O comando

```
> install.packages()
```

abre interfaces para instalar novas bibliotecas; mais sobre esse assunto na página 13 deste texto.

No site `http://cran.r-project.org` estão disponíveis as bibliotecas oficiais.

2.1.3 Leitura e Importação de Dados

Com a instalação completa do `R` ficam disponíveis vários conjuntos de dados, e para lê-los basta utilizar a função `data()`:

```
> data(iris)
```

Dados podem ser importados dados no sistema a partir de várias fontes, como arquivos ASCII (extensão `txt` ou `csv`), bancos de dados e planilhas. Dois tipos de importação bastante utilizados são as de arquivos de tipo `txt` e `csv`. Para importar arquivos ASCII, `R` oferece duas funções interessantes: `read.table()` e `read.csv` (ver exemplo a seguir).

```
> read.table("dados.txt")
> read.csv("dados.csv", sep=";")
```

2.2 Definições

Quando trabalhamos com uma amostra de dados, ela nada mais é que uma realização (idealmente representativa) de uma população de interesse. É conveniente, para não dizer imprescindível, ter uma idéia de como se comportam os dados da amostra antes de fazer qualquer

tipo de inferência sobre a população. A estatística nos dá mecanismos para formular conjecturas sobre a população utilizando como base de inferência a amostra, por isso esta última deve ser muito bem descrita.

Para facilitar a exposição posterior apresentaremos a seguir algumas definições e notações, fazendo referência exclusivamente a quantidades amostrais. Estas quantidades são os pilares da análise estatística de dados, em suas modalidades quantitativa (resumos numéricos), qualitativa (descrições textuais) e gráfica.

Uma referência importante para este tema é o texto [31], que trata o problema específico de dados multivariados. Quando se trata de análise gráfica, os livros de Edward Tufte [43, 44, 45, 46] são uma referência importante para a preparação de gráficos, diagramas e figuras de qualidade.

Consideremos uma amostra de valores reais $\boldsymbol{y} = (y_1, \ldots, y_n)$. Um dos elementos gráficos mais importantes para descrever uma amostra é o histograma. Denotando $y_{1:n}$ e $y_{n:n}$ os valores mínimo e máximo da amostra $\boldsymbol{y}$, definamos um intervalo que inclui estes dois valores $I \supseteq [y_{1:n}, y_{n:n}]$, e seja $\mathcal{I} = \{I_0, \ldots, I_k\}$ uma partição de I. Seja x_m o ponto central de cada intervalo I_m, $0 \leq m \leq k$. O histograma é a função que a cada x_m associa o valor $H(\boldsymbol{y}, m) = \#\{1 \leq i \leq n \colon y_i \in I_m\}$, isto é, o número de observações da amostra que estão dentro do intervalo I_m. O histograma de proporções consiste em dividir os valores $H(\boldsymbol{y}, m)$ pelo tamanho da amostra, isto é, é a função que a cada x_m associa o valor $h(\boldsymbol{y}, m) = H(\boldsymbol{y}, m)/n$. A escolha da partição $\mathcal{I}$ tem enorme efeito na qualidade do gráfico.

Denotaremos por $y_{1:n} \leq y_{2:n} \leq \cdots \leq y_{n-1:n} \leq y_{n:n}$ os elementos do vetor $\boldsymbol{y}$ ordenados em forma não-decrescente.

Segundo [8], uma análise quantitativa elementar deve conter, pelo menos, as seguintes quantidades tabuladas e analisadas:

- Descrição geral:
  - Tamanho da amostra n
  - Valores mínimo $y_{1:n}$ e máximo $y_{n:n}$
- Medidas de tendência central:
  - Média amostral $\bar{\boldsymbol{y}} = n^{-1} \sum_{i=1}^{n} y_i$

  - Mediana amostral $q_{1/2}(\boldsymbol{y}) = y_{(n+1)/2:n}$ se n é ímpar ou $q_{1/2}(\boldsymbol{y}) = (y_{n/2:n} + y_{n/2+1:n})/2$ caso contrário
  - Moda: as abscissas x_t onde $h(\boldsymbol{y}, t) \geq h(\boldsymbol{y}, v)$ para todo $v \neq t$; em caso de haver uma única moda, enfatizá-la

- Medidas de dispersão:

  - Variância amostral $s^2(\boldsymbol{y}) = n^{-1} \sum_{i=1}^{n} (y_i - \bar{y})^2$; a distinção entre o uso de n^{-1} ou de $(n-1)^{-1}$ é irrelevante para tamanhos de amostra razoáveis (por exemplo, superiores a trinta)
  - Desvio padrão amostral $s(\boldsymbol{y}) = \sqrt{s^2}$
  - Desvio médio absoluto $n^{-1} \sum_{i=1}^{n} |y_i - \bar{y}|$
  - Desvio mediano absoluto $\operatorname{mad}(\boldsymbol{y}) = q_{1/2}(\boldsymbol{z})$, onde $\boldsymbol{z} = (|y_i - q_{1/2}(\boldsymbol{y})|)_{1 \leq i \leq n}$
  - Distância interquartil $\operatorname{IQR}(\boldsymbol{y}) = k(y_{[3n/4:n]} - y_{[n/4:n]})$, onde os colchetes denotam o inteiro mais próximo e a constante k se ajusta para cada situação

  É conveniente notar que as quatro últimas medidas estão na mesma escala dos dados, sendo que a primeira está em escala quadrática.

- Estatísticas de ordem superior:

  - Assimetria amostral $\hat{\gamma}_1 = n^{-1} \sum_{i=1}^{n} (y_i - \bar{y})^3 / s^3$
  - (Excesso de) Curtose amostral ou coeficiente de curtose amostral $\hat{\gamma}_2 = n^{-1} \sum_{i=1}^{n} (y_i - \bar{y})^4 / s^4 - 3$

Analogamente ao histograma, podemos visualizar o gráfico *stem-and-leaf* (vástago e folha?, caule e folha?). Expressemos os valores do vetor de observações $\boldsymbol{y}$ na forma de e dígitos decimais $d_1 d_2 \ldots d_e$; por exemplo, se $e = 4$, ao valor $4,53$ se agregará um zero a esquerda e trabalharemos com $4,530$. Escrevamos duas colunas: a primeira tem os $e-1$ primeiros dígitos das entradas (sem repetições) e na segunda uma entrada com o dígito restante por cada valor que tenha os primeiros $e-1$ dígitos. Um exemplo deste recurso é mostrado na página 15.

Uma forma de representação gráfica interessante e que pode complementar a informação revelada pelo histograma é o *Boxplot*, ou *Box-and-whisker plot*. Dada a amostra $\boldsymbol{y}$, o Boxplot mostra uma caixa do tamanho da distância interquartil $y_{[3n/4:n]} - y_{[n/4:n]}$, com uma barra interna na posição da mediana. São identificadas as observações potencialmente surpreendentes (*outliers*), são desenhadas como pontos isolados e são removidas da amostra. Uma vez retirados os outliers, são agregados segmentos denotando os valores mínimo e máximo restantes. O Boxplot é particularmente útil para comparar várias amostras em um mesmo gráfico. Exemplos deste recurso são mostrados nas Figuras 2.1 e 2.2.

2.3 Amostras Univariadas

A plataforma `R` oferece diversas funções para o cálculo de estatísticas descritivas, como a média, a mediana, estatísticas de ordem, medidas de dispersão, assimetria e curtose. Para ilustrar o uso destas funções será utilizado o conjunto de dados `iris`, disponível no `R`. Este conjunto de dados consiste em 151 linhas com seis colunas cada uma. A primeira linha, de tipo texto, descreve o conteúdo de cada coluna. As cinco primeiras colunas correspondem a medidas realizadas sobre flores, e a última, que é de tipo texto, categoriza em uma de três espécies cada flor medida.

A primeira coluna está rotulada `Sepal.Length`; para ver os valores basta emitir o seguinte comando:

```
> iris$Sepal.Length
  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8
 [13] 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1
 [25] 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
 [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6
 [49] 5.3 5.0 7.0 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2
 [61] 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
 [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0
 [85] 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7
 [97] 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
[109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0
[121] 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9
```

```
[133] 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
[145] 6.7 6.7 6.3 6.5 6.2 5.9
```

Se queremos ter acesso às variáveis diretamente, sem necessidade de fazer referência ao conjunto de dados (`iris`), podemos colocar as variáveis na lista de objetos definidos com o comando

```
> attach(iris)
```

Para calcular a média amostral da variável `Sepal.Length` basta fazer

```
> mean(Sepal.Length)
[1] 5.843333
```

A mediana amostral é obtida com

```
> median(Sepal.Length)
[1] 5.8
```

Para calcular os quartis fazemos

```
> quantile(Sepal.Length)
  0%  25%  50%  75% 100%
 4.3  5.1  5.8  6.4  7.9
```

A função `quantile()` admite como argumento opcional um vetor de valores no intervalo $[0, 1]$, retornando os percentis da amostra nesses pontos. Se, por exemplo, queremos calcular os decis deveríamos entrar `quantile(iris$Sepal.Length, v)`, onde `v` é o vetor que contém os valores $(i/10)_{1 \leq i \leq 9}$. Podemos fazê-lo manualmente, ou utilizar uma função do `R` para gerar este vetor auxiliar.

```
> quantile(Sepal.Length, seq(.1,.9,.1))
 10%  20%  30%  40%  50%  60%  70%  80%  90%
4.80 5.00 5.27 5.60 5.80 6.10 6.30 6.52 6.90
```

Já que usaremos este vetor várias vezes, é conveniente guardá-lo em uma variável de nome mais curto e manejável com o comando

```
> l_s <- Sepal.Length
```

As últimas versões do R admitem "=" como comando de atribuição, em vez do mais exótico (porém mais utilizado, até agora) "<-".

R também oferece funções para calcular medidas de dispersão como variância, desvio padrão e desvio médio absoluto, tal como é mostrado a seguir.

```
> var(l_s)
[1] 0.6856935
> sd(l_s)
[1] 0.8280661
> mad(l_s)
[1] 1.03782
```

O máximo, o mínimo e o tamanho da amostra podem ser obtidos com

```
> max(l_s)
[1] 7.9
> min(l_s)
[1] 4.3
> length(l_s)
[1] 150
```

Para calcular estatísticas de ordem superior, como assimetria e curtose, é necessário carregar o pacote `e1071`, que provê as funções `skewness()` e `kurtosis()`.

```
1 > install.packages("e1071")
2 > library(e1071)
3 > skewness(l_s)
4 [1] 0.3086407
5 > kurtosis(l_s)
6 [1] -0.6058125
```

A linha 1 é necessária para baixar uma biblioteca que não está disponível localmente. R usará a conexão a Internet para obtê-la. Se o comando é dado para uma biblioteca já instalada, R verificará se há uma versão mais atual e, se houver, a instalará.

R permite construir gráficos com facilidade. Por exemplo, para construir um `boxplot` é necessário apenas emitir o comando

```
> boxplot(l_s, horizontal=T)
```

Seu resultado é mostrado na Figura 2.1.

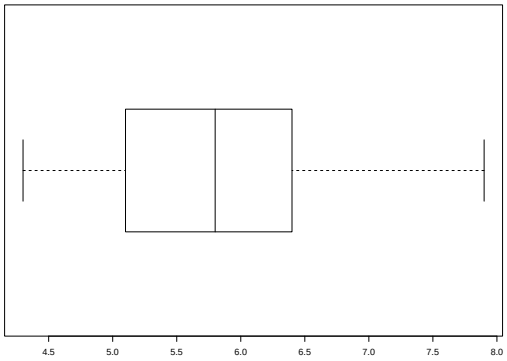

Figura 2.1: Boxplot dos dados `Sepal.Length`.

De fato, para gerar o arquivo que armazena o gráfico mostrado na Figura 2.1 é necessário ativar o dispositivo de saída, fazer o gráfico e desativar o dispositivo. A seqüência de instruções é

```
> postscript("box_plot.eps")
> boxplot(l_s, horizontal=T)
> dev.off()
```

Tal como comentamos anteriormente, o Boxplot é particularmente útil para realizar uma comparação visual rápida entre várias amostras. Para isso, basta emitir o comando com os nomes das amostras separadas por comas; em nosso caso

```
boxplot(Sepal.Length, Sepal.Width, Petal.Length,
 Petal.Width, horizontal=T, names=names(iris)[1:4])
```

que nos dá como resultado o gráfico mostrado na Figura 2.2.

Outro gráfico importante é o histograma, cuja versão mais simples pode ser construída com o seguinte comando:

```
> hist(Petal.Length, main="", freq=FALSE,
   xlab="Largura de Pétalas", ylab="Proporções")
```

e seu resultado pode ser visto na Figura 2.3. R oferece uma grande variedade de parâmetros para controlar o aspecto com que os histogramas em particular, e todos os gráficos em geral, são produzidos e exibidos.

O diagrama *stem-and-leaf* é obtido a partir do comando

```
> stem(Petal.Length)

  The decimal point is at the |

  1 | 01223333333444444444444
  1 | 55555555555556666666777799
  2 |
  2 |
  3 | 033
  3 | 55678999
  4 | 000001112222334444
  4 | 55555555666777778888999999
  5 | 000011111111223344
  5 | 55566666677788899
  6 | 0011134
  6 | 6779
```

2.4 Amostras Multivariadas

R trata com facilidade dados multivariados, isto é, onde para cada indivíduo temos um vetor de observações. A notação que utilizaremos para denotar um conjunto de n vetores k-dimensionais é $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$, com $y_i \in \mathbb{R}^k$. Este tipo de dados aparece naturalmente em estudos onde se mede mais de um atributo para cada indivíduo como, por exemplo, em antropometria onde se registram o peso, a estatura, a idade e diversas medidas corporais de cada pessoa. Este tipo de análise está recebendo atualmente muita atenção, já que é um passo importante na cadeia de operações conhecida como *KDD – Knowledge Discovery in Databases*.

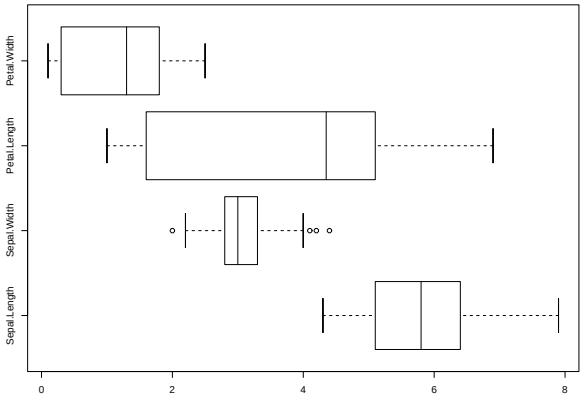

Figura 2.2: Boxplots das quatro variáveis.

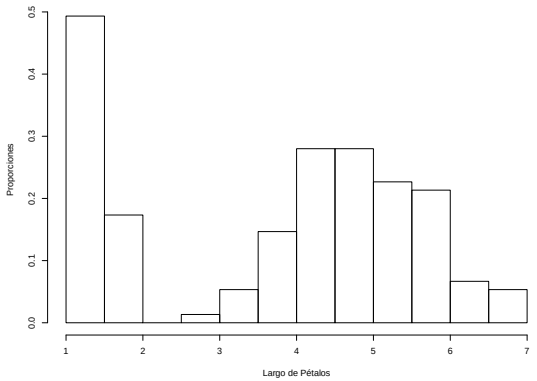

Figura 2.3: Histograma dos comprimentos de sépalas.

Para obter uma visão geral de um conjunto de dados deste tipo podemos emitir o seguinte comando

```
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length
 Min.   :4.300   Min.   :2.000   Min.   :1.000
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600
 Median :5.800   Median :3.000   Median :4.350
 Mean   :5.843   Mean   :3.057   Mean   :3.758
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100
 Max.   :7.900   Max.   :4.400   Max.   :6.900
   Petal.Width        Species
 Min.   :0.100   setosa    :50
 1st Qu.:0.300   versicolor:50
 Median :1.300   virginica :50
 Mean   :1.199
 3rd Qu.:1.800
 Max.   :2.500
```

A matriz de covariância descreve relações entre variáveis, assim como sua variância:

```
> var(iris[1:150, 1:4])
             Sepal.Length Sepal.Width Petal.Length
Sepal.Length   0.68569351 -0.04243400    1.2743154
Sepal.Width   -0.04243400  0.18997942   -0.3296564
Petal.Length   1.27431544 -0.32965638    3.1162779
Petal.Width    0.51627069 -0.12163937    1.2956094
              Petal.Width
Sepal.Length    0.5162707
Sepal.Width    -0.1216394
Petal.Length    1.2956094
Petal.Width     0.5810063
```

Nota-se que eliminamos a última coluna, que não contêm valores reais mas rótulos. Analogamente, é possível obter a matriz de correlações:

```
> cor(iris[1:150, 1:4])
             Sepal.Length Sepal.Width Petal.Length
Sepal.Length    1.0000000  -0.1175698    0.8717538
```

```
Sepal.Width    -0.1175698   1.0000000   -0.4284401
Petal.Length    0.8717538  -0.4284401    1.0000000
Petal.Width     0.8179411  -0.3661259    0.9628654
             Petal.Width
Sepal.Length    0.8179411
Sepal.Width    -0.3661259
Petal.Length    0.9628654
Petal.Width     1.0000000
```

Um gráfico muito interessante para se ver simultaneamente o comportamento de todos os pares de variáveis de um conjunto multivariado é o diagrama de pares, que é obtido com

```
> pairs(iris)
```

e é mostrado na Figura 2.4

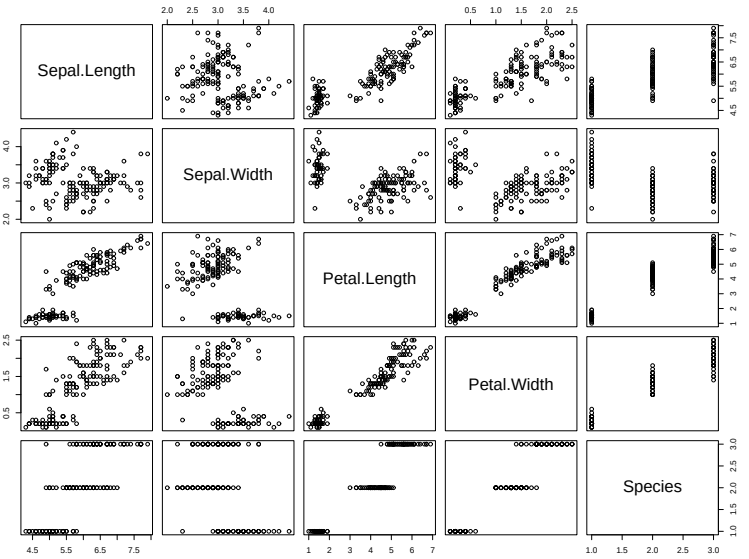

Figura 2.4: Diagrama de pares para os dados iris.

Nota-se que a coluna de espécies foi transformada em uma entrada numérica, e que não é muito interessante visualizá-la como se contivesse dados. Podemos aproveitá-la para rotular os pontos com cores diferentes, fazendo

```
> pairs(iris[1:4], pch=21,
     bg = c("red", "green", "blue")[unclass(Species)])
```

O resultado é mostrado na Figura 2.5. A função `unclass` transforma classes em atributos numéricos, que por sua vez são utilizados como índices para as cores.

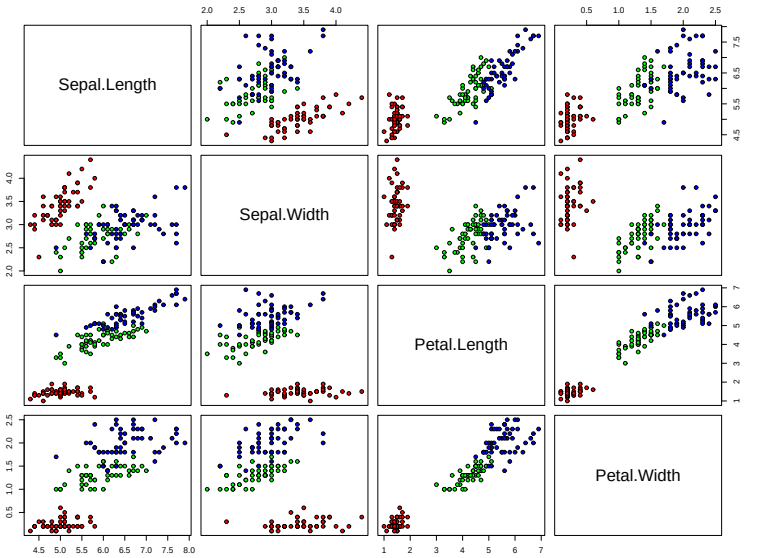

Figura 2.5: Diagrama de pares rotulados com cores.

A função `stars()` também é muito utilizada:

```
> stars(iris, nrow=13, key.loc=c(23,0))
```

O resultado é mostrado na Figura 2.6.

As variações possíveis para estes gráficos são, também, muitas.

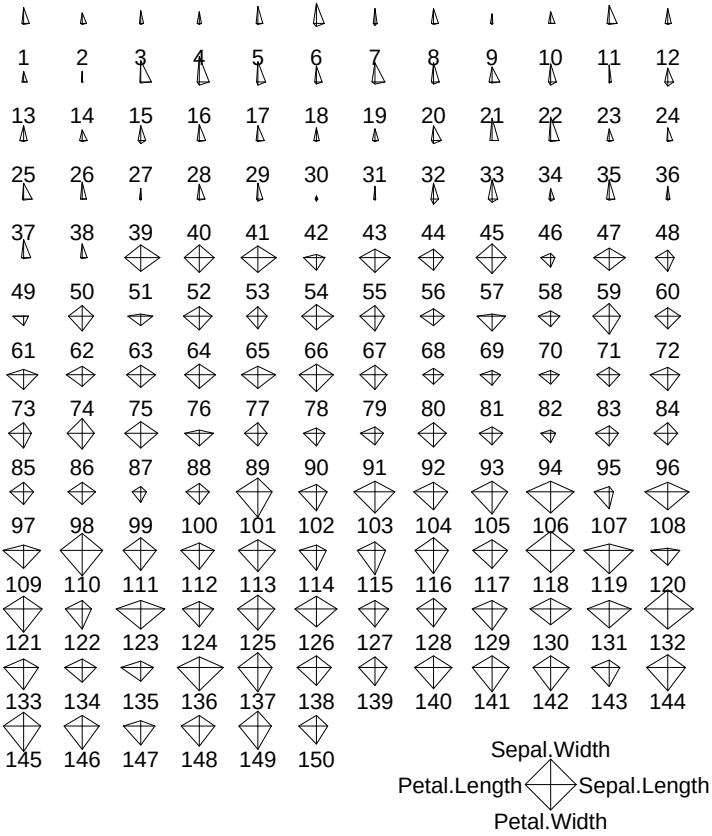

Figura 2.6: Diagrama de estrelas a partir do conjunto de dados iris.

Capítulo 3

Inferência pelo Método de Substituição e Solução de Sistemas de Equações não Lineares

3.1 Introdução a Plataforma Ox

Ox é uma linguagem de programação matricial orientada a objetos que, utilizando uma sintaxe muito parecida com as de C e de C++, oferece uma enorme gama de recursos matemáticos e estatísticos. Para a preparação deste curso utilizou-se a versão 3.40 para Linux (para mais detalhes ver [15, 20]).

Do ponto de vista da precisão numérica, Ox é uma das mais confiáveis plataformas para computação científica. A versão que não oferece interface gráfica está disponível gratuitamente para uso acadêmico e de pesquisa. Ox está organizado em um núcleo básico e em bibliotecas adicionais. É possível chamar funções de Ox a partir de programas externos, bem como ter acesso a executáveis compilados externamente ao Ox.

Um primeiro programa em Ox poderia ser o seguinte:

```
#include <oxstd.h> // include Ox standard library header
main() // function main is the starting point
{
 decl m1, m2; // declare two variables, m1 and m2
 m1 = unit(3);// assign to m1 a 3 x 3 identity matrix
 m1[0][0] = 2; // set top-left element to 2
 m2 = <0,0,0;1,1,1>; //m2 is a 2 x 3 matrix, the first
                     // row consists of zeros, the
                     // second of ones
 print("two matrices", m1, m2); // print the matrices
}
```

Ao executá-lo, teremos como saída

```
frery@frery$ oxl primero

Ox version 3.40 (Linux) (C) J.A. Doornik, 1994-2004
two matrices
       2.0000       0.0000       0.0000
       0.0000       1.0000       0.0000
       0.0000       0.0000       1.0000

       0.0000       0.0000       0.0000
       1.0000       1.0000       1.0000
```

A fim de ilustrar a similaridade de sintaxes entre `C` e `Ox`, veremos a seguir o exemplo apresentado em [15], onde são comparados programas com o mesmo propósito (gerar uma tabela de equivalência entre graus Celsius e Fahrenheit). Esta similaridade de sintaxes é, de fato, uma vantagem da linguagem `Ox`; conhecimento de `C` auxilia sobremaneira no aprendizado de `Ox` e, para aqueles que não têm domínio de `C`, o aprendizado de `Ox` conduz a uma familiaridade inicial com a linguagem `C`.

Primeiramente, o código `C`:

```
1 /*********************************************
2  * PROGRAM: celsius.c
3  *
4  * USAGE: To generate a conversion table of
5  *         temperatures (from Fahrenheit to Cel
```

```
 *         sius). Based on an example in the
 *         Kernighan & Ritchie's book.
 *
 ********************************************/

#include <stdio.h>

int main(void)
{
   int fahr;

   printf( "\nConversion table (F to C)\n\n" );
   printf( "\t%3s %5s\n", "F", "C" );

   /* Loop over temperatures */
   for(fahr = 0; fahr <= 300;  fahr += 20)
{
        printf( "\t%3d %6.1f\n", fahr, 5.0*(fahr
               -32)/9.0 );
}

   printf( "\n" );

   return 0;
}
```

e a sua saída depois de compilado em ambiente Linux usando o compilador gcc:

```
Conversion table (F to C)

          F      C
          0  -17.8
         20   -6.7
         40    4.4
         60   15.6
         80   26.7
        100   37.8
        120   48.9
```

```
11          140    60.0
12           160    71.1
13          180    82.2
14          200    93.3
15          220   104.4
16          240   115.6
17          260   126.7
18          280   137.8
19          300   148.9
```

A seguir o código equivalente em Ox

```
1  /*********************************************
2   * PROGRAM: celsius.ox
3   *
4   * USAGE: To generate a conversion table of
5   *         temperatures (from Fahrenheit to Cel
6   *         sius). Based on an example in the
7   *         Kernighan & Ritchie's book.
8   *********************************************/
9
10 #include <oxstd.h>
11
12 main()
13 {
14     decl fahr;
15
16     print( "\nConversion␣table␣(F␣to␣C)\n\n" );
17     print( "\t␣␣F␣␣␣␣␣␣␣␣C\n" );
18
19     // Loop over temperatures
20     for(fahr = 0; fahr <= 300;  fahr += 20)
21 {
22         print( "\t", "%3d", fahr );
23         print( "␣␣␣␣", "%6.1f", 5.0*(fahr-32)
24                /9.0, "\n" );
25 }
26
27     print( "\n" );
```

```
}
```

e a sua saída

```
Ox version 3.40 (Linux) (C) J.A. Doornik, 1994-
2004

Conversion table (F to C)

           F        C
           0    -17.8
          20     -6.7
          40      4.4
          60     15.6
          80     26.7
         100     37.8
         120     48.9
         140     60.0
         160     71.1
         180     82.2
         200     93.3
         220    104.4
         240    115.6
         260    126.7
         280    137.8
         300    148.9
```

Nos documentos de ajuda incluídos com as diversas distribuições do `Ox` existe uma grande variedade de exemplos, assim como na detalhada documentação que acompanha esta plataforma.

3.2 Modelos Estatísticos Paramétricos

Os modelos estatísticos são referenciais teóricos que são utilizados para descrever fenômenos. Os fenômenos naturais são, em sua maioria, excessivamente complexos para que possamos extrair informação útil a partir de sua observação direta. Os modelos são simplificações desta realidade que, ao perder detalhes e buscar um certo grau de

generalização, aspiram a ajudar-nos a formular leis de certa validade. Neste trabalho trataremos exclusivamente de modelos estatísticos.

Um modelo estatístico paramétrico é uma família de distribuições de probabilidade indexadas (determinadas) por um vetor p dimensional $\boldsymbol{\theta}$ sobre o qual só sabemos que pertence a um conjunto $\Theta \subset \mathbb{R}^p$. Os dados nos servirão para termos uma idéia do valor parâmetro $\boldsymbol{\theta}$.

A literatura é vasta em modelos estatísticos, mais ou menos adequados para certas situações. Referências importantes para este tema são os textos [26, 27, 28]. Mencionaremos a seguir somente uns poucos modelos que aparecem freqüentemente em aplicações.

A variável aleatória não trivial mais simples é a que pode adotar só dois valores: 1, com probabilidade $0 \leq p \leq 1$, e 0 com probabilidade $1 - p$. Dizemos que esta variável aleatória tem distribuição Bernoulli com probabilidade p de êxito. Para este e outros conceitos de probabilidade, recomendamos o texto [25].

A distribuição da soma de m variáveis aleatórias independentes e identicamente distribuídas, cada uma com distribuição Bernoulli com probabilidade p de êxito, é uma variável aleatória que pode adotar $n + 1$ valores, $0 \leq k \leq n$, cada um com probabilidade

$$\Pr(Y = k) = \binom{nk}{p}^k (1-p)^{n-k}, \tag{3.1}$$

onde $\binom{n}{k} = n!/(k!(n-k)!)$. Diremos que a variável aleatória Y obedece distribuição binomial com parâmetros n e p.

A média e a variância de uma variável aleatória com distribuição binomial com parâmetros n e p são, respectivamente, np e $np(1-p)$. É imediato que uma variável aleatória com distribuição binomial com parâmetros $n = 1$ e p segue distribuição Bernoulli com probabilidade de êxito p.

Consideremos uma situação onde um bom modelo para as observações é a distribuição binomial. Suponhamos que a probabilidade p de êxito individual seja muito pequena, com a qual a probabilidade de observar qualquer evento distinto de zero será, também, muito pequena. Para compensar esta situação, suponhamos que sejam realizadas muitas observações (repetições) independentes, isto é, que n seja grande. É possível provar, usando somente ferramentas analíticas,

que

$$\lim_{\substack{p\to 0\\ n\to\infty\\ np\to\theta}} \Pr(Y=k) = \lim_{\substack{p\to 0\\ n\to\infty\\ np\to\theta}} \binom{n}{k} p^k (1-p)^{n-k} = \frac{\theta^k}{k!} e^{-\theta}.$$

Esta lei de probabilidade é denominada distribuição de Poisson com parâmetro $\theta > 0$. Uma variável aleatória que obedece a distribuição de Poisson com parâmetro θ tem média e variância iguais a θ.

As distribuições mencionadas até agora são discretas, isto é, os valores que as variáveis aleatórias cuja distribuição está caracterizada por elas são finitos ou, como máximo, contáveis (numeráveis). A seguir veremos distribuições contínuas, onde esses valores não são contáveis.

A distribuição uniforme sobre o intervalo (a, b) é aquela que a cada intervalo $(b, c) \subset (a, b)$ atribui probabilidade

$$\Pr(Y \in (b, c)) = \frac{c-b}{b-a}.$$

Para o caso particular $a = 0$ tem-se que a esperança de uma variável aleatória com esta distribuição é $b/2$ e sua variância é $b/12$.

Uma variável aleatória Y com distribuição normal ou gaussiana de média $\mu \in \mathbb{R}$ e variância $\sigma^2 > 0$ tem sua distribuição caracterizada pela densidade

$$f(y; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right). \tag{3.2}$$

Denota-se $Y \sim N(\mu, \sigma^2)$. Na plataforma `R` temos esta densidade disponível através da função `dnorm`, só que parametrizada pelo desvio padrão σ.

A variável aleatória Y segue uma lei gama com parâmetros $\alpha, \beta > 0$ se sua densidade é dada por

$$f(y; \alpha, \beta) = \frac{1}{\beta^\alpha \Gamma(\alpha)} y^{\alpha-1} \exp(-y/\beta) \mathbb{I}_{\mathbb{R}_+}(y), \tag{3.3}$$

onde $\mathbb{I}_A$ denota a função indicadora do conjunto A. Esta situação denota-se $Y \sim \Gamma(\alpha, \beta)$. Esta densidade está disponível na plataforma `R` através da função `dgamma`. A esperança de uma variável aleatória com esta distribuição é $\alpha\beta$, sua variâcia sendo $\alpha\beta^2$.

A variável aleatória Y segue uma lei triangular com parâmetro $\alpha > 0$ se a sua densidade é dada por

$$f(y;\alpha) = \begin{cases} 0 & \text{se} \quad y < -\alpha \\ \alpha^{-1}(1+\alpha^{-1}y) & \text{se} \quad -\alpha \leq y < 0 \\ \alpha^{-1}(1-\alpha^{-1}y) & \text{se} \quad 0 \leq y \leq \alpha \\ 0 & \text{se} \quad y > \alpha. \end{cases} \tag{3.4}$$

A sua função de distribuição acumulada é dada por

$$F(y;\alpha) = \begin{cases} 0 & \text{se} \quad y < -\alpha \\ \frac{(\alpha+y)^2}{2\alpha^2} & \text{se} \quad -\alpha \leq y < 0 \\ \frac{1}{2}\left(1 - \frac{y(y-2\alpha)}{\alpha^2}\right) & \text{se} \quad 0 \leq y \leq \alpha \\ 1 & \text{se} \quad y > \alpha. \end{cases} \tag{3.5}$$

A inversa da função de distribuição acumulada é dada por

$$F^{-1}(u;\alpha) = \begin{cases} \alpha\left(\sqrt{2u} - 1\right) & \text{se} \quad 0 < u \leq 1/2 \\ \alpha\left(1 - \sqrt{2\,(1-u)}\right) & \text{se} \quad 1/2 < u \leq 1 \end{cases} \tag{3.6}$$

A variável aleatória Y segue uma lei de Weibull-Gnedenko com parâmetros $\alpha \neq 0$ e $\beta > 0$ se a sua densidade é dada por

$$f(y;\alpha,\beta) = |\alpha|\,\beta y^{\alpha-1}\exp(-\beta y^{\alpha})\mathbb{I}_{\mathbb{R}_+}(y). \tag{3.7}$$

Esta situação é denotada $Y \sim \mathcal{W}(\alpha,\beta)$.

A variável aleatória Y segue uma lei Erlang com parâmetro $\alpha \in \mathbb{N}$ se a sua densidade é dada por

$$f(y;\alpha) = \frac{1}{\Gamma(\alpha)} y^{\alpha-1}e^{-y}\mathbb{I}_{\mathbb{R}_+}(y). \tag{3.8}$$

É possível ver que a sua função de distribuição acumulada é

$$F(y;\alpha) = 1 - e^{-y}\left(1 + \sum_{1\leq i\leq \alpha-1} \frac{y^i}{i!}\right). \tag{3.9}$$

3.3 O Problema de Inferência

A tarefa de fazer inferência consiste em, dado um conjunto de n observações reais $\boldsymbol{y} = (y_1, \ldots, y_n)$ e aceitando que elas são eventos de variáveis aleatórias cuja distribuição é conhecida a menos do parâmetro $\boldsymbol{\theta}$, estimar o valor deste parâmetro.

Na literatura estatística existem diversos métodos para cumprir com esta tarefa, cada um com vantagens e desvantagens. Os textos [2, 4] são referências de excelente nível para este problema.

Uma *estatística* $T_n \equiv T_n(Y_1, \ldots, Y_n)$ é qualquer função das variáveis aleatórias $Y_1, \ldots, Y_n$ que descrevem os dados. Diremos que uma estatística utilizada para estimativa de um parâmetro desconhecido $\boldsymbol{\theta}$ é um *estimador* de $\boldsymbol{\theta}$, e neste trabalho denotaremos estimadores por $\widehat{\boldsymbol{\theta}}$ (se houvesse necessidade de trabalhar com mais de um estimador simultaneamente, utilizaríamos notações do tipo $\widetilde{\boldsymbol{\theta}}$, $\breve{\boldsymbol{\theta}}$, $\check{\boldsymbol{\theta}}$ etc.). Um estimador é sempre uma variável aleatória, visto que é uma função das variáveis aleatórias $Y_1, \ldots, Y_n$. Quando $Y_1, \ldots, Y_n$ são independentes e oriundas da mesma distribuição dizemos que as observações são independentes e identicamente distribuídas (i.i.d.). Uma *estimativa*, por outro lado, não é uma variável aleatória já que é o resultado de calcular um estimador em uma amostra observada $y_1, \ldots, y_n$.

Diversos estimadores podem ser comparados através de certas características de interesse. Uma propriedade importante é a de ser *não viesado*; um estimador $\widehat{\boldsymbol{\theta}}$ é não viesado para $\boldsymbol{\theta}$ se $\mathrm{E}(\widehat{\boldsymbol{\theta}}) = \boldsymbol{\theta}$ para todos os valores de $\boldsymbol{\theta}$ no espaço paramétrico Θ. Em outras palavras, o estimador se iguala em média ao parâmetro desconhecido que desejamos estimar. O viés de um estimador é definido, por outro lado, como a diferença entre seu valor esperado e o parâmetro desconhecido. Esta propriedade pode ser estudada no limite dizendo que um estimador é assintoticamente não-viesado se

$$\lim_{n\to\infty} \mathrm{E}(\widehat{\boldsymbol{\theta}}_n) = \boldsymbol{\theta}, \quad \forall \boldsymbol{\theta} \in \Theta,$$

onde o subscrito n deixa explícita a dependência do estimador no tamanho da amostra.

Uma segunda propriedade importante de estimadores pontuais é a *consistência*. Um estimador $\widehat{\boldsymbol{\theta}}_n$ do parâmetro $\boldsymbol{\theta}$ é consistente se $\widehat{\boldsymbol{\theta}}_n$

converge em probabilidade para $\boldsymbol{\theta}$, denotada $\widehat{\boldsymbol{\theta}}_n \stackrel{\Pr}{\to} \boldsymbol{\theta}$, isto é, se para todo $\epsilon > 0$ vale que

$$\lim_{n\to\infty} \Pr(|\widehat{\boldsymbol{\theta}}_n - \boldsymbol{\theta}| > \epsilon) = 0,$$

ou, equivalentemente, vale que

$$\lim_{n\to\infty} \Pr(|\widehat{\boldsymbol{\theta}}_n - \boldsymbol{\theta}| \leq \epsilon) = 1.$$

É importante reforçar que estimadores não viesados não são necessariamente estimadores consistentes para, com isso, enfatizar que a qualidade de um estimador pode ser medida de diversas maneiras e que estas não são necessariamente coincidentes.

3.4 Método de Substituição

O método de substituição consiste em resolver sistemas de equações formadas, por um lado, por esperanças de funções da variável aleatória que modela os dados (estas funções devem depender de forma interessante do parâmetro desconhecido) e, por outro lado, das versões amostrais destas funções. Para se ter estimadores bem determinados devem ser formadas tantas equações independentes entre si como parâmetros desconhecidos.

Este método apoia-se na lei dos grandes números que diz que, sob certas condições,

$$\frac{1}{n}\sum_{1\leq i\leq n} \Psi(Y_i) \stackrel{\Pr}{\to} \mathrm{E}_{\boldsymbol{\theta}}[\Psi(Y)],$$

onde $Y, Y_1, \ldots, Y_n$ é uma seqüência de variáveis aleatórias i.i.d. e Ψ é uma função mensurável. Se o parâmetro desconhecido tem a forma $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$, então o método de substituição consiste em estimar $\boldsymbol{\theta}$ através de $\widehat{\boldsymbol{\theta}} = (\widehat{\theta}_1, \ldots, \widehat{\theta}_p)$, que é a solução do sistema

$$\left\{\begin{array}{rcl} \dfrac{1}{n}\displaystyle\sum_{1\leq i\leq n} \Psi_1(y_i) & = & \mathrm{E}_{\boldsymbol{\theta}}[\Psi_1(Y)], \\ \vdots & & \vdots \\ \dfrac{1}{n}\displaystyle\sum_{1\leq i\leq n} \Psi_p(y_i) & = & \mathrm{E}_{\boldsymbol{\theta}}[\Psi_p(Y)]. \end{array}\right. \tag{3.10}$$

Uma referência importante para esta técnica é o livro [33].

Ainda que o método de substituição (também conhecido como método de analogia) seja geral em sua formulação, sua versão mais popular é baseada nos momentos amostrais. Quando o lado direito das equações do sistema dado em (3.10) são momentos, o método é conhecido como método de momentos.

Tomemos como exemplo a distribuição gama, caracterizada pela densidade apresentada na equação (3.3). Sua esperança é $\alpha\beta$ e seu segundo momento é $\alpha\beta^2(1+\alpha)$. O sistema de equações que podemos formar com esta informação é

$$\left\{\begin{array}{rcl} \dfrac{1}{n}\displaystyle\sum_{1\leq i\leq n} x_i - \widehat{\alpha}\widehat{\beta} & = & 0, \\ \dfrac{1}{n}\displaystyle\sum_{1\leq i\leq n} x_i^2 - \widehat{\alpha}\widehat{\beta}^2(1+\widehat{\alpha}) & = & 0, \end{array}\right. \tag{3.11}$$

que requer resolver um sistema de equações não-lineares. Esta é a situação geral de estimativa pelo método de substituição, formulado na equação (3.10); veremos a seguir como resolvê-lo na plataforma `Ox`.

3.5 Solução Algorítmica de Sistemas de Equações Não-lineares

O código fonte a seguir tem como propósitos:

1. gerar uma amostra de tamanho `t_amostra` de eventos de variáveis aleatórias i.i.d. que seguem a distribuição gama com parâmetros $\alpha =$ `p_a` e $\beta =$ `p_b`;

2. calcular $\widehat{(\alpha,\beta)} = (\widehat{\alpha},\widehat{\beta})$, o estimador de (α,β) baseado no primeiro e segundo momentos amostrais;

3. exibir o resultado na saída padrão.

```
1 #include <oxstd.h>
2 #include <oxprob.h> // rotinas de probabilidade
3 #import <solvenle> // resolucao de sistemas de equacoes
```

```
4                       // nao-lineares
5
6  decl g_m1, g_m2; // declara as variaveis que armazenarao
7                   // os momentos de primera e segunda ordem
8    // variaveis globais para que sejam vistas pelas funcoes
9    // a resolver
10
11 sist_ec_gama12(const avF, const vX) {
12   decl alfa, beta;
13
14   alfa = fabs(vX[0]);
15   beta = fabs(vX[1]);
16
17   avF[0] = (g_m1 - alfa * beta | g_m2 - alfa
18            * (beta .^ 2.) * (1. + alfa));
19
20   return 1;
21 }
22
23 main() //
24 {
25  decl t_amostra = 10000;     // declara e atribui valor ao
26                              // tamanho da amostra
27  decl p_a = 10., p_b = .01; // declara e atribui valores
28                              // aos parametros verdadeiros
29  decl v_amostra; // declara o vetor que armazenara a
30                  // amostra gerada
31  decl v_solucao = <1;1>;
32
33  ranseed("LE");
34  ranseed({2,11,111,1111});
35
36  v_amostra = rangamma(t_amostra, 1, p_a, 1. / p_b);
37                                      // gera eventos
38    // cuidado com a parametrizacao de Ox
39    // armazenamendo e processamento por filas pode ser
40    // mais rapido
41    // dados armazenados numa coluna
42
43  g_m1 = meanc(v_amostra); // media das colunas
44  g_m2 = meanc(v_amostra .^ 2.); // media do quadrado das
45                                 // colunas
46
47  SolveNLE(sist_ec_gama12, &v_solucao);
48  println("Solucao=", fabs(v_solucao));
49 }
```

Alguns pontos a serem comentados deste programa são:

l. 25-31 `Ox` exige que toda variável a ser utilizada seja declarada; note que não se diz o tipo, já que o mesmo é dinâmico e depende das atribuições feitas a cada variável.

l. 36 A função `rangamma` gera um vetor de `t_amostra` linhas e `1` coluna de eventos de variáveis aleatórias i.i.d. com distribuição gamma e parâmetros $\alpha =$ `p_a` e $\beta = 1/$`p_b`. Como esta, existem funções que geram eventos de variáveis aleatórias de diversas distribuições interessantes (beta, binomial, Cauchy, gaussiana inversa generalizada etc.); todas elas têm prefixo '`ran`' e um sufixo que lembra a lei.

l. 43-44 Um dos pontos fortes do `Ox` é sua orientação a matrizes. Um exemplo disto é a função `meanc`, que admite como argumento uma matriz de m linhas e n colunas, e retorna um vetor de dimensão n onde cada elemento é a média dos m elementos de cada coluna da matriz de entrada. Note que na linha 44 passamos como entrada da função `meanc` o vetor formado pelo quadrado de cada elemento do vetor `v_amostra`.

l. 47 É o núcleo central do programa. Chamamos a função `SolveNLE` com dois argumentos obrigatórios: a função que implementa o sistema de equações que queremos resolver (`sist_ec_gama12`) e o endereço (por isso o uso de '`&`') de um vetor que, na entrada, tem a solução inicial e, na saída, terá a solução. Imprimimos a saída na linha 48; note que tomamos o valor absoluto dos valores obtidos.

l. 11-21 Aqui declaramos a função que implementa o sistema de equações que queremos resolver (dado em (3.11)). A função `sist_ec_gama12` (sistema de equações para a estimativa dos parâmetros da distribuição gamma pelo método de substituição utilizando os momentos de ordem 1 y 2) será avaliada nos valores do vetor `vX` e seu resultado (vetorial) será armazenado no vetor `avF`.

l. 14-15 Impomos a restrição de utilizar só valores positivos ao calcular o valor absoluto dos argumentos e ao considerar o valor absoluto da solução encontrada (linha 48).

l. 17 Atribuímos um vetor com o resultado de avaliar cada equação do sistema dado em (3.11) a `avF[0]`. Note como formamos um vetor através da concatenação em coluna (operador '`|`') dos valores.

l. 20 A função deve retornar 1 quando for possível fazer a resolução sem nenhum problema; outros valores sinalizam outras situações.

l. 33-34 Instruções opcionais com as quais se indica qual gerador de números pseudo-aleatórios deverá ser empregado (neste caso escolhemos o gerador de L'Ecuyer) e sua semente (que para este gerador é um vetor de dimensão 4).

A saída deste programa é

```
Ox version 3.40 (Linux) (C) J.A. Doornik, 1994-2004
Solução =
       9.8731
     0.010129
```

que, usando a notação já definida, significa $(\widehat{\alpha}, \widehat{\beta})(\omega) = (9.87, 0.01)$.

É importante notar que este resultado só é válido para a amostra que utilizamos. Ao alterar a semente, o tamanho da amostra ou o valor de algum parâmetro teremos outra estimativa para (α, β); a particularidade do caso está manifestada ao usar '(ω)', indicando que é um único evento. Não podemos afirmar nada sobre $(\widehat{\alpha}, \widehat{\beta})$ em geral; para isso deveríamos repetir muitas vezes a experiência numérica para chegar a algum tipo de conclusão. Esse é o o assunto do Capítulo 7.

A função `SolveNLE` admite vários parâmetros e entradas, sendo uma das opções mais importantes o uso do jacobiano do sistema de equações que se deseja resolver. Informar o jacobiano tem o efeito (usualmente) positivo de acelerar a convergência, ao custo (sempre) de requerer mais operações. Quando o jacobiano não está disponível de forma analítica é possível utilizar a função `NumJacobian`, que o calcula de forma numérica.

Capítulo 4

Inferência pelo Método de Máxima Verossimilhança e Otimização

No capítulo anterior vimos uma técnica de estimação de aplicabilidade universal. Nem sempre se conhecem as propriedades exatas dos estimadores calculados pelo método de substituição, o que estimula a busca por outros métodos de estimação. Neste capítulo veremos a técnica de inferência baseada no conceito de *verossimilhança* e algoritmos que a implementam. Veremos que, em geral, estimadores de máxima verossimilhança podem ser calculados através da solução de um problema de otimização; em alguns casos este problema pode ser transformado em um problema de solução de sistemas de equações, tal como visto no Capítulo 3.

Ainda que este conceito possa ser aplicado a qualquer modelo estatístico paramétrico, por razões de espaço limitaremos a discussão à análise de observações que são eventos de variáveis aleatórias i.i.d.

4.1 O Conceito de Verossimilhança

Dizemos que $\widehat{\boldsymbol{\theta}}$ é um estimador de máxima verossimilhança para o parâmetro $\boldsymbol{\theta}$ sob a amostra $\boldsymbol{y} = (y_1, \ldots, y_n)$ se

$$\widehat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}\in\Theta} L(\boldsymbol{\theta}; \boldsymbol{y}), \tag{4.1}$$

onde L é a verossimilhança dos dados $\boldsymbol{y}$. Para dados provenientes de variáveis aleatórias i.i.d., temos que

$$L(\boldsymbol{\theta}; \boldsymbol{y}) = \prod_{1\leq i\leq n} f(\boldsymbol{\theta}; y_i),$$

onde $f(\boldsymbol{\theta}; y_i) = f_Y(y_i; \boldsymbol{\theta})$ e $f_Y(y_i; \boldsymbol{\theta})$ é a densidade da variável aleatória indexada pelo parâmetro $\boldsymbol{\theta}$. Em outras palavras, a verossimilhança é a função de densidade de probabilidade, só que com o argumento y fixo (visto que foi observado), e variando o parâmetro. Desta forma, a verossimilhança **não** é um produto de densidades.

Um estimador de máxima verossimilhança maximiza a verossimilhança conjunta (equação (4.1)), isto é, é um valor do parâmetro que faz com que a amostra observada seja a mais verossímil. Na maioria das aplicações não interessa o valor que a função de verossimilhança adota; só estamos interessados em argumentos que a maximizam.

Para o problema de estimativa dos parâmetros de uma distribuição gama que vínhamos tratando, dada a amostra $\boldsymbol{y} = (y_1, \ldots, y_n)$ um estimador de máxima verossimilhança $\widehat{\boldsymbol{\theta}} = \widehat{(\alpha, \beta)}$ para o parâmetro $\boldsymbol{\theta} = (\alpha, \beta)$ é qualquer ponto de $\mathbb{R}^2_+$ que satisfaça

$$\begin{aligned}
\widehat{(\alpha, \beta)} &= \arg\max_{(\alpha,\beta)\in\mathbb{R}^2_+} \prod_{1\leq i\leq n} \frac{1}{\beta^\alpha \Gamma(\alpha)} y_i^{\alpha-1} \exp(-y_i/\beta) \\
&= \arg\max_{(\alpha,\beta)\in\mathbb{R}^2_+} (\beta^\alpha \Gamma(\alpha)^{-n} \prod_{1\leq i\leq n} y_i^{\alpha-1} \exp(-y_i/\beta). \qquad (4.2)
\end{aligned}$$

Já que todos os termos da equação (4.2) são positivos, podemos trabalhar com o logaritmo; fazendo isto temos que a equação (4.2)

reduz-se a

$$\widehat{(\alpha,\beta)} = \arg\max_{(\alpha,\beta)\in\mathbb{R}_+^2}\left\{-n\alpha\ln\beta - n\ln\Gamma(\alpha) + (\alpha-1)\sum_{1\le i\le n}\ln y_i - \frac{1}{\beta}\sum_{1\le i\le n} y_i\right\}. \tag{4.3}$$

A equação (4.3) é conhecida como *equação de log-verossimilhança*, e costuma ser mais simples de resolver que a equação de verossimilhança. Podemos simplificar ainda um pouco mais o problema ao notar que existem termos na equação (4.3) que não dependem de α nem de β e, por isso, podemos descartá-los. Assim sendo, nosso problema final é encontrar

$$\widehat{(\alpha,\beta)} = \arg\max_{\widehat{\boldsymbol{\theta}}\in\Theta}\ell(\boldsymbol{\theta};\boldsymbol{y}) = \arg\max_{(\alpha,\beta)\in\mathbb{R}_+^2}\left\{-n\alpha\ln\beta - n\ln\Gamma(\alpha) + \alpha\sum_{1\le i\le n}\ln y_i - \frac{1}{\beta}\sum_{1\le i\le n} y_i\right\}. \tag{4.4}$$

Esta última equação costuma ser chamada de *equação de log-verossimilhança reduzida*. Pode constatar-se facilmente que não se pode resolvê-la de forma explícita e, por isso, para encontrar um estimador de máxima verossimilhança tem-se que utilizar rotinas de otimização. Este será o tema da seção 4.2.

Alternativamente, é possível tratar o problema resolvendo o sistema de equações formado pelo gradiente da equação de log-verossimilhança reduzida, isto é, tomar $\widehat{\boldsymbol{\theta}}$ como sendo algum valor que satisfaça

$$\nabla\ell(\widehat{\boldsymbol{\theta}};\boldsymbol{y}) = \mathbf{0}$$

que, em nosso caso, reduz-se a

$$\begin{cases} \dfrac{1}{n}\displaystyle\sum_{1\le i\le n}\ln y_i - \ln\widehat{\beta} - \Psi(\widehat{\alpha}) = 0 \\ \dfrac{1}{n\widehat{\beta}}\displaystyle\sum_{1\le i\le n} y_i - \widehat{\alpha} = 0, \end{cases}$$

onde $\Psi(\nu) = \Gamma'(\nu)/\Gamma(\nu)$ é a função digama. Uma vez formulado desta maneira, o problema do cálculo de estimadores pelo método

de máxima verossimilhança pode ser resolvido de maneira análoga à apresentada no Capítulo 3.

4.2 Algoritmos para Otimização

A maximização direta da função de verossimilhança ou da função de log-verossimilhança reduzida pode ser facilmente realizada no `Ox`, já que a plataforma oferece rotinas para isso.

Comecemos por escrever log-verossimilhança, a partir de (4.3), de forma mais tratável:

$$\ell(\alpha, \beta; \boldsymbol{y}) = -\alpha \ln \beta - \ln \Gamma(\alpha) + \alpha \frac{1}{n} \sum_{1 \leq i \leq n} \ln y_i - \frac{1}{\beta} \frac{1}{n} \sum_{1 \leq i \leq n} y_i. \quad (4.5)$$

O termo $n^{-1} \sum_{i=1}^{n} y_i$ foi calculado no programa principal e armazenado na variável global `g_m1`. Calcularemos o termo $n^{-1} \sum_{i=1}^{n} \ln y_i$ com o comando

```
g_logm1 = meanc(log(v_amostra));
```

e armazenaremos o resultado na variável global `g_logm1`.

O código que implementa a função dada na equação (4.5) é o seguinte:

```
1  func_verossgamma(const vP, const adFunc, const avScore,
2                   const amHessian) {
3
4    decl alfa, beta;
5
6    alfa = fabs(vP[0]);
7    beta = fabs(vP[1]);
8
9    adFunc[0] = -alfa * log(beta) - loggamma(alfa) +
10                alfa * g_logm1 - g_m1 / beta;
11
12   return 1;
13 }
```

e a chamada à função que a maximiza é

```
1  ir = MaxBFGS(func_verossgamma, &v_solucao, &dFunc, 0, 1);
2  println("Estimador␣maxver␣por␣çãotimizao␣=␣",
3          fabs(v_solucao));
4  println("Convergencia:␣", MaxConvergenceMsg(ir));
```

Na linha 1 atribuímos à variável `ir` o resultado da chamada à função de otimização `MaxBFGS`. Esse resultado nos informa o tipo de convergência obtido pelo algoritmo, e para sabê-lo utilizamos a chamada à função `MaxConvergenceMsg`, tal como mostrado na linha 4.

Nem sempre diferentes algoritmos convergem para a mesma solução. Em alguns casos que podem ser considerados patológicos, diferentes algoritmos (ou o mesmo algoritmo com diferentes ajustes ou valores iniciais) podem levar a soluções muito diferentes, tal como se discute em [21].

Para concluir este capítulo comentaremos o resultado de calcular os estimadores obtidos pelo método de momentos e pelo método de máxima verossimilhança implementado por otimização para uma amostra de tamanho 10000 e $(\alpha, \beta) = (10, 1/10)$. Utilizando uma certa semente para o gerador escolhido, os resultados foram, respectivamente, $(\widehat{\alpha}_{MO} = 10.131, \widehat{\beta}_{MO} = 0.099)$ e $(\widehat{\alpha}_{MV} = 10.059, \widehat{\beta}_{MV} = 0.100)$. O que podemos afirmar sobre estes estimadores? Nada! Estes resultados são amostras de tamanho unitário de variáveis aleatórias, e qualquer comparação medianamente justa deverá basear-se não em uma amostra deste tipo e sim em alguma propriedade mais geral.

Já que não conhecemos tais propriedades gerais, e já que elas são difíceis de derivar em geral, veremos no próximo capítulo como ter uma idéia aproximada sobre elas utilizando técnicas computacionais de simulação.

O `R`, através do pacote `MASS`, provê uma função que permite realizar a estimação por máxima verosimilhança dos parâmetros de vários modelos de uso freqüente na estatística: a função `fitdistr`. As distribuições implementadas na versão utilizada na época da preparação destas notas são beta, Cauchy, χ^2, exponencial, $\mathcal{F}$, gama, log-normal, logística, binomial negativa, gaussiana, t, uniforme e Weibull. A seguir mostramos o código que implementa a simulação de uma amostra de tamanho 100 da distribuição beta com parâmetros 4 e 2 (linha 3), que estima estes parâmetros (linhas 4 e 5) usando como valores iniciais 2 e 4, respectivamente. A partir da linha 13 o código implementa a visualização simultânea das densidades verdadeira e estimada, mostradas na Figura 4.1.

```
1 > # estimacao por maxima verossimilhanca
2 > library(MASS)
```

```
> random <- rbeta(100, shape1=4, shape2=2)
 > a = fitdistr(random, dbeta, start=list(shape1
+ =2, shape2=4))
> a
     shape1       shape2
  4.5502955   2.2374403
 (0.6405170) (0.2977327)
> a = unlist(a)
> z = seq(from = 0.01, to = 0.99, length
+ = 200)
> plot(z, dbeta(z, shape1=4, shape2=2),
+ xlab="", ylab="", type="l", ylim=c(0,2.5),
+ xlim=c(0.01,0.95))
> lines(z, dbeta(z, shape1=a[1], shape2=a[2]),
+ type="l", lty=2)
```

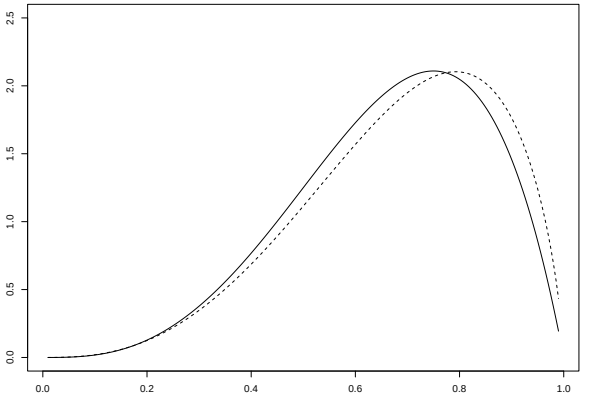

Figura 4.1: Densidades teórica e estimada (linhas contínua e tracejada, respectivamente)

Capítulo 5

Otimização Não-linear

5.1 Introdução

Em muitas situações práticas é comum precisarmos minimizar ou maximizar funções. Um exemplo de grande importância é a obtenção de estimativas de máxima verossimilhança em modelos estatísticos e econométricos; em muitos casos de interesse o estimador de máxima verossimilhança não possui forma fechada e as estimativas devem ser obtidas a partir da maximização numérica da função de verossimilhança ou da função de log-verossimilhança, ou seja, precisamos construir esta função com base no modelo postulado e depois maximizá-la numericamente a fim de encontrar as estimativas de máxima verossimilhança dos parâmetros que definem o modelo. Um outro exemplo envolve o problema de mínimos quadrados, onde o interesse reside na minimização da soma dos quadrados de um conjunto de erros, por exemplo, na estimação de modelos não-lineares de regressão pelo método de mínimos quadrados não-lineares.

O presente capítulo apresenta um conjunto de ferramentas que são úteis na tarefa de encontrar mínimos e máximos de funções. Não nos preocuparemos inicialmente com a existência de mais de um mínimo ou máximo; a técnica de *simulated annealing*, apresentada mais adiante, será útil na localização de mínimos e máximos globais.

Ao longo do capítulo trataremos da maximização de funções; para minimizar uma função utilizando os métodos descritos a seguir basta

multiplicá-la por -1 e proceder à sua maximização. Para maiores detalhes sobre os métodos apresentados neste capítulo, ver [38].

5.2 O Problema de Interesse

Suponha que o nosso interesse reside na maximização de uma determinada função, digamos $\Lambda\colon \Theta \to \mathbb{R}$, onde Θ é um subespaço de $\mathbb{R}^p$. Suponha inicialmente que a função de interesse é quadrática, ou seja, suponha que Λ pode ser escrita como

$$\Lambda(\boldsymbol{\theta}) = \alpha + \boldsymbol{\beta}'\boldsymbol{\theta} + \frac{1}{2}\boldsymbol{\theta}'\Gamma\boldsymbol{\theta},$$

onde α é um dado escalar, $\boldsymbol{\beta}$ é um vetor de dimensão $p \times 1$ e Γ é uma matriz positiva-definida de ordem $p \times p$. A condição de primeira ordem para a maximização de Λ é dada por $\boldsymbol{\beta} - \Gamma\boldsymbol{\theta} = 0$, resultando assim na solução

$$\boldsymbol{\theta} = \Gamma^{-1}\boldsymbol{\beta},$$

com a condição de que Γ é positiva-definida garantindo que Γ^{-1} existe. Este é um problema de otimização *linear* que resulta numa solução que possui *forma fechada*. Os problemas encontrados com maior freqüência, contudo, são aqueles onde a condição de primeira ordem,

$$\frac{\partial\Lambda(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} = \mathbf{0},$$

constitui um sistema de equações não-lineares que não apresenta solução em forma fechada. Os métodos apresentados abaixo buscam encontrar o máximo da função Λ através do uso de *algoritmos iterativos*.

5.3 Métodos Gradiente

O nosso objetivo, como mencionado acima, é o de localizar o ponto de máximo da função Λ; para tanto, utilizaremos um esquema iterativo. Iniciando em $\boldsymbol{\theta}_0$, na iteração t se o valor ótimo de $\boldsymbol{\theta}$ não houver sido

alcançado, calcule o vetor direcional $\boldsymbol{\Delta}_t$ ($p \times 1$) e o 'tamanho do passo' λ_t; o próximo valor de $\boldsymbol{\theta}$ no esquema iterativo é dado por

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \lambda_t \boldsymbol{\Delta}_t.$$

Convém notar que para dados $\boldsymbol{\theta}_t$ e $\boldsymbol{\Delta}_t$, um processo secundário de otimização deve ser empregado para que seja localizado o tamanho de passo (λ_t) mais apropriado; este processo auxiliar de otimização é usualmente conhecido como *procura em linha*. Seja f o vetor de derivadas parciais de Λ. O problema de procura em linha pode ser descrito da seguinte forma: busca-se λ_t tal que

$$\frac{\partial \Lambda(\boldsymbol{\theta}_t + \lambda_t \boldsymbol{\Delta}_t)}{\partial \lambda_t} = f(\boldsymbol{\theta}_t + \lambda_t \boldsymbol{\Delta}_t)' \boldsymbol{\Delta}_t = 0.$$

É importante ressaltar, todavia, que a introdução de buscas em linha em algoritmos de otimização não-linear tipicamente torna estes algoritmos muito intensivos e custosos do ponto de vista computacional. Muitas implementações substituem o mecanismo de procura em linha por um conjunto de regras *ad hoc* menos custosas computacionalmente.

A classe mais utilizada de algoritmos iterativos é conhecida como *classe de métodos gradiente*. Aqui,

$$\boldsymbol{\Delta}_t = M_t f_t,$$

onde M_t é uma matriz positiva-definida e f_t é o gradiente de Λ, ambos na iteração t. Segundo esta notação, $f_t = f_t(\boldsymbol{\theta}_t) = \partial \Lambda(\boldsymbol{\theta}_t)/\partial \boldsymbol{\theta}_t$. Para entender sua motivação, considere uma expansão de Taylor de $\Lambda(\boldsymbol{\theta}_t + \lambda_t \boldsymbol{\Delta}_t)$ em torno de $\lambda_t = 0$:

$$\Lambda(\boldsymbol{\theta}_t + \lambda_t \boldsymbol{\Delta}_t) \approx \Lambda(\boldsymbol{\theta}_t) + \lambda_t f(\boldsymbol{\theta}_t)' \boldsymbol{\Delta}_t.$$

Seja $\Lambda(\boldsymbol{\theta}_t + \lambda_t \boldsymbol{\Delta}_t) = \Lambda_{t+1}$. Assim, temos que

$$\Lambda_{t+1} - \Lambda_t \approx \lambda_t f_t' \boldsymbol{\Delta}_t.$$

Se $\boldsymbol{\Delta}_t = M_t f_t$, como na classe de métodos gradiente, então

$$\boldsymbol{\Lambda}_{t+1} - \boldsymbol{\Lambda}_t \approx \lambda_t f_t' M_t f_t.$$

Suponha que f_t é diferente de zero e que λ_t é suficientemente pequeno. Temos assim que se $\Lambda(\boldsymbol{\theta})$ não se encontra no máximo, podemos sempre encontrar um tamanho de passo tal que uma iteração adicional conduzirá a um incremento no valor da função. Isto é verdade porque M_t é positiva-definida e, como não estamos no ponto de máximo, o gradiente da função a ser maximizada é diferente de zero, o que implica $f_t' M_t f_t > 0$.

5.3.1 *Steepest Ascent*

O algoritmo mais simples é o da *subida mais inclinada*, também conhecido como algoritmo de *steepest ascent*. A idéia por trás deste algoritmo é usar

$$M_t = I,$$

ou seja toma-se a matriz M_t, considerada positiva-definida acima, como sendo a matriz identidade de ordem p em todos os passos do esquema iterativo, o que resulta em $\boldsymbol{\Delta}_t = f_t$. Este algoritmo tende a ser pouco utilizado em aplicações práticas, pois tipicamente apresenta convergência lenta.

5.3.2 Newton-Raphson

O método de Newton ou de Newton-Raphson pode ser descrito pela seguinte equação de atualização:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - H_t^{-1} f_t,$$

onde

$$H = H(\boldsymbol{\theta}) = \frac{\partial^2 \Lambda(\boldsymbol{\theta})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}'},$$

i.e., H é a matriz hessiana. Neste método, temos, portanto, $M_t = -H_t^{-1}$ e $\lambda_t = 1$ para todo t.

Para entender a motivação por trás deste método, considere uma expansão em série de Taylor da condição de primeira ordem em torno de um ponto arbitrário, digamos $\boldsymbol{\theta}_0$:

$$\frac{\partial \Lambda(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \approx f(\boldsymbol{\theta}_0) + H(\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0).$$

Resolvendo para $\boldsymbol{\theta}$ e colocando $\boldsymbol{\theta} = \boldsymbol{\theta}_{t+1}$ e $\boldsymbol{\theta}_0 = \boldsymbol{\theta}_t$, obtemos o esquema iterativo de Newton-Raphson dado acima.

A forma mais usual do algoritmo inclui um mecanismo de procura em linha e o esquema iterativo é dado por

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t H_t^{-1} f_t,$$

onde λ_t é como descrito anteriormente.

O método de Newton-Raphson funciona bem em muitas situações, mas pode apresentar desempenho ruim em alguns casos. Em particular, se a função não for aproximadamente quadrática ou se a estimativa corrente se encontrar muito distante do ponto ótimo, pode haver problemas de convergência. Em particular, em pontos distantes do ponto maximizador de Λ, a matriz de segundas derivadas pode não ser negativa-definida, o que violaria a suposição de que M_t é positiva-definida no esquema iterativo geral.

5.3.3 BHHH

O método BHHH foi proposto por [1] e é semelhante ao método de Newton-Raphson. A única diferença reside no fato de que se usa a matriz $f_t f_t'$ (conhecida como *outer product of the gradient*) ao invés de H_t no esquema iterativo. Ou seja, usamos

$$-H^* = -H^*(\boldsymbol{\theta}) = -\frac{\partial \Lambda(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}_t} \frac{\partial \Lambda(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}_t'}$$

ao invés de

$$H = H(\boldsymbol{\theta}) = \frac{\partial^2 \Lambda(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}_t \partial \boldsymbol{\theta}_t'}$$

no esquema iterativo de Newton. Note que aqui não precisamos calcular a matriz de segundas derivadas. Este método é muito usado em várias aplicações econonométricas; por exemplo, em [5] sugere-se o uso deste algoritmo para estimação de modelos GARCH (modelos de heteroscedasticidade condicional auto-regressiva generalizada).

5.3.4 Escore de Fisher

O método escore de Fisher (*Fisher's scoring*) também é semelhante ao método de Newton-Raphson. A diferença é que no esquema iterativo

usamos o valor esperado da matriz de segundas derivadas ao invés da matriz de segundas derivadas (H) em si. Ou seja, usamos

$$K = K(\boldsymbol{\theta}) = \mathrm{E}\left(\frac{\partial^2 \Lambda(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}_t \partial \boldsymbol{\theta}_t'}\right)$$

ao invés de

$$H = H(\boldsymbol{\theta}) = \frac{\partial^2 \Lambda(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}_t \partial \boldsymbol{\theta}_t'}.$$

Note que se Λ, a função a ser maximizada, for uma função de log-verossimilhança, então K é a matriz de informação de Fisher e, portanto, $M = [\mathrm{E}\{H(\boldsymbol{\theta})\}]^{-1}$ é a variância assintótica de $\sqrt{n}$ vezes o estimador de máxima verossimilhança de $\boldsymbol{\theta}$. Este método é muito utilizado, por exemplo, para estimação de modelos lineares generalizados; ver, e.g., [35].

5.3.5 Quasi-Newton

Há uma classe de algoritmos muito eficientes que elimina a necessidade do cálculo de segundas derivadas e tipicamente apresenta bom desempenho: a classe de *algoritmos quasi-Newton*, também conhecida como classe de *métodos de métricas variáveis*. Nesta classe, usa-se a seguinte seqüência de matrizes:

$$M_{t+1} = M_t + N_t,$$

onde N_t é uma matriz positiva-definida. Note que se M_0, a matriz inicial da seqüência, for positiva-definida, então todas as demais matrizes da sequüência também o serão. A idéia básica é construir iterativamente uma boa aproximação para $-\{H(\boldsymbol{\theta})\}^{-1}$, ou seja, usar uma seqüência de matrizes M_t tal que $\lim_{t\to\infty} M_t = -H^{-1}$. A idéia central do método remonta a um artigo que Davidon escreveu no final da década de 1950 [17]; este artigo, contudo, não foi aceito para publicação à época, e sua publicação só veio a se dar em 1991, mais de trinta anos mais tarde [18]. Hoje há diferentes algoritmos que pertencem a esta classe. Por exemplo, o algoritmo DFP (Davidon, Fletcher e Powell) usa

$$M_{t+1} = M_t + \frac{\boldsymbol{\delta}_t \boldsymbol{\delta}_t'}{\boldsymbol{\delta}_t' \nu_t} + \frac{M_t \nu_t \nu_t' M_t}{\nu_t' M_t \boldsymbol{\delta}_t},$$

onde $\boldsymbol{\delta}_t = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t$ e $\nu_t = f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t)$.

O algoritmo quasi-Newton mais utilizado é o BFGS (Broyden, Fletcher, Goldfarb e Shanno). Aqui, subtraímos o seguinte termo do esquema de atualização DFP: $a_t b_t b_t'$, onde $a_t = \nu_t' M_t \nu_t$ e

$$b_t = \frac{\boldsymbol{\delta}_t}{\boldsymbol{\delta}_t' \nu_t} - \frac{M_t \nu_t}{\nu_t' M_t \nu_t}.$$

Ou seja, no algoritmo BFGS temos

$$\begin{aligned} M_{t+1} &= M_t + \frac{\boldsymbol{\delta}_t \boldsymbol{\delta}_t'}{\boldsymbol{\delta}_t' \nu_t} + \frac{M_t \nu_t \nu_t' M_t}{\nu_t' M_t \boldsymbol{\delta}_t} - \nu_t' M_t \nu_t \left(\frac{\boldsymbol{\delta}_t}{\boldsymbol{\delta}_t' \nu_t} - \frac{M_t \nu_t}{\nu_t' M_t \nu_t} \right) \\ &\quad \left(\frac{\boldsymbol{\delta}_t}{\boldsymbol{\delta}_t' \nu_t} - \frac{M_t \nu_t}{\nu_t' M_t \nu_t} \right)'. \end{aligned}$$

Note que nos algoritmos DFP e BFGS a matriz M_t é sempre positiva-definida, desde que se inicie a seqüência de atualização em uma matriz que possua esta propriedade. Assim, supera-se uma limitação do método Newton-Raphson, pois neste método a matriz $M_t = -H_t^{-1}$ pode não ser positiva-definida se estivermos longe do ponto ótimo. Elimina-se também a necessidade do cálculo de segundas derivadas, da inversão da matriz hessiana e da avaliação desta matriz em cada iteração do processo otimizador.

O algoritmo BFGS tem geralmente desempenho melhor que a versão DFP, sendo assim mais utilizado em aplicações práticas. Para uma implementação em `C` do algoritmo BFGS, ver [39].

A terminologia *quasi*-Newton se deve ao fato de que nós não usamos a matriz hessiana, mas usamos uma aproximação para ela construída de forma iterativa. Não se deve subentender que este método é inferior ao método de Newton-Raphson por não utilizar a matriz hessiana; de fato, em muitas situações práticas ele tem desempenho superior ao método de Newton-Raphson.

5.4 Problemas Combinatórios e *Simulated Annealing*

A técnica de *simulated annealing* é um pontos altos da pesquisa em otimização e simulação estocástica dos anos 80 e 90. Esta técnica é

inspirada em um processo físico para gerar cristais de alta pureza, isto é, muito regulares, conhecida como *recozido*; daí a (imperdoável) tradução "recozido simulado".

O princípio do recozido é fundir o cristal, a alta temperatura, para depois ir esfriando-o muito devagar. Ao fazer este esfriamento, pelo menos em princípio, as moléculas irão se acomodar da forma mais regular possível, já que quanto mais regular a estrutura menor será a energia total do sistema.

A analogia consiste em considerar um problema através das suas soluções possíveis, cada uma delas associada a um custo e a um conjunto de outras soluções viáveis. Começando em uma solução arbitrária x_0, o algoritmo escolhe uma solução candidata na vizinhança e computa o seu custo. Caso o custo da candidata seja inferior ao custo de x_0, ela é escolhida como nova solução e se prossegue. Caso o custo da solução candidata seja maior do que o custo de x_0, ela ainda tem alguma chance de ser aceita. Essa chance depende de um parâmetro chamado temperatura, que controla a evolução do algoritmo.

Tal como originalmente formulado, este algoritmo é absolutamente geral. O problema particular sendo tratado, isto é, o domínio de aplicação, irá ditar formas mais ou menos eficientes de implementá-lo. Esta implementação consiste, essencialmente, da especificação de

1. a vizinhança de cada solução viável
2. a forma em que será escolhida uma solução na vizinhança
3. a probabilidade com que soluções piores (de maior custo) do que a atual serão aceitas
4. a regra ou regras de iteração e de parada do algoritmo.

Existem várias provas da convergência do algoritmo de *simulated annealing* para o conjunto dos mínimos globais da função de custo, contudo todas elas requerem um número infinito de iterações. Há algumas pesquisas recentes que fornecem resultados para um número finito de iterações, que é a situação real que sempre deveríamos considerar. Somente enunciaremos o resultado assintótico.

Embora esta técnica possa ser aplicada em princípio a absolutamente qualquer problema discreto, ela é mais atraente para problemas

combinatórios. Estes problemas são os mais complexos de serem resolvidos do ponto de vista computacional. A técnica também pode ser empregada em problemas de otimização contínua [3]. Uma referência muito boa para o assunto é o artigo da revista *Science* de [29].

Um problema de otimização combinatória pode ser formalizado como um par (R, C), onde R é o conjunto (finito ou enumerável) das configurações ou soluções viáveis, e C é uma função de custo que a cada elemento em R associa um valor real, isto é, $C\colon R \to \mathbb{R}$. O custo é tipicamente, mas não necessariamente, não negativo. A função C é definida de tal forma que quanto menor, melhor a solução. Com estes ingredientes, o problema consiste em achar a(s) configuração(ções) nas quais o custo alcança o menor valor, isto é, achar

$$\Xi^* = \arg\min_{\xi \in R} C(\xi).$$

O algoritmo de *simulated annealing* requer a definição de uma vizinhança de configurações, isto é, para cada elemento $\xi \in R$ deve existir $\partial_\xi = \{t \in R \setminus \{\xi\}\colon t \in \partial_\xi \Leftrightarrow \xi \in \partial_t\}$. Este conjunto de vizinhanças deve ser tal que para todo par de configurações (ξ_0, ξ_L) de vizinhanças ∂_0 e ∂_L existe pelo menos uma seqüência de configurações $(\xi_1, \ldots, \xi_{L-1})$ cujas vizinhanças $(\partial_1, \ldots, \partial_{L-1})$ têm a propriedade $\partial_i \cap \partial_{i+i} \neq \emptyset$ para todo $0 \leq i \leq L-1$. Em outras palavras, começando em qualquer configuração ξ_0 é possível chegar em qualquer outra configuração ξ_L transitando pelas vizinhanças. Esta condição é necessária para garantir que uma certa cadeia de Markov definida em R seja irredutível. Na literatura física, é comum encontrar esta condição descrita como "o ponto ótimo é alcançável (*reachable*) a partir de qualquer configuração inicial".

Evidentemente, é possível definir um conjunto de vizinhanças que satisfaz esta condição fazendo $\partial_\xi = \{R \setminus \{\xi\}\}$ para todo $\xi \in R$, mas esta escolha é pouco conveniente.

Outro ingrediente fundamental do algoritmo é a sua *dinâmica*, isto é, o conjunto de regras segundo o qual o procedimento se rege. Começando de uma configuração inicial $\xi(0) \in R$ uma configuração será escolhida na vizinhança de $\xi(0)$, digamos ζ. Esta configuração será chamada candidata. A candidata será aceita como a nova configuração, isto é $\xi(1) = \zeta$, se $C(\zeta) < C(\xi(0))$; em outras palavras, toda vez que a candidata diminuir o custo ela será aceita. Caso esta seja a

única especificação do algoritmo, ele irá se deter no primeiro mínimo local que alcançar, o que não é desejável.

Para fugir dos mínimos locais deve ser fornecida uma regra de escape. Uma regra que garante a convergência ao conjunto de mínimos globais Ξ^*, conhecida como *relaxação estocástica de Metropolis*, é dada por

$$\Pr(\xi(i) = \zeta \mid C(\zeta) \geq C(\xi(i-i))) = \exp\Big\{\frac{1}{T_i}\big(C(\xi(i-i)) - C(\zeta)\big)\Big\},$$

onde o parâmetro $T_i > 0$ é chamado *temperatura no instante* i. Em palavras, uma configuração ζ pior do que a atual $\xi(i-1)$ será aceita com uma certa probabilidade que depende inversamente da diferença de custos. Quanto maior a temperatura maior a chance de serem aceitas configurações "ruins". Para garantir a convergência do algoritmo ao conjunto Ξ^* é necessário que a seqüência de temperaturas obedeça a uma certa regra, sempre com $T_i \geq T_{i+1}$, isto é, as temperaturas deverão ser não-crescentes.

O principal resultado pode ser enunciado como "sob certas condições impostas sobre a seqüência de temperaturas $(T_i)_{i\geq 1}$ vale que $\Pr(\xi(i) \in \Xi^*) \to 1$ quando $i \to \infty$ para qualquer $\xi(0) \in R$ obedecendo a dinâmica acima especificada."

Existem várias provas deste resultado, sendo possível classificá-las segundo vários critérios. O critério mais famoso é o que se baseia na homogeneidade ou heterogeneidade da cadeia de Markov que define a dinâmica. O algoritmo acima está definido sobre uma cadeia homogênea se a temperatura fica fixa durante um certo tempo, para depois diminuir e ficar fixa novamente durante outro certo tempo e assim por diante; caso contrário a cadeia é dita ser heterogênea. Para que haja convergência ao conjunto Ξ^* com cadeias homogêneas é necessário que em cada temperatura o algoritmo siga cadeias de comprimento infinito, e que a temperatura diminua. Quando a cadeia é não homogênea, é imprescindível que a diminuição da temperatura seja da forma $T_i = k/\ln(i)$. A constante k depende do problema sendo tratado.

A prova é geral, mas como implementar cadeias de Markov de comprimento infinito? Esta tarefa não é possível em geral, e devem ser empregadas versões finitas onde a convergência ao conjunto Ξ^* não está garantida. Mesmo assim, o poder deste algoritmo é inegável.

5.5 Implementação Computacional

Na plataforma `Ox`, o método BFGS se encontra implementado através da função `MaxBFGS`. A implementação permite a escolha entre primeiras derivadas analíticas (fornecidas pelo usuário) e primeiras derivadas numéricas (calculadas pela plataforma). A função nativa `MaxNewton` implementa os métodos BHHH, escore de Fisher, Newton-Raphson e da subida mais inclinada, permitindo ao usuário escolher entre segundas derivadas analíticas ou numéricas. Código para estimação via *simulated annealing* em `Ox` foi desenvolvido por Charles Bos (`MaxSA`); ver o código fonte em `http://www.tinbergen.nl/~cbos/software/maxsa.html`.

Na plataforma `R`, pode-se usar a função `optim` para realizar otimização não-linear. Entre outros métodos, estão disponíveis para utilização em `optim`: BFGS, Newton-Raphson e *simulated annealing*. Há também a opção de se usar BFGS com restricções de caixa, onde é possível especificar limites inferior e/ou superior para os elementos do vetor $\boldsymbol{\theta}$ (ver [12]). Convém notar que `optim` realiza *minimização* de funções, contrariamente às funções disponíveis na plataforma `Ox`; para *maximizar* funções, use a opção `fnscale=-1`.

5.6 Exemplos

Seja $Y_1, \ldots, Y_n$ uma amostra aleatória de uma distribuição t_m, onde m denota o número de graus de liberdade da distribuição t de Student. Suponha que desejamos, para uma amostra gerada aleatoriamente com $n = 50$, estimar m por máxima verossimilhança. Isto é feito no programa abaixo, escrito usando a linguagem `Ox`.

```
1  /* PROGRAMA: t.ox */
2
3  #include <oxstd.h>
4  #include <oxprob.h>
5  #import <maximize>
6
7  const decl N = 50;
8  static decl s_vx;
9
```

```
fLogLik(const vP, const adFunc, const avScore,
         const amHess)
{
        decl vone = ones(1,N);
        adFunc[0] = double( N*loggamma((vP[0]
                  +1)/2) - (N/2)*log(vP[0])
                  - N*loggamma(vP[0]/2)
                  - ((vP[0]+1)/2)*(vone*log(1
                  + (s_vx .^ 2)/vP[0])) );

        if( isnan(adFunc[0]) ||
            isdotinf(adFunc[0]) )
                return 0;

        else
            return 1;    // 1 indica successo
}

main()
{
        decl vp, dfunc, dm, ir;

        ranseed("GM");

        vp = 2.0;
        dm = 3.0;
        s_vx = rant(N,1,dm);

        ir = MaxBFGS(fLogLik, &vp, &dfunc,
                     0, TRUE);

        print("\nCONVERGENCIA:␣␣␣␣␣␣␣␣␣␣␣␣␣␣",
             MaxConvergenceMsg(ir) );

        print("\nLog-vessom.␣maximizada:␣␣␣",
              "%7.3f", dfunc);
        print("\nValor␣verdadeiro␣de␣m:␣␣␣␣",
              "%6.3f", dm);
```

```
48          print("\nEMV␣de␣m:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣",
49                "%6.3f", double(vp));
50          print("\nTamanho␣amostral:␣␣␣␣␣␣␣␣␣",
51                "%6d", N);
52          print("\n");
53
54  }
```

Este programa fornece a seguinte saída, quando executado usado a versão 3.30 da linguagem Ox:

```
1  Ox version 3.30 (Linux) (C) J.A. Doornik, 1994-
2  2003
3
4  CONVERGENCIA:                Strong convergence
5  Log-vessom. maximizada:     -72.813
6  Valor verdadeiro de m:       3.000
7  EMV de m:                    1.566
8  Tamanho amostral:               50
```

Notamos que o valor verdadeiro do parâmetro m é 3 e que a estimativa de máxima verossimilhança obtida é 1.566. É importante notar ainda que: (i) a estimação foi realizada utilizando o método BFGS (quasi-Newton); (ii) foi utilizada primeira derivada numérica; (iii) é necessário fornecer um valor inicial para o parâmetro em estimação; o chute inicial usado foi 2; (iv) houve 'convergência forte', o que significa que dois testes diferentes de convergência indicaram que se atingiu o valor ótimo.

Os possíveis retornos das funções `MaxBFGS` e `MaxNewton` são os seguintes: `MAX_CONV` (convergência forte), `MAX_WEAK_CONV` (convergência fraca), `MAX_MAXIT` (não houve convergência, máximo número de iterações alcançado), `MAX_LINE_FAIL` (não houve convergência, falha no mecanismo de busca em linha), `MAX_FUNC_FAIL` (falha na avaliação da função), `MAX_NOCONV` (não houve convergência).

Neste exemplo, notamos que a estimativa de máxima verossimilhança (1.566) encontra-se distante do valor verdadeiro do parâmetro (3). Isto não se deve ao funcionamento do método de otimização não-linear, mas sim ao fato do estimador de máxima verossimilhança de m ser muito viesado em amostras finitas. Para $n = 500$ e 10000 obtemos,

respectivamente, as seguintes estimativas para m: 2.143 e 2.907. Ou seja, é necessário considerar tamanhos amostrais muito grandes para que se obtenham estimativas razoavelmente precisas. Um estimador de m corrigido por viés foi obtido é apresentado em [47].

Suponha agora que se deseja encontrar o mínimo global da função

$$f(x,y) = x^2 + 2y^2 - \frac{3}{10}\cos(3\pi x) - \frac{2}{5}\cos(4\pi y) + \frac{7}{10}.$$

Note que esta função possui vários mínimos locais. Nós usaremos *simulated annealing* em `R` (versão 1.7.0) para minimizar $f(x,y)$. O primeiro passo é construir o gráfico da função. O código

```
myfunction<-function(x,y)
{
  return(x^2+2*y^2-(3/10)*cos(3*pi*x)-(2/5)*
         cos(4*pi*y)+(7/10))
}
x <- y <-seq(-2,2,length=100)
z <- outer(x,y,myfunction)
persp(x,y,z)
```

produz a Figura 5.1 (página 55).

O próximo passo é escrever a função a ser minimizada em uma forma apropriada para o processo de otimização a ser realizado:

```
fn <- function(x)
{
   x1 <- x[1]
   x2 <- x[2]
   x1^2+2*x2^2-(3/10)*cos(3*pi*x1)-(2/5)*
     cos(4*pi*x2)+7/10
}
```

Sabemos que o mínimo de f ocorre no ponto $(0,0)$, onde $f(0,0) = 0$. De início, tentemos usar um método tradicional de otimização, digamos BFGS, iniciando o esquema iterativo em $(0.5, 0.5)$:

```
optim(c(0.5,0.5), fn, method="BFGS")
```

o que resulta em:

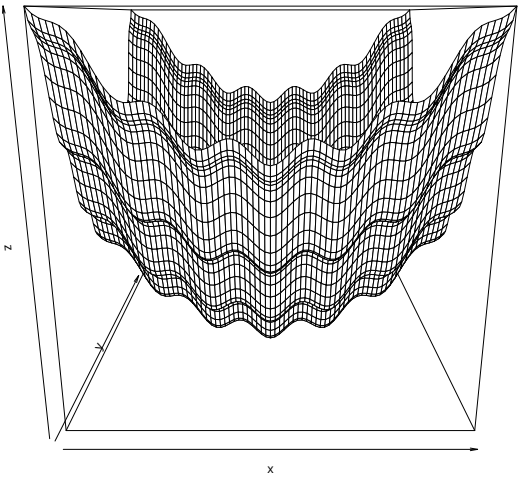

Figura 5.1: Gráfico de $f(x, y)$ vs. x e y.

```
$par
[1] 0.6186103 0.4695268

$value
[1] 0.8828092

$counts
function gradient
      13        6

$convergence
[1] 0

$message
NULL
```

Ou seja, convergimos para um mínimo local onde o valor da função é aproximadamente igual a 0.88. Usemos, agora, *simulated annealing*:

```
optim(c(0.5,0.5), fn, method="SANN")
```

Obtemos, com isto:

```
$par
[1] -0.0002175535 -0.0031842382

$value
[1] 0.0003411431

$counts
function gradient
   10000       NA

$convergence
[1] 0

$message
NULL
```

Assim, o ponto ótimo obtido é $(-0.0002175535, -0.0031842382)$, o valor da função neste ponto sendo 0.0003411431; não ficamos, portanto, presos em um mínimo local. Note que, em ambos os casos, o valor de `convergence` foi `0`, indicando que houve convergência (quando não há convergência, `optim` retorna `1`). Mas somente no segundo caso esta convergência se deu para o mínimo global.

Capítulo 6

Modelos de Séries Temporais

6.1 Modelos de Previsão

Modelos de séries temporais são úteis quando o interesse recai na modelagem e na previsão de dados coletados ao longo do tempo. Enfocaremos a seguir duas estratégias distintas de geração de previsões de valores futuros, a saber: algoritmos de alisamento exponencial e modelos ARIMA (e extensões). Para maiores detalhes sobre as estratégias de previsão descritas a seguir, ver [6, 7, 37].

De início, consideremos os algoritmos de alisamento exponencial, que são de natureza *ad hoc*, mas que tendem a ter bom desempenho em muitas situações práticas. Os três principais algoritmos são: simples, Holt e Holt–Winters. Eles se destinam, respectivamente, à modelagem de séries que possuem apenas nível, que possuem nível e tendência, e que possuem nível, tendência e sazonalidade.

O algoritmo de alisamente exponencial simples é apropriado para séries que não apresentam tendência nem sazonalidade. O nível atual da série $\{N_t\}$ é estimado através de uma média ponderada das observações anteriores, com os pesos descrescendo exponencialmente à medida que regredimos no tempo. A expressão do nível atual é

$$N_t = (1-\alpha)N_{t-1} + \alpha y_t, \quad t \in \mathbb{N}, \tag{6.1}$$

onde $N_{t-1} = \alpha y_{t-1} + \alpha(1-\alpha)y_{t-2} + \cdots$, com $0 < \alpha < 1$.

É necessário selecionar um valor para α. Uma forma razoável de escolher o valor de α é através de inspeção visual, ou seja, se a série evolui de forma suave faz sentido usar um valor alto para α, ao passo que se a série evolui de forma errática faz sentido atribuir peso pequeno à última observação.

Uma estratégia mais objetiva é escolher o valor de α que minimiza a soma dos quadrados dos erros de previsão um passo à frente,

$$S_\alpha = \sum_{t=3}^{n} e_t^2,$$

onde

$$e_t = y_t - N_{t-1} \quad \text{e} \quad N_{t-1} = \widehat{y}_{t-1}(1), \quad t = 3, 4, \ldots, n. \tag{6.2}$$

Aqui $\widehat{y}_{t-1}(1)$ denota a previsão de y_t no instante $t-1$.

Os algoritmos de alisamento exponencial podem ser vistos como um sistema de aprendizado. A partir das equações (6.1) e (6.2) temos que

$$N_t = N_{t-1} + \alpha e_t,$$

ou seja, a estimativa do nível num instante é a soma da estimativa anterior e de um múltiplo do erro de previsão. Se $e_t = 0$, a última previsão foi perfeita, então não há razão para que seja alterada. Todavia, se a última previsão subestimou ou superestimou o valor da série, então é aplicada uma correção quando da previsão da próxima observação.

O alisamento exponencial de Holt é um algoritmo que permite obter estimativas do nível e da tendência da série, sendo, assim, útil para utilização com séries que apresentam comportamentos locais de acréscimo ou decréscimo. Não é necessário que a série possua uma tendência global; o comportamento de tendência pode ser local, sendo requerido apenas que suas mudanças sejam imprevisíveis. A forma de recorrência do algoritmo é dada por

$$N_t = \alpha y_t + (1-\alpha)(N_{t-1} + T_{t-1}), \quad 0 < \alpha < 1,$$

com

$$T_t = \beta(N_t - N_{t-1}) + (1-\beta)T_{t-1}, \quad 0 < \beta < 1,$$

onde N_t e T_t são estimativas do nível e da tendência, respectivamente, no instante t e α e β são constantes de suavização.

A previsão de y_{t+h} feita no instante t é

$$\widehat{y}_t(h) = N_t + hT_t,\ h = 1, 2, \ldots.$$

A escolha objetiva dos valores de α e β pode ser feita através da minimização da soma dos quadrados dos erros de previsão um passo à frente.

Este algoritmo também possui uma forma de correção dos erros, a saber:

$$\begin{aligned} N_t &= N_{t-1} + T_{t-1} + \alpha e_t,\ 0 < \alpha < 1, \\ T_t &= T_{t-1} + \alpha\beta e_t,\ 0 < \beta < 1. \end{aligned}$$

Essa representação do algoritmo revela que ele possui um mecanismo de auto-aprendizado a partir dos erros de previsão cometidos. Quando a previsão anterior é perfeita ($e_t = 0$), as estimativas prévias do nível e da tendência são mantidas. Quando, por outro lado, há um erro de previsão, estas componentes são ajustadas por múltiplos desse erro.

O algoritmo de alisamento exponencial de Holt–Winters tem como objetivo principal permitir a incorporação de padrões sazonais ao algoritmo de Holt. Ele se baseia em três equações que utilizam constantes de alisamento diferentes, cada uma correspondendo a uma das componentes do padrão da série: nível, tendência e sazonalidade. A introdução do comportamento sazonal pode ser feita de duas formas distintas, a saber: aditivamente ou multiplicativamente. A seguir denotaremos o período de sazonalidade da série por s.

Considere de início a forma multiplicativa. A forma de recorrência do algoritmo é dada por

$$\begin{aligned} N_t &= \alpha\frac{y_t}{F_{t-s}} + (1-\alpha)(N_{t-1} + T_{t-1}), \quad 0 < \alpha < 1, \\ T_t &= \beta(N_t - N_{t-1}) + (1-\beta)T_{t-1}, \quad 0 < \beta < 1, \\ F_t &= \gamma\frac{y_t}{N_t} + (1-\gamma)F_{t-s}, \quad 0 < \gamma < 1. \end{aligned}$$

As previsões dos valores futuros da série são obtidas das seguintes expressões:

$$\begin{aligned}
\widehat{y}_t(h) &= (N_t + hT_t)F_{t+h-s},\ h = 1, 2, \ldots, s, \\
\widehat{y}_t(h) &= (N_t + hT_t)F_{t+h-2s},\ h = s+1, s+2, \ldots, 2s, \\
&\vdots \qquad\quad \vdots
\end{aligned}$$

Neste caso, a forma de correção dos erros é

$$\begin{aligned}
N_t &= N_{t-1} + T_{t-1} + \alpha\frac{e_t}{F_{t-s}}, \quad 0 < \alpha < 1, \\
T_t &= T_{t-1} + \alpha\beta\frac{e_t}{F_{t-s}}, \quad 0 < \beta < 1, \\
F_t &= F_{t-s} + \gamma(1-\alpha)\frac{e_t}{N_t}, \quad 0 < \gamma < 1.
\end{aligned}$$

O procedimento anterior pode ser modificado para tratar com situações onde o fator sazonal é aditivo. As equações de atualização, no algoritmo aditivo, são

$$\begin{aligned}
N_t &= \alpha(y_t - F_{t-s}) + (1-\alpha)(N_{t-1} + T_{t-1}), \quad 0 < \alpha < 1, \\
T_t &= \beta(N_t - N_{t-1}) + (1-\beta)T_{t-1}, \quad 0 < \beta < 1, \\
F_t &= \gamma(y_t - N_t) + (1-\gamma)F_{t-s}, \quad 0 < \gamma < 1.
\end{aligned}$$

Os valores futuros são previstos a partir das equações a seguir:

$$\begin{aligned}
\widehat{y}_t(h) &= N_t + hT_t + F_{t+h-s}, \quad h = 1, 2, \ldots, s, \\
\widehat{y}_t(h) &= N_t + hT_t + F_{t+h-2s}, \quad h = s+1, s+2, \ldots, 2s, \\
&\vdots \qquad\quad \vdots
\end{aligned}$$

O mecanismo de correção dos erros passa a ser

$$\begin{aligned}
N_t &= N_{t-1} + T_{t-1} + \alpha e_t, \quad 0 < \alpha < 1, \\
T_t &= T_{t-1} + \alpha\beta e_t, \quad 0 < \beta < 1, \\
F_t &= F_{t-s} + \gamma(1-\alpha)e_t, \quad 0 < \gamma < 1.
\end{aligned}$$

Os algoritmos de alisamento exponencial descritos acima possuem a vantagem de serem de simples implementação e de baixo custo

computacional. Todavia, eles não possuem embasamento estatístico e não permitem a incorporação de variáveis explicativas no processo de geração de previsões.

Uma estratégia alternativa é a 'modelagem de Box–Jenkins'. Essa modelagem utiliza a classe de modelos ARIMA e extensões. Considere, de início, o processo ARMA(p, q), definido como

$$y_t = c + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} + u_t + \theta_1 u_{t-1} + \ldots + \theta_q u_{t-q},$$

onde u_t é ruído branco, ou seja, $u_t \sim \text{RB}(0, \sigma^2)$, os ϕ's e os θ's são os parâmetros auto-regressivos e de médias móveis, respectivamente. Podemos escrever ainda

$$\phi(B) y_t = c + \theta(B) u_t,$$

onde $\phi(B)$ e $\theta(B)$ são os polinômios AR e MA usuais, i.e.,

$$\begin{aligned} \phi(B) &= 1 - \phi_1 B - \cdots - \phi_p B^p, \\ \theta(B) &= 1 + \theta_1 B + \cdots + \theta_p B^p. \end{aligned}$$

Aqui, B é o operador de defasagens, i.e., $By_t = y_{t-1}$, $B^2 y_t = y_{t-2}$, etc.

Suponha que y_t processo é integrado de ordem d, ou seja, y_t é não-estacionário, mas $\triangle^d y_t = (1 - B)^d y_t$ é estacionário, onde $\triangle$ é o operador de diferenças. Podemos modelar a série como seguindo um processo ARIMA(p, d, q), definido como

$$\phi(B)[(1 - B)^d y_t - \mu] = \theta(B) u_t,$$

em que μ é a média de $\triangle^d y_t$.

A classe de modelos ARIMA pode ser ampliada para lidar com séries sazonais. Muitas vezes não é possível transformar y_t de forma a remover a sazonalidade, ou seja, a própria sazonalidade pode apresentar um padrão dinâmico. Isto significa que há necessidade de se considerar uma sazonalidade estocástica e ajustar à série original um modelo ARIMA sazonal (SARIMA). Seja y_t a série de interesse e seja s o período de sazonalidade, como antes. Sejam

$$\Phi(B^s) = 1 - \Phi_1 B^s - \cdots - \Phi_P B^{sP}$$

o operador autorregressivo sazonal de ordem P,

$$\Theta(B^s) = 1 - \Theta_1 B^s - \cdots - \Theta_Q B^{sQ},$$

o operador de médias móveis sazonal de ordem Q, e $\triangle_s^D = (1-B^s)^D$, D indicando o número de 'diferenças sazonais'. A classe de modelos sazonais multiplicativos SARIMA$(p,d,q) \times (P,D,Q)$ é dada por

$$(1 - \phi_1 B - \cdots - \phi_p B^p)(1 - \Phi_1 B^s - \cdots - \Phi_P B^{sP})[(1-B)^d$$
$$(1 - B^s)^D y_t - \mu] = (1 - \theta_1 B - \cdots - \theta_p B^p)(1 - \Theta_1 B^s - \cdots$$
$$- \Theta_P B^{sP})u_t,$$

onde $u_t \sim \text{RB}(0, \sigma^2)$, ou ainda

$$\phi(B)\Phi(B^s)[(1-B)^d(1-B^s)^D y_t - \mu] = \theta(B)\Theta(B^s)u_t.$$

Um caso particular muito importante é o modelo 'airline'. Box e Jenkins usaram este modelo para modelar o logaritmo do número mensal de passageiros em companhias aéreas. Depois, este modelo se mostrou útil na modelagem de outras séries. Trata-se de um modelo SARIMA$(0,1,1) \times (0,1,1)$, ou seja,

$$(1-B)(1-B^s)y_t = \mu + (1+\theta_1 B)(1+\Theta_1^s B^s)u_t.$$

6.2 Aplicação: Modelagem da Arrecadação do ICMS no Brasil

Nosso objetivo é modelar o comportamento dinâmico da arrecadação do ICMS (Imposto sobre Operações Relativas à Circulação de Mercadorias e sobre Prestações de Serviços de Transporte Interestadual e Intermunicipal e de Comunicação) total e prever seu valor em dezembro de 2004 utilizando dados relativos ao período de julho de 1994 a novembro de 2004. O valor observado em dezembro de 2004, a preços de novembro do mesmo ano, foi R$ 11,741,730,000, ou seja, aproximadamente R$ 11.7 bilhões. Os dados foram obtidos do banco de dados do IPEA (`http://wwww.ipeadata.gov.br`), encontram-se expressos em milhares de reais e sua fonte é o Ministério da Fazenda/Cotepe.

A modelagem será realizada utilizando o `R`. Os dados encontram-se reunidos em um arquivo texto (ASCII) chamado `icms.dat`.

Após a inicialização do `R`, deve-se ler os dados. Ao fazê-lo, criaremos um objeto no ambiente `R` chamado `icms` onde serão armazenadas as observações.

```
> icms = scan("icms.dat")
> icms.ts = ts(icms, start=c(1994,7),
+ frequency=12)
```

O objeto `icms.ts` contém os dados formatados como uma série temporal que inicia em julho de 1994 e é observada mensalmente (`frequency = 12`). Inicialmente, desejamos visualizar os dados graficamente e calcular algumas medidas descritivas (média, mediana, desvio-padrão, etc.).

```
> plot(icms.ts)
> mean(icms.ts)
[1] 6689545
> median(icms.ts)
[1] 5962868
> sqrt(var(icms.ts))
[1] 2447499
> fivenum(icms.ts)
[1]  2459901  4853711  5962868  8405321
[5]  12150937
```

Os cinco números retornados pela função `fivenum` são mínimo, primeiro quartil, mediana, terceiro quartil e máximo. Notamos que a arrecadação média do ICMS no período (julho de 1994 a novembro de 2004) foi de R$ 6.7 bilhões, com desvio-padrão de R$ 2.5 bilhões; a arrecadação mediana foi de R$ 6 bilhões.

Em seguida, devemos analisar as funções de autocorrelação amostral e de autocorrelação parcial amostral, tanto da série em nível quanto de sua primeira diferença.

```
> acf(icms.ts)
> pacf(icms.ts)
> acf(diff(icms.ts))
> acf(diff(icms.ts))
```

Não encontramos indícios de sazonalidade, o que não está de acordo com nossa expectativa, já que arrecadações tributárias tendem a apresentar comportamentos sazonais.

Os dados com que estamos, até o momento, trabalhando encontram-se a preços correntes, ou seja, não se encontram ajustados por movimentos inflacionários. A fim de expressar os dados a preços constantes de novembro de 2004, leremos dados relativos ao Índice de Preços ao Consumidor (IPC) e utilizaremos este índice para realizar o deflacionamento desejado:

```
> ipc = scan("ipc.dat")
> ipc = ipc/100
> ipc = ipc/ipc[length(ipc)]
> icms.r = icms.ts/ipc
```

Em seguida, examinamos visualmente tanto a série em nível quanto seu logaritmo (natural):

```
> icms.r.log = log(icms.r)
> plot(icms.r, xlab="tempo", ylab="ICMS␣real")
> plot(icms.r.log, xlab="tempo",
+ ylab="log(ICMS␣real)")
```

Trabalharemos com o logaritmo dos dados, a fim de reduzir flutuações; as previsões geradas serão posteriormente exponenciadas para se obter previsões na escala original. Algumas estatísticas descritivas sobre a arrecadação real do ICMS:

```
> mean(icms.r)
[1] 9475308
> sqrt(var(icms.r))
[1] 1300045
> fivenum(icms.r)
[1]  6759850  8321933  9206320 10550639
[5]  12227921
> mean(diff(log(icms.r)))
[1] 0.003958339
> sqrt(var(diff(log(icms.r))))
[1] 0.05326186
> median(diff(log(icms.r)))
[1] 0.004188365
```

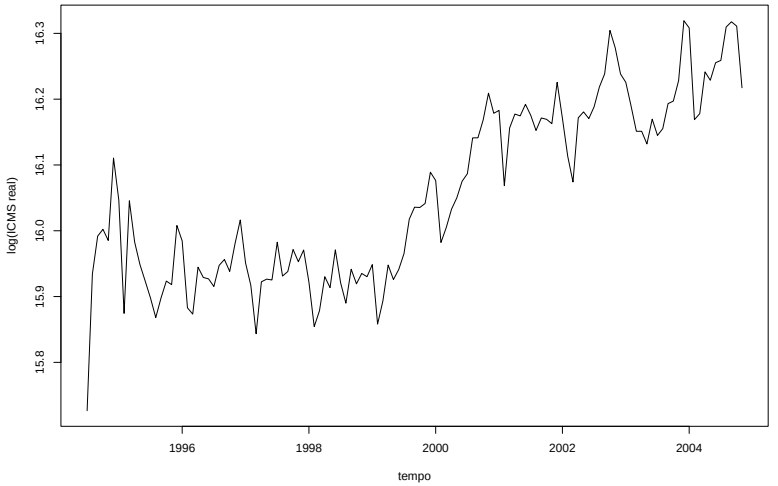

Figura 6.1: Evolução temporal do logaritmo do ICMS real no Brasil

```
> min(diff(log(icms.r)))
 [1] -0.172175
> max(diff(log(icms.r)))
[1] 0.2078939
```

Notamos que a taxa média de crescimento da arrecadação, entre meses consecutivos, foi de 0.4%. A arrecadação média (em valores de novembro de 2004) foi de R$ 9.5 bilhões, com desvio-padrão de R$ 1.3 bilhão.

A seguir, analisamos as funções amostrais de autocorrelação e autocorrelação amostral:

```
> acf(icms.r.log, lag.max=36)
> pacf(icms.r.log, lag.max=36)
> acf(diff(icms.r.log), lag.max=36)
> pacf(diff(icms.r.log), lag.max=36)
```

Notamos que: (i) a série parece ser integrada de primeira ordem, sendo, assim, não-estacionária; (ii) há sazonalidade, que é revelada

pelos picos de correlação (1, 2 e 3 no correlograma correspondem a 12, 24 e 36 defasagens, respectivamente).

Inicialmente preveremos o valor da arrecadação total do ICMS em dezembro de 2004 através do algoritmo de alisamento exponencial de Holt–Winters (aditivo):

```
> hw.fit.1 = HoltWinters(icms.r.log)
> hw.fit.1
Holt-Winters exponential smoothing without
 trend and with additive seasonal component.

Call:
 HoltWinters(x = icms.r.log)

Smoothing parameters:
 alpha:  0.7537135
 beta :  0
 gamma:  0.6929771

Coefficients:
              [,1]
a   16.2371339403
s1   0.0454481744
s2   0.0403461047
s3  -0.0361265360
s4  -0.0305452153
s5   0.0181958453
s6   0.0098216101
s7   0.0233084999
s8   0.0060877359
s9   0.0027844363
s10 -0.0026977290
s11 -0.0009882357
s12 -0.0122171962
>
> plot.ts(icms.r.log, xlim=c(1994.1, 2006.12),
+ xlab="tempo", ylab="log(ICMS)")
> p1 = predict(hw.fit.1, 12)
```

```
> lines(p1, lty=2, lwd=1.2)
 > lines(hw.fit.1$fitted[, 1], lty=3, lwd=1.2)
> p1
           Jan       Feb       Mar       Apr
2004
2005 16.27748 16.20101 16.20659 16.25533
           May       Jun       Jul       Aug
2004
2005 16.24696 16.26044 16.24322 16.23992
           Sep       Oct       Nov       Dec
2004                                 16.28258
2005 16.23444 16.23615 16.22492

> exp(p1)
           Jan       Feb       Mar       Apr
2004
2005 11727887 10864459 10925267 11470967
           May       Jun       Jul       Aug
2004
2005 11375307 11529764 11332913 11295538
           Sep       Oct       Nov       Dec
2004                                 11787876
2005 11233784 11253004 11127352
```

A previsão obtida é, assim, de R$ 11.8 bilhões (mais exatamente, R$ 11,787,876,000). O erro de previsão, definido como a diferença entre o valor previsto e o observado, foi de cerca de R$ 46 milhões, representando um erro relativo de 0.4%.

Passaremos, a seguir, à modelagem de Box–Jenkins. De início, ajustamos o modelo 'airline' e obtemos a previsão para dezembro de 2004:

```
> ajuste.1 = arima(icms.r.log, order=c(0,1,1),
+ seasonal=list(order=c(0,1,1)))
> ajuste.1

Call:
arima(x = icms.r.log, order = c(0, 1, 1),
 seasonal = list(order = c(0, 1, 1)))
```

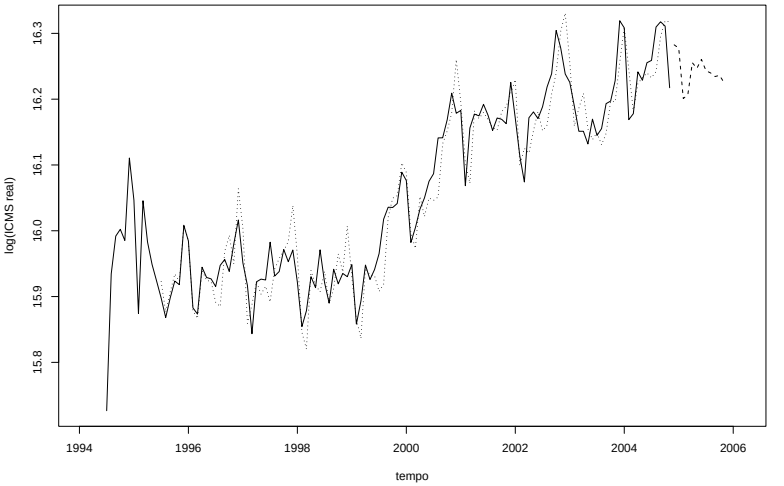

Figura 6.2: Previsão por Holt–Winters (valores reais em linha contínua, ajuste dentro da amostra em linha pontilhada e previsões futuras em linha tracejada)

```
8
9  Coefficients:
10            ma1      sma1
11        -0.3697   -0.9999
12 s.e.    0.1011    0.1976
13
14 sigma^2 estimated as 0.001672:  log likelihood
15 = 185.05,  aic = -364.1
16
17
18 > predict(ajuste.1, 12)$pred
19            Jan       Feb       Mar       Apr
20 2004
21 2005 16.26969 16.17890 16.19783 16.23826
22            May       Jun       Jul       Aug
```

```
2004
 2005 16.22898 16.24306 16.23452 16.26204
           Sep       Oct       Nov       Dec
2004                                 16.29664
2005 16.28552 16.29615 16.29322
> p2 = exp(predict(ajuste.1, 12)$pred)
> p2
           Jan       Feb       Mar       Apr
2004
2005 11636862 10626909 10830022 11276822
           May       Jun       Jul       Aug
2004
2005 11172610 11331107 11234723 11548189
           Sep       Oct       Nov       Dec
2004                                 11954760
2005 11822535 11948929 11913966

> plot.ts(icms.r.log, xlim=c(1994.1, 2006.12),
+ xlab="tempo", ylab="log(ICMS)")
> lines(icms.r.log - ajuste.1$resid, col="red")
> lines(log(p2), lty=2, col="blue")
```

Nossa previsão é, assim, de R$ 11,954,760; o erro relativo, neste caso, é de 1.8%.

Nosso próximo objetivo é a seleção do modelo SARIMA$(p, d, q) \times (P, D, Q)$ via minimização do Critério de Informação de Akaike (AIC). Para tanto, escrevemos a seguinte função, que encontra o modelo ótimo para dados valores de d, P, D, Q. Como sabemos que a série é integrada de primeira ordem, utilizaremos $d = 1$. O nome da função é `selecao.de.modelos` (algum outro nome mais informativo poderia ser usado):

```
> # funcao para selecao de modelos (P,D,Q
fixos)
> selecao.de.modelos <- function(serie=
+ icms.r.log, p.max=3, q.max=3, d=1, P=0, D=1,
+ Q=1)
+ {
+
```

```
+ # matriz para armazenar os resultados
 + M <- matrix(0,p.max+1,q.max+1)
+
+ if(P == 0 && Q == 0)
+ {
+   for(i in 0:p.max)
+   {
+      for(j in 0:q.max)
+      {
+          if(i == 0 && j == 0) M[1,1] <- NA
+          else
+             M[i+1,j+1] <- arima(serie,
+            order=c(i,d,j), seasonal=
+            list(order=c(P,D,Q)))$aic
+      }
+   }
+ }
+ else
+ {
+   for(i in 0:p.max)
+   {
+      for(j in 0:q.max)
+      {
+             M[i+1,j+1] <- arima(serie, order=
+             c(i,d,j),seasonal=list(order=
+             c(P,D,Q)))$aic
+      }
+   }
+ }
+
+ return(M)
+
+ }
```

Vejamos como utilizar esta função. Utilizando os valores *default* para d, P, D, Q e variando p e q de zero a 3, temos:

```
> M = selecao.de.modelos(icms.r.log)
> M
```

```
           [,1]       [,2]       [,3]       [,4]
 [1,] -355.0234 -364.1028 -362.1555 -361.9951
[2,] -362.2293 -362.1274 -360.4484 -361.1197
[3,] -363.3592 -361.4697 -361.1453 -361.4283
[4,] -361.5934 -360.7235 -362.2949 -358.2407
>
> which(M == min(M), arr.ind = TRUE)
     row col
[1,]   1   2
```

A minimização do AIC nos sugere, assim, o seguinte modelo: SARIMA$(0,1,1) \times (0,1,1)$, ou seja, o modelo 'airline'.

Para $Q = 0$ (mantendo inalteradas as demais quantidades), temos:

```
> M = selecao.de.modelos(icms.r.log, Q=0)
> M
           [,1]       [,2]       [,3]       [,4]
[1,]         NA -330.8475 -329.1365 -328.2120
[2,] -325.6842 -329.0308 -333.6179 -331.8163
[3,] -330.3921 -328.4218 -331.7940 -329.8270
[4,] -328.4193 -326.4164 -329.8642 -332.5980
> which(M == min(M, na.rm=TRUE), arr.ind
+ = TRUE)
     row col
[1,]   2   3
```

Aqui, o critério recomenda o uso do modelo SARIMA$(1,1,2) \times (0,1,0)$.

Trabalhemos com o segundo modelo:

```
> ajuste.2 = arima(icms.r.log, order=c(1,1,2),
 seasonal=list(order=c(0,1,0)))
> ajuste.2

Call:
arima(x = icms.r.log, order = c(1, 1, 2),
seasonal = list(order = c(0, 1, 0)))

Coefficients:
          ar1      ma1      ma2
```

```
      -0.8355  0.4881  -0.5119
s.e.   0.0608  0.1031   0.0993

sigma^2 estimated as 0.002709:  log
likelihood = 170.81,  aic = -333.62
> p3 = exp(predict(ajuste.2, 6)$pred)
> p3
          Jan       Feb       Mar       Apr
2004
2005 12895173 10922312 11269007 11788037
          May       Jun       Jul       Aug
2004
2005 11820197
          Sep       Oct       Nov       Dec
2004                            12628327
2005
```

A previsão é R$ 12,628,327,000; o erro relativo correspondente é 7.6%. Todavia, esse modelo não parece adequado, pois os resíduos associados aparentam ter comportamento sazonal. Isso pode ser visto através do segundo gráfico do painel de três gráficos de diagnóstico gerado por

```
> tsdiag(ajuste.2)
```

A seguir, tentamos encontrar o melhor modelo, por minimização do AIC, correspondente a $P = Q = 1$. O modelo selecionado foi SARIMA$(0,1,1)\times(1,1,1)$, mas não passou na análise de diagnóstico, uma vez que os resíduos demonstraram ter comportamento sazonal.

Para $P = 1, D = 1, Q = 0$, temos

```
> M = selecao.de.modelos(icms.r.log, P=1, D=1,
+ Q=0)
Warning message:
NaNs produced in: log(x)
> M
           [,1]      [,2]      [,3]      [,4]
[1,] -338.8450 -350.2087 -348.2097 -346.9554
[2,] -347.6846 -348.2093 -348.8278 -348.6142
[3,] -349.1787 -347.2332 -346.8745 -346.6370
```

```
[4,] -347.2437 -348.6473 -348.7300 -344.8170
 > which(M == min(M), arr.ind=TRUE)
     row col
[1,]   1   2
> ajuste.5 = arima(icms.r.log, order=c(0,1,1),
+ seasonal=list(order=c(1,1,0)))
> tsdiag(ajuste.5)
> p6 = exp(predict(ajuste.5, 6)$pred)
> p6
          Jan       Feb       Mar       Apr
2004
2005 11684818 10728629 10561029 10885926
          May       Jun       Jul       Aug
2004
2005 10711180
          Sep       Oct       Nov       Dec
2004                            11823303
2005
```

A previsão obtida é de R$ 11,823,303,000 à qual corresponde erro relativo de 0.7%. A inspeção visual dos gráficos de diagnóstico não sugere a rejeição do modelo.

Foi também escolhido, via minimização do AIC, o modelo SARIMA$(1,1,2)\times(0,1,0)$, mas a análise de diagnóstico forneceu evidência contra ele.

Assim, temos dois modelos SARIMA, como resultado da análise, a saber: SARIMA$(0,1,1)\times(0,1,1)$ ('airline') e SARIMA$(0,1,1)\times(1,1,0)$. Os respectivos erros relativos de previsão são 1.8% e 0.7%. O gráfico da série juntamente com os valores ajustados do modelo SARIMA$(0,1,1)\times(1,1,0)$ pode ser produzido da seguinte forma:

```
> plot.ts(icms.r.log, xlim=c(1994.1, 2006.12),
+ xlab="tempo", ylab="log(ICMS)")
> lines(icms.r.log - ajuste.5$resid, col="red")
> lines(log(p6), lty=2, col="blue")
```

Rob Hyndman (da Monash University, Austrália) desenvolveu uma coleção de funções para o `R` úteis para previsões, e as agrupou no pacote `forecast`, que se encontra disponível em `http://`

`www-personal.buseco.monash.edu.au/~hyndman/Rlibrary`. Particularmente útil é a função `best.arima`, que seleciona o melhor modelo arima variando não apenas p e q, mas também P e Q (d e D devem ser especificados pelo usuário). Esta função se encontra disponibilizada em `http://www.de.ufpe.br/~cribari/arima.R`. Antes de utilizar a função, é necessário importá-la no `R`:

```
> source("arima.R")
```

Usemos esta função com $d = 1$ e $D = 1$:

```
> best.arima(icms.r.log, d=1, D=1)
Series: icms.r.log
ARIMA(0,1,1)(0,1,1)[12] model

Coefficients:
          ma1     sma1
      -0.3697  -0.9999
s.e.   0.1011   0.1976

sigma^2 estimated as 0.001672:  log likelihood
= 185.05,  aic = -364.1
```

Notamos que o modelo selecionado é o 'airline', que já foi considerado. Usemos, agora, $d = 1$ e $D = 0$:

```
> best.arima(icms.r.log, d=1, D=0)
Series: icms.r.log
ARIMA(1,1,2)(1,0,1)[12] model

Coefficients:
          ar1     ma1      ma2    sar1     sma1
      -0.8864  0.5913  -0.3889  0.9987  -0.9584
s.e.   0.0592  0.1051   0.1008  0.0063   0.0987

sigma^2 estimated as 0.001671:  log likelihood
= 210.33,  aic = -408.66

> fit = arima(icms.r.log, order=c(1,1,2),
+ seasonal=list(order=c(1,0,1)))
> exp(predict(fit, 12)$pred)
```

```
16              Jan       Feb       Mar       Apr
17  2004
18  2005 11700500 10593339 10964530 11171090
19              May       Jun       Jul       Aug
20  2004
21  2005 11250186 11236349 11284519 11445117
22              Sep       Oct       Nov       Dec
23  2004                                  11702370
24  2005 11786081 11802897 11851785
```

O modelo selecionado foi SARIMA$(1,1,2) \times (1,0,2)$ e a previsão para dezembro de 2005 foi R$ 11,702,370,000, com erro absoluto de R$ $-39,360,000$ e erro relativo de -0.3%. Esse modelo não é descartado por uma análise de diagnóstico realizada a partir de `tsdiag(fit)`.

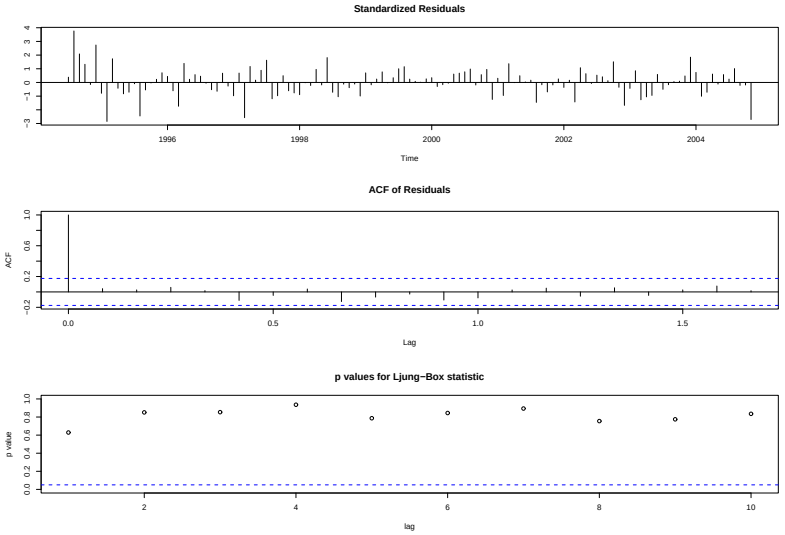

Figura 6.3: Gráficos de diagnóstico do modelo SARIMA ajustado

Uma outra função útil da coleção de funções de R. Hyndman é `fitted.Arima`, que retorna previsões um passo à frente para a série em uso. Por exemplo:

```
> plot(icms.r.log, xlab="tempo",
 + ylab="log(ICMS␣real)")
> lines(fitted.Arima(fit), lty=3, lwd=1.2)
```

produz um gráfico contendo o logaritmo natural da série (linha preta) e o conjunto de previsões um passo à frente do modelo R em uso.

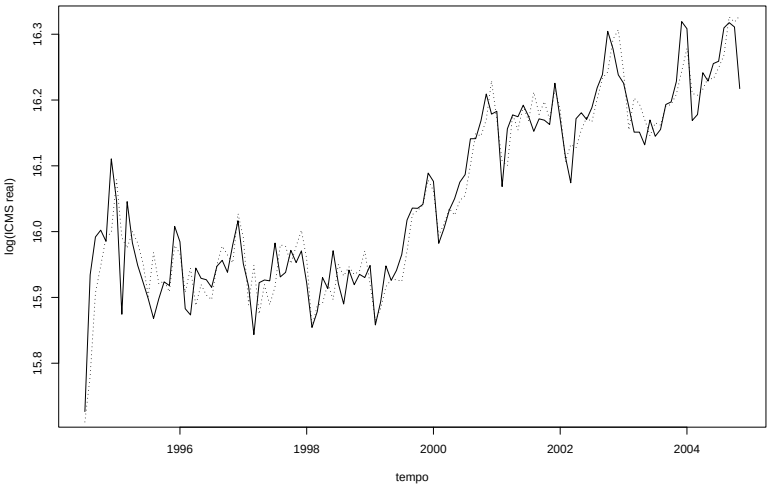

Figura 6.4: Dados e previsões SARIMA um passo à frente (linha contínua e linha pontilhada, respectivamente)

Nosso próximo objetivo é a decomposição da série observada em componentes não-observáveis; estas componentes são: tendência, sazonalidade e irregular. Ou seja, enxergamos nossa série como a soma dessas componentes *não-observáveis* e desejamos estimá-las. Para tanto, podemos usar a decomposição STL, que se encontra implementada no `R` através da função `stl`; para detalhes,

```
> help(stl)
> # ou
> ?stl
```

Podemos realizar a decomposição STL da seguinte forma:

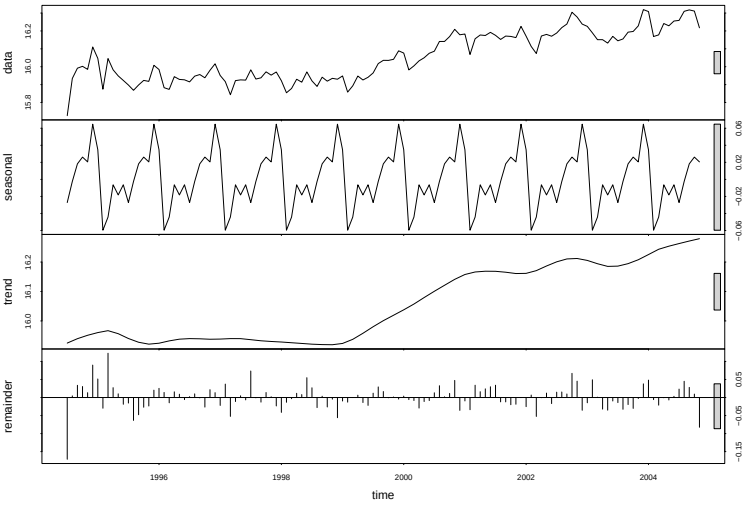

Figura 6.5: Decomposição STL

```
> icms.r.stl = stl(icms.r.log, "periodic")
 > plot(icms.r.stl)
```

Notamos, por exemplo, que há acentuado crescimento da arrecadação real do ICMS a partir de 1999. Para extrair a componente tendência do resultado produzido pela decomposição,

```
> tendencia = icms.r.stl$time.series[, "trend"]
```

Capítulo 7

Simulação Estocástica e Ensaios Monte Carlo

"Simulação Estocástica" é uma denominação genérica para um conjunto de teorias, técnicas e procedimentos que utilizam recursos de computação digital para resolver problemas quantitativos através do uso de fenômenos que se mostram como estocásticos para um observador casual. Simulação estocástica tem muitas aplicações, entre as quais figuram a comparação de procedimentos estatísticos, a resolução de integrais, o estudo e a implementação de jogos.

Para vários autores a simulação estocástica é ao mesmo tempo uma ciência e uma arte (ver, por exemplo, [9]). O lado artístico refere-se à busca de caminhos mais adequados para chegar ao resultado desejado com máxima qualidade e mínimo custo. Nesse texto foi dito que, por princípio, **nenhum** problema deve ser resolvido por simulação estocástica; a técnica é extremamente poderosa, porém, por definição, imprecisa. Por ser de grande generalidade deve ser reservada àqueles problemas para os quais a busca de uma solução exata demandaria tempo e/ou recursos inaceitavelmente caros.

Definição 1 (O problema geral). *Seja* $\boldsymbol{Y}\colon \Omega \to \mathbb{R}^k$ *um vetor aleatório* k*-dimensional e* $\psi\colon \mathbb{R}^k \to \mathbb{R}^r$ *uma função mensurável. Calcular* $\boldsymbol{\theta} = \mathrm{E}(\psi(\boldsymbol{Y}))$*, isto é, resolver a integral* $\boldsymbol{\theta} = \int\limits_{\mathbb{R}^r} \psi(\boldsymbol{y}) f(\boldsymbol{y}) d\boldsymbol{y}$

que, freqüentemente, não tem forma analítica fechada e para a qual os métodos numéricos disponíveis são pouco confiáveis ou instáveis.

Uma solução simples (que pode ser melhorada) é por Monte Carlo Força Bruta, que consiste em "aproximar" $\boldsymbol{\theta}$ por $\hat{\boldsymbol{\theta}} = n^{-1} \sum_{i=1}^{n} \psi(\boldsymbol{y}_i)$, onde $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$ são amostras independentes da variável aleatória $\boldsymbol{Y}$.

A geração de eventos de variáveis aleatórias i.i.d. com distribuição uniforme em $(0, 1)$ é essencial para construir eventos de variáveis aleatórias com outras distribuições e/ou outras estruturas de dependência. Veremos, por este motivo, técnicas de geração destes eventos. Nosso objetivo é, então, poder trabalhar com $(u_1, \ldots, u_n)$ observações do vetor aleatório $(U_1, \ldots, U_n)$ que satisfaça as seguintes propriedades:

1. $\Pr(U_{i_1} \leq u_{i_1}, \ldots, U_{i_k} \leq u_{i_k}) = \Pr(U_1 \leq u_1) \times \cdots \times \Pr(U_{i_1} \leq u_{i_k})$ para todo $k \geq 2$ e todo $1 \leq i_1 < \cdots < i_k \leq n$, isto é, as variéveis aletórias de qualquer subseqüência são independentes (independência coletiva);

2. Para cada $1 \leq i \leq N$ e para cada $u_i \in \mathbb{R}$ vale que $\Pr(U_i \leq u_i) = u_i \mathbb{I}_{(0,1)}(u_i) + \mathbb{I}_{[1,\infty)}(u_i)$, isto é, cada uma das variáveis aleatórias segue uma lei uniforme no intervalo $(0, 1)$.

Pode-se questionar se faz sentido usar um dispositivo eminentemente determinístico como um computador digital para obter eventos de variáveis aleatórias. Isto não seria uma contradição intrínseca?

O início da simulação estocástica demandava o uso de dispositivos realmente aleatórios, como contadores Geiger. Logo foram publicadas tabelas de números obtidos em condições bastante controladas de aleatoriedade, porém a demanda crescente de mais e mais valores para experiências mais e mais complexas logo fez com que estas alternativas deixassem de ser práticas, levando à necessidade de construir algoritmos com este propósito.

Novamente, todo algoritmo é intrínsecamente determinístico. A solução é construir algoritmos que produzam seqüências de números que, vistos por um observador que não conhece a seqüência de instruções, pareçam satisfazer as propriedades 1 e 2. Este observador dispõe, em princípio, de um tempo limitado para encontrar evidência

que lhe permita rejeitar a seqüência gerada por não ter as propriedades desejadas. Estas seqüências são chamadas *pseudo-aleatórias*.

As duas propriedades básicas de um algoritmo $f\colon \mathbb{R}^k \to \mathbb{R}$ capaz de gerar seqüências pseudo-aleatórias são

1. A função $u_i = f(u_{i-1}, \ldots, u_{i-k})$ deve ser de custo computacional relativamente baixo.

2. Deve ser difícil conhecer u_{i-1} dado só o conhecimento dos antecedentes $(u_i, u_{i-2}, \ldots, u_{i-k})$.

Outras propriedades desejáveis para um gerador deste tipo de seqüências são a repetibilidade, a portabilidade e a rapidez. A referência fundamental para este tema é o livro [30].

Basta gerar números inteiros não-negativos e dividí-los pelo máximo valor possível da seqüência. Quanto maior for este divisor, melhor será a aproximação a números reais no intervalo $[0, 1]$. O custo computacional de realizar aritmética inteira é menor que o de fazê-lo em ponto flutuante, e os resultados são mais previsíveis e menos dependentes da máquina e do sistema operacional.

Veremos a seguir alguns algoritmos básicos para a geração de seqüências pseudo-aleatórias uniformes.

7.1 Geradores Uniformes

Um dos primeiros algoritmos para a geração de seqüências pseudo-aleatórias uniformes foi proposto por von Neumann em 1952 para aplicações de computação a problemas de física nuclear. O método, também conhecido como *mid-square*, consiste nas seguintes instruções:

1. Definir um número u_0 de quatro dígitos decimais e atribuir $i = 0$.

2. Calcular u_i^2 e, eventualmente, agregar zeros à esquerda para que o valor calculado possa ser escrito como $u_i^2 = d_7 d_6 \cdots d_0$, onde cada d_j é um inteiro entre 0 e 9.

3. Fazer $u_{i+1} = d_5 d_4 d_3 d_2$.

4. Atribuir $i \leftarrow i + 1$ e continuar no passo (2).

Ainda que muitas das seqüências geradas por este algoritmo sejam interessantes e exibam boas propriedades, nem sempre é possível garantir o bom comportamento a longo prazo das mesmas. Como exemplo, verifique quais seqüências são geradas a partir de $u_0 = 0$, $u_0 = 2100$ e $u_0 = 3792$.

Uma classe interessante de geradores, conhecida como *congruenciais lineares*, é definida pela recursão $u_i = (au_{i-1} + c) \bmod M$, para $i \geq 1$, onde u_0 recebe o nome de semente, M de módulo, a de multiplicador e c de incremento. Se o incremento é nulo trata-se de um gerador misto, caso contrário de um gerador multiplicativo. Como exercício sugerimos implementar um gerador deste tipo e verificar as seqüências geradas com $M = 64$, semente arbitrária e

- $a = 29$, $c = 17$,
- $a = 9$, $c = 1$,
- $a = 13$, $c = 0$,
- $a = 11$, $c = 0$.

Os principais resultados sobre estes geradores estão nos textos [30, 41]. A literatura é rica em técnicas de melhoria destes algoritmos (ver, por exemplo, as sugestões dadas por [9, 22, 30]), porém existem outras classes de geradores com propriedades mais interessantes.

Um usuário cuidadoso de plataformas computacionais que utiliza simulação estocástica deveria verificar a qualidade dos geradores disponíveis através de, por exemplo,

1. Avaliação qualitativa: gráficos para avaliar a aderência à distribuição uniforme, dependência seqüencial e ausência de vazios no espaço.
2. Testes de aderência gerais: χ^2, Kolmogorov-Smirnov etc.
3. Testes de aderência específicos: serial, do intervalo, pôquer, do máximo etc.
4. Testes de independência: de permutações, de dependência linear (ver [9]).

5. Testes consistentes: de vazios no espaço, *stringent tests* (ver [34] e as referências ali citadas).

Algumas empresas de desenvolvimento de software deveriam seguir pelo menos uma destas sugestões, já que, como se mostra em [36], as versões 97, 2000 e XP da popular planilha Excel falham na geração de seqüências pseudo-aleatórias. Segundo estes autores, os algoritmos utilizados não estão documentados e são de qualidade duvidosa. Outra plataforma que não oferece grandes garantias no que se diz respeito a seu generador de números pseudo-aleatórios é IDL (ver [10]).

Uma referência extremamente completa para este tema é o sítio Web [23].

A plataforma `Ox` v. 3.40, oferece três geradores de eventos uniformes. O usuário escolhe o gerador desejado com a função `ranseed`, fornecendo-lhe como argumento uma das seguintes possibilidades:

`"PM"` Modified Park and Miller (período aproximado 2^{32}).

`"GM"` George Marsaglia (período aproximado 2^{60}).

`"LE"` Pierre L'Ecuyer (período aproximado 2^{113}).

A mesma função `ranseed` permite informar a semente e recuperar a semente depois de ter feito funcionar o gerador.

A plataforma `R` v. 2.0.1, oferece seis geradores de eventos uniformes:

Wichmann-Hill De período aproximado $7\,10^{12}$.

Marsaglia-Multicarry De período aproximadamente igual a 2^{60}, passa em testes rigorosos.

Super-Duper De período aproximado $5\,10^{18}$, não passa em alguns testes rigorosos.

Mersenne-Twister De período $2^{19937} - 1$ e boa distribuição em espaços de dimensões inferiores a 623.

Knuth-TAOCP Este gerador é definido pela relação $u_j = (u_{j-100} - u_{j-37}) \bmod 2^{30}$ e seu período é de aproximadamente 2^{129}.

Knuth-TAOCP-2002 Uma versão atualizada e melhorada do anterior.

O usuário tem como sétima alternativa a possibilidade de fornecer um gerador próprio. Para detalhes ver [24].

Alguns dos fatores mais importantes para escolher o gerador mais adequado para cada aplicação são:

- O número de eventos que iremos necessitar para a experiência, que sempre deverá ser muito menor que o período do gerador.
- A rapidez do gerador, que não deverá comprometer os prazos de entrega dos resultados.
- A possível estrutura de correlação das seqüências, que não deverá comprometer a qualidade dos resultados.

Suponhamos que dispomos de bons geradores de seqüências pseudoaleatórias. Na próxima seção veremos um resultado muito importante para simulação estocástica.

7.2 Geração por Transformação

É comum precisarmos de eventos provenientes de variáveis aleatórias que obedecem outras distribuições, além da uniforme. Veremos um resultado de validade universal, que utiliza variáveis aleatórias com distribuição uniforme em $(0, 1)$ para construir variáveis aleatórias com qualquer distribuição. Outras técnicas podem ser vistas nos textos [19, 42].

Seja $F\colon \mathbb{R} \to [0, 1]$ a função de distribuição acumulada de uma variável aleatória contínua e F^{-1} sua inversa. Se U é uma variável aleatória que obedece uma lei uniforme em $(0, 1)$, então a variável aleatória resultante da transformação $V = F^{-1}(U)$ segue a lei caracterizada pela função F. Caso se trate de variáveis aleatórias discretas, necessitamos definir a inversa de outra forma. Neste caso, substituindo F^{-1} por $F^{-}(t) = \inf\{x \in \mathbb{R} : t \leq F(x)\}$ o resultado segue sendo válido. Temos, assim, os seguintes resultados importantes.

Teorema 1 (Inversão – Caso Geral). *Sejam $F\colon \mathbb{R} \to [0, 1]$ a função de distribuição acumulada de uma variável aleatória e F^{-} a*

sua inversa generalizada, dada por $F^-(t) = \inf\{x \in \mathbb{R} : t \leq F(x)\}$. *Se* U *é uma variável aleatória que segue uma lei uniforme no intervalo* $(0,1)$ *então* F *é a função de distribuição acumulada da variável aleatória resultante da transformação* $V = F^-(U)$.

Lema 1. *Quando a variável aleatória de interesse é contínua vale que* $F^-(t) = F^{-1}(t)$ *para todo* $t \in \mathbb{R}$.

A aplicabilidade deste resultado geral restringe-se somente à disponibilidade de boas implementações de F^{-1} ou de F^-. Dois casos célebres para os quais não há formas explícitas simples são as distribuições gaussiana e gama.

Problema 1. *Deseja-se obter ocorrências de variáveis aleatórias uniformes no intervalo próprio* (a,b), *isto é, quando* $a < b$.

Solução 1. *A distribuição da variável aleatória* $V \sim \mathcal{U}_{(a,b)}$ *é caracterizada pela função de distribuição acumulada* $F(t) = (t-a)(b-a)^{-1}\mathbb{I}_{(a,b)}(t) + \mathbb{I}_{[b,\infty)}(t)$. *Assim sendo,* $V = (b-a)U + a$ *terá a distribuição desejada quando* $U \sim \mathcal{U}_{(0,1)}$.

Problema 2. *Proponha um método para gerar ocorrências de variáveis aleatórias triangulares, cuja distribuição está caracterizada pela densidade dada na equação (3.4) na página 28.*

Solução 2. *A equação (3.5) fornece a função de distribuição acumulada destas variáveis aleatórias, enquanto a equação (3.6), na página 28, nos dá a inversa desta função. Com esta última equação, e munidos de um bom gerador de uniformes, o problema está resolvido.*

Problema 3. *Gerar observações de variáveis aleatórias que seguem a lei de Weibull-Gnedenko, caracterizada pela densidade dada na equação (3.7), página 28.*

Solução 3. *Verifique que a função de distribuição acumulada destas variáveis aleatórias é, para* $\alpha > 0$, $F(t) = (1 - e^{-\beta t^\alpha})\mathbb{I}_{\mathbb{R}_+}(t)$ *e, portanto, basta retornarmos os valores* $\left(-\beta^{-1}\ln(1-u)\right)^{1/\alpha}$, *onde* u *é obtido usando um bom gerador de ocorrências uniformes.*

Na questão anterior, precisamos calcular $1-u$? Se U segue uma lei uniforme em $(0,1)$, qual é a lei que segue a variável aleatória $1-U$? Quais as conseqüências computacionais desta constatação?

Embora o método seja geral, é freqüente encontrar situações como a da distribuição de Erlang. Verifique quão complicado é inverter a função de distribuição acumulada desta lei, dada na equação (3.9), página 28. Em casos como este é necessário fazer a inversão utilizando ferramentas numéricas ou tentar fazer uma expansão da inversa da função de distribuição acumulada.

Quando a variável aleatória desejada é discreta devemos utilizar a inversa generalizada definida no Teorema 1. Graficamente é fácil, mas computacionalmente pode ser complicado. Vejamos alguns exemplos antes de discutir formas de implementar esta técnica.

Exemplo 1. *Deseja-se gerar ocorrências de variáveis aleatórias que seguem uma lei Bernoulli com probabilidade p de sucesso. Sabemos que a função de distribuição acumulada desta lei é dada por*

$$F(t) = \begin{cases} 0 & se \quad t < 0 \\ 1-p & se \quad 0 \leq t < 1 \\ 1 & se \quad t \geq t. \end{cases}$$

Analisando graficamente esta função, podemos concluir que a transformação procurada é $Y = F^{-}(U)$*, onde*

$$F^{-}(u) = \begin{cases} 0 & se \quad 0 \leq u \leq 1-p \\ 1 & se \quad 1-p \leq u \leq 1. \end{cases}$$

De posse deste exemplo podemos propor o seguinte

Algoritmo 2. *Geração de ocorrências Bernoulli com probabilidade de ocorrência p: gerar u ocorrência de uma* $U \sim \mathcal{U}_{(0,1)}$*; se* $u \leq 1-p$ *retornar* 0*, caso contrário retornar* 1*.*

É imediata a generalização para qualquer vetor de probabilidade $\boldsymbol{p} = (p_1, p_2, \ldots)$, desde que se disponha explicitamente de tal vetor. Temos, assim, a seguinte proposta:

Algoritmo 3. *O método de geração de ocorrências de variáveis aleatórias discretas pelo método de busca seqüencial consiste em, dispondo do vetor de probabilidades* $\boldsymbol{p} = (p_0, p_2, \ldots, p_n)$*, executar os seguintes passos:*

1. *Atribuir* $y = 0$*,* $s = p_0$ *e gerar u ocorrência de* $U \sim \mathcal{U}_{(0,1)}$

2. Enquanto $u > s$ fazer

(a) $y \leftarrow y + 1$

(b) $s \leftarrow s + p_x$

3. Retornar y

Há três grandes problemas com este algoritmo. O primeiro é que a soma acumulada na variável s pode acumular grandes erros de arredondamento. O segundo é que a avaliação das probabilidades p_y a partir das suas expressões analíticas pode ser muito cara computacionalmente e/ou sujeita a grandes imprecisões numéricas. O terceiro é que o número de avaliações até podermos retornar o valor y, i.e., o tempo até sair do laço 2, pode ser muito grande. Por estas razões, em [19] sugere-se sempre procurar alternativas mais eficientes, e deixar este método geral como último recurso a ser empregado.

Veremos a seguir exemplos onde o cômputo das probabilidades pode ser realizado de forma mais eficiente.

Exemplo 2. *Uma situação muito freqüente é a de se desejar obter ocorrências da variável aleatória Y com distribuição uniforme nos inteiros entre 0 e $n-1$, isto é, $\Pr(Y = k) = n^{-1}$ para todo $k \in \{0, \ldots, n-1\}$. Ao invés de fazermos uma busca, basta gerar u da distribuição uniforme em $(0, 1)$ e retornar $y = [nu]$. A distribuição de Y é denotada por $\mathcal{U}_{\{0,\ldots,n-1\}}$.*

Exemplo 3. *Geração de ocorrências binomiais com parâmetros n e p. Sabemos gerar Bernoullis, e uma binomial é a soma de n Bernoullis independentes; o problema, portanto, não oferece grandes desafios, mas o algoritmo baseado nessa técnica é pouco eficiente. Mais interessante é procurar aplicar a técnica de inversão diretamente e, para isso, precisamos calcular o vetor de probabilidades $\mathbf{p} = (p_0, p_2, \ldots, p_n)$ associado aos valores $(0, 1, \ldots, n)$. Lembrando que a distribuição binomial é caracterizada pela equação* (3.1) *(página 26), a tarefa é simples. Todavia, podemos fazer melhor do que reavaliar essa expressão da equação para cada $k \in \{0, 1, \ldots, n\}$, com todos os possíveis erros de arredondamento. Basta constatar que existe uma recursão útil que começa em $p_0 = \Pr(Y = 0) = (1-p)^n$*

e que prossegue com

$$p_{j+1} = \frac{n-1}{j+1}\frac{p}{1-p},$$

que é numericamente mais estável.

Exemplo 4. *Geração de ocorrências Poisson de parâmetro* λ. *Basta constatar que neste caso* $\Pr(Y = k) = e^{-\lambda}\lambda^k/k!$. *Podemos, então, escrever* $p_{j+1} = \lambda p_j/(j+1)$.

Para que o procedimento baseado em regras recursivas seja eficiente é necessário que os valores calculados sejam armazenados entre chamadas da rotina. A linguagem `C` oferece o recurso de variáveis estáticas, que são preservadas após a chamada a uma função. Ainda nesta linguagem, é útil o uso de dimensionamento dinâmico de vetores já que, por exemplo no caso da lei Poisson, não sabemos *a priori* o maior valor a ser observado na geração e, com isso, também não é possível prever o tamanho vetor de probabilides que será necessário.

7.3 Método de Aceitação-Rejeição

Em muitas situações não se dispõe da função de distribuição acumulada da distribuição alvo em forma tratável como, por exemplo, ao lidar com a distribuição gaussiana. Em outras situações a sua inversa não é tratável. Um método muito geral para lidar com estes casos é o que se baseia na aceitação e rejeição. Suponhamos que queremos gerar eventos da distribuição contínua caracterizada pela densidade f; este método requer dois ingredientes:

1. um bom gerador de ocorrências uniformes
2. um gerador de ocorrências da distribuição contínua $\mathcal{D}$, escolhida de tal maneira que existe uma constante M tal que a densidade g que caracteriza a distribuição $\mathcal{D}$ satisfaz $f(x) \leq Mg(x)$ para todo x real.

Teorema 4. *O seguinte algoritmo produz observações de uma variável aleatória cuja distribuição é caracterizada pela densidade* f*:*

Algoritmo 5 (Algoritmo de Aceitação-Rejeição). *Executar*

1. *Gerar $y = Y(\omega)$ ocorrência da variável aleatória com distribuição $\mathcal{D}$.*
2. *Gerar $u = U(\omega)$ ocorrência da variável aleatória com distribuição $\mathcal{U}(0,1)$.*
3. *Se $u \leq f(y)/(Mg(y))$ retornar y, caso contrário voltar ao início.*

Demonstração. A observação y retornada pelo algoritmo de aceitação-rejeição, tem distribuição dada por

$$\Pr(Z \leq t) = \Pr\Big(X \leq t \mid U \leq \frac{f(y)}{Mg(y)}\Big) = \frac{\Pr\left(Y \leq t, U \leq \frac{f(Y)}{Mg(Y)}\right)}{\Pr\left(U \leq \frac{f(Y)}{Mg(Y)}\right)};$$

escrevendo esta probabilidade em forma de integrais temos

$$\Pr(Z \leq t) = \frac{\int_{-\infty}^{t}\left(\int_0^{f(v)/(Mg(v))} du\right) g(y)dy}{\int_{-\infty}^{\infty}\left(\int_0^{f(v)/(Mg(v))} du\right) g(y)dy} = \frac{\frac{1}{M}\int_{-\infty}^{t} f(y)dy}{\frac{1}{M}\int_{-\infty}^{\infty} f(y)dy} = \int_{-\infty}^{t} f(y)dy.$$

□

Exemplo 5. *Gaussiana a partir de Cauchy: lembrando que a densidade Cauchy padrão é dada por*

$$g(x) = \frac{1}{\pi(1+x^2)}$$

e que é imediato gerar ocorrências desta lei pelo método de inversão, basta verificar que a razão da densidade gaussiana padrão para g é dada por

$$\sqrt{\frac{\pi}{2}}(1+x^2)\exp\{-x^2/2\},$$

cujo gráfico é mostrado na Figura 7.1.

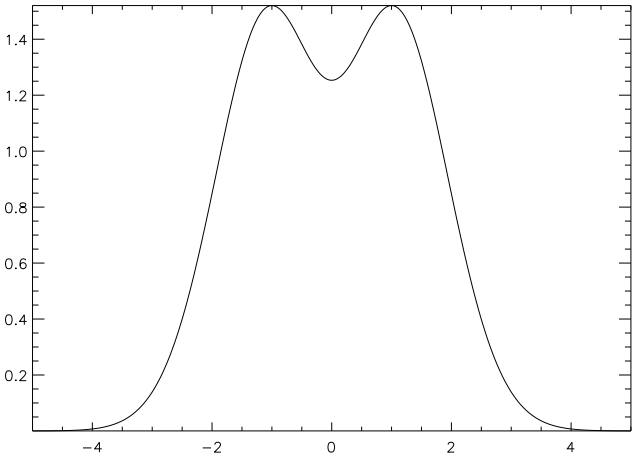

Figura 7.1: Razão das densidades Cauchy padrão e gaussiana padrão.

Esta razão é, portanto, maximizada em $x = \pm 1$*, onde vale* $\sqrt{2\pi/e}$*. Assim sendo, temos que* $f(x) \leq \sqrt{2\pi/e}\, g(x)$*, e o algoritmo fica especificado. Contudo, como gerar as ocorrências Cauchy? Verifique que a função de distribuição acumulada da Cauchy padrão é dada por* $F(t) = (2 \arctan t + \pi)/(2\pi)$ *e, portanto, se* U *segue uma lei uniforme em* $(0, 1)$ *teremos que* $X = \tan(\pi(U - 1/2))$ *seguirá uma lei Cauchy padrão. A Tabela 7.1, onde* 'T' *denota 'True', isto é, condição verificada, e* 'F' *o contrário, mostra a execução passo a passo deste algoritmo.*

Exemplo 6. *Como já sabemos gerar ocorrências de variáveis aleatórias com distribuição exponencial, podemos gerar ocorrências de variáveis aleatórias com distribuição de Laplace padrão, cuja densidade é dada por* $g(x) = \exp(-|x|)/2$*. Esta densidade será útil para gerar ocorrências de variáveis aleatórias gaussianas padrão, pois como* $f(x)/g(x) \leq \sqrt{2/(\pi e)}$ *o algoritmo fica especificado.*

Tabela 7.1: Ilustração do método de geração de gaussianas através de ocorrências Cauchy.

U	$X = \tan(\pi(U - 1/2))$	$f(X)/(Mg(X))$	V	$V \leq f(X)/(Mg(X))$	Y
0.002829	-112.494	0.00000000	0.825308	F	—
0.134913	-0.076306	0.82675011	0.386090	T	-0.076306
0.041035	-0.389111	0.87997152	0.583836	T	-0.389111
0.940134	-0.822105	0.98535727	0.544890	T	-0.822105
0.411034	1.011870	0.99992938	0.968074	T	1.011870
0.484675	1.637850	0.79388186	0.808003	F	—
0.169943	0.033904	0.82483407	0.007725	T	0.033904
0.325046	0.574104	0.92953568	0.628396	T	0.574104
0.269194	0.360159	0.87280824	0.455025	T	0.360159
0.552381	2.868500	0.12430593	0.859248	F	—

Uma consideração computacional relevante é o número de vezes que a ocorrência $Y(\omega)$ será descartada. É fácil ver que o número médio de tentativas até uma aceitação é dado por M^{-1} e, portanto, quanto menor esta quantidade mais interessante o algoritmo. Mas esta não é a única consideração, já que também deverá ser levado em conta o custo de gerar cada observação da distribuição caracterizada pela densidade g. De fato, é possível provar a seguinte proposição:

Proposição 6. *O número de tentativas até a aceitação de uma ocorrência de g do método de aceitação-rejeição segue uma lei geométrica com média M, isto é,*

$$\Pr(C = k) = (1 - p)^{k-1} p,$$

com $p = M^{-1}$. *Incidentalmente, a variância deste número de ensaios é* $(1 - p)p^{-2}$.

7.4 Método de Composição

O método de composição é útil por si só, e também em conjunção com os métodos vistos nas seções anteriores. A técnica pode ser empregada para gerar ocorrências de distribuições cuja densidade é da forma

$$f(x) = \sum_i p_i f_i(x),$$

onde $\mathbf{p} = (p_1, p_2, \ldots)$ é um vetor de probabilidades e $(f_1, f_2, \ldots)$ são densidades. Para gerar uma ocorrência de X deve-se, primeiro, escolher um índice (inteiro) segundo as probabilidades $\mathbf{p}$, por exemplo o índice j. Feito isto, gera-se uma ocorrência da distribuição caracterizada pela densidade f_j.

Uma das mais importantes aplicações deste método é para estudos de robustez. É interessante, em geral, verificar o comportamento de estimadores perante amostras contaminadas. Existem vários tipos de contaminação, mas a decorrente da aparição de observações espúrias é um dos mais perigosos. Imagine a situação de haver derivado o estimador $\hat{\theta}$ para ser aplicado a amostras independentes e identicamente distribuídas segundo uma lei $\mathcal{D}$. Qual será o comportamento deste estimador se parte da amostra vier da distribuição $\mathcal{D}'$ diferente de

$\mathcal{D}$? Fixando idéias, e ilustrando com o caso gaussiano, um modelo bastante empregado é aquele em que a variável aleatória de interesse obedece uma lei $N(\mu_1, \sigma_1^2)$ com probabilidade p e uma lei $N(\mu_2, \sigma_2^2)$ com probabilidade $1-p$ (suponha os que parâmetros das duas densidades são distintos). A densidade de uma variável aleatória com esta distribuição é dada por

$$f(x) = \frac{p}{\sqrt{2\pi\sigma_1^2}} \exp\left\{-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right\} + \frac{1-p}{\sqrt{2\pi\sigma_2^2}} \exp\left\{-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right\}. \tag{7.1}$$

A despeito da dificuldade de lidar diretamente com a densidade dada na equação (7.1), a geração de ocorrências desta lei é imediata dispondo de um gerador de gaussianas e de um gerador de Bernoullis, através da técnica de composição. É conveniente notar que o tipo de contaminação que pode assim ser gerado é muito geral, não restringindo-se às leis mostradas nem a um único contaminante (ver, por exemplo, o trabalho [11]).

7.5 Experiências Monte Carlo

A experiência Monte Carlo mais simples que podemos montar para resolver o problema de comparar a qualidade dos estimadores propostos para os parâmetros da distribuição gama é a já definida Monte Carlo Força Bruta.

Para isso, basta simular uma certa quantidade grande de eventos, digamos `n_rep` por 'replicações', como os que mostramos nos capítulos anteriores e registrar cada um dos estimadores. Fazendo isso poderemos calcular a média, a variância e outras quantidades sobre os `n_rep` eventos disponíveis; estamos ainda muito longe de ter os eventos infinitos necessários para poder calcular esperanças, mas uma escolha criteriosa do número de replicações pode dar-nos uma idéia muito boa do comportamento das variáveis aleatórias que nos interessam.

Porém, ao fazer somente o que prescrevemos, estaremos comparando o comportamento dos estimadores em um único ponto do espaço paramétrico Θ. Para que o estudo seja completo deveríamos, em princípio, varrer todo este conjunto. Por não ser possível, porque por exemplo é um contínuo, podemos selecionar alguns pontos

interessantes e supor que o comportamento dos estimadores pode ser completamente inferido a partir deles.

Outro fator que deve ser modificado para dar uma boa semelhança ao comportamento de estimadores é o tamanho das amostras consideradas. Quanto mais fatores façamos intervir mais completo será nosso estudo, mas pagaremos por isso em tempo de computação.

O trabalho [8] faz uma série de sugestões a respeito de como montar este tipo de experiências, bem sobre formas eficazes de mostrar os resultados. Sugerimos consultar o artigo [13] para um exemplo de aplicação.

Agradecimentos

Alejandro C. Frery agradece o apoio dado pela Fundação de Amparo à Pesquisa do Estado de Alagoas (FAPEAL), através do projeto PRONEX 2003.002, para participar do XXV Colóquio Brasileiro de Matemática. Os autores agradecem ao CNPq pelo apoio parcial à preparação deste texto através dos projetos 300989/97-0, 620026/2003-0, 55.2076/02-3 e 307577/2003-1.

Referências Bibliográficas

[1] E. B. Berndt, B. Hall, R. Hall, J. Hausman. Estimation and inference in nonlinear structural models. *Annals of Economic and Social Measurement*, 3/4:653–665, 1974.

[2] P. J. Bickel, K. A. Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics*, vol. 1. Prentice-Hall, NJ, 2 ed., 2001.

[3] I. O. Bohachevsky, M. E. Johnson, M. L. Stein. Generalized simulated annealing for function optimization. *Technometrics*, 28(3):209–217, 1986.

[4] H. Bolfarine, M. C. Sandoval. *Introdução à Inferência Estatística*. Coleção Matemática Aplicada. Sociedade Brasileira de Matemática, Rio de Janeiro, 2001.

[5] T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31:307–327, 1986.

[6] G. E. P. Box, G. M. Jenkins, G. C. Reinsel. *Time Series Analysis: Forecasting and Control*, 3. ed. Englewood Cliffs, Prentice, 1994.

[7] P. J. Brockwell, R. A Davis. *Introduction to Time Series and Forecasting*, 2 ed. New York, Springer-Verlag, 2002.

[8] O. H. Bustos, A. C. Frery. Reporting Monte Carlo results in statistics: suggestions and an example. *Revista de la Sociedad Chilena de Estadística*, 9(2):46–95, 1992.

[9] O. H. Bustos, A. C. Frery. *Simulação estocástica: teoria e algoritmos (versão completa)*. Monografias de Matemática, 49. CNPq/IMPA, Rio de Janeiro, RJ, 1992.

[10] O. H. Bustos, A. C. Frery. Statistical functions and procedures in IDL 5.6 and 6.0. *Computational Statistics and Data Analysis*, in press.

[11] O. H. Bustos, M. M. Lucini, A. C. Frery. M-estimators of roughness and scale for GA0-modelled SAR imagery. *EURASIP Journal on Applied Signal Processing*, 2002(1):105–114, 2002.

[12] R. H. Byrd, P. Lu, J. Nocedal, C. Zhu. A limited memory algorithm for bound constraints optimization. *SIAM Journal on Scientific Computing*, 16:1190–1208, 1995.

[13] F. Cribari-Neto, A. C. Frery, M. F. Silva. Improved estimation of clutter properties in speckled imagery. *Computational Statistics and Data Analysis*, 40(4):801–824, 2002.

[14] F. Cribari-Neto, S. G. Zarkos. R: yet another econometric programming environment. *Journal of Applied Econometrics*, 14:319–329, 1999.

[15] F. Cribari-Neto, S. G. Zarkos. Econometric and statistical computing using Ox. *Computational Economics*, 21:277–295, 2003.

[16] P. Dalgaard. *Introductory Statistics with R*. Statistics and Computing. Springer, New York, 2002.

[17] W. C. Davidon. Variable metric method for minimization. Technical Report ANL-5990 (revised), Argonne National Laboratory, 1959.

[18] W. C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1:1–17, 1991.

[19] L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, 1986.

[20] J. A. Doornik. *Object-Oriented Matrix Programming Using Ox*. Timberlake Consultants Press & Oxford, London, 3 ed., 2002.

[21] A. C. Frery, F. Cribari-Neto, M. O. Souza. Analysis of minute features in speckled imagery with maximum likelihood estimation. *EURASIP Journal on Applied Signal Processing*, 2004(2004):2476–2491, 2004.

[22] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Statistics and Computing. Springer, New York, 2000.

[23] P. Hellekalek, G. Wesp, J.-W. Kim. Random number generators: The pLab project home page, 2003.
`http://random.mat.sbg.ac.at`

[24] K. Hornik. Frequently asked questions on R, 2002.
`http://www.r-project.org`

[25] B. James. *Probabilidade: um Curso em Nível Intermediário*. Projeto Euclides. Instituto de Matemática Pura e Aplicada, Rio de Janeiro, 1981.

[26] N. L. Johnson, S. Kotz, N. Balakrishnan. *Continuous Univariate Distributions*. John Wiley & Sons, New York, 2 ed., 1994.

[27] N. L. Johnson, S. Kotz, N. Balakrishnan. *Continuous Univariate Distributions*, vol. 2. John Wiley & Sons, New York, 2 ed., 1995.

[28] N. L. Johnson, S. Kotz, A. W. Kemp. *Univariate Discrete Distributions*. John Wiley & Sons, New York, 2 ed., 1993.

[29] S. Kirkpatrick, C. D. Gelatt, M. P. Vechhi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[30] D. E. Knuth. *The Art of Computer Programming*, vol. 2 (Seminumerical Algorithms). Addison-Wesley, 3 ed., 1997.

[31] W. J. Krzanowski. *Recent Advances in Descriptive Multivariate Analysis*. Claredon Press, Oxford, 1995.

[32] J. Maindonald, J. Braun. *Data Analysis and Graphics with R: an Example-based Approach*. Cambridge, Cambridge, 2003.

[33] C. F. Manski. *Analog Estimation Methods in Econometrics*. Chapman & Hall, New York, 1988.
`http://elsa.berkeley.edu/books/analog.html`

[34] G. Marsaglia, W. W. Tsang. Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, 7(3):1–8, 2002.

[35] P. McCullagh, J. A. Nelder. *Generalized Linear Models*. Chapmann and Hall, New York, 2 ed., 1989.

[36] B. D. McCullough, B. Wilson. On the accuracy of statistical procedures in Microsoft Excel 2000 and Excel XP. *Computational Statistics and Data Analysis*, 40(4):713–721, 2002.

[37] P. A. Morettin, C. M. C. Toloi. *Análise de Séries Temporais*. Editora Edgar Blücher, São Paulo, 2004.

[38] J. Nocedal, S. J. Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

[39] W. H. Press, B. P. Flannery, S. A. Teulosky, W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University, 2 ed., 1992.

[40] J. Racine, R. Hyndman. Using R to teach econometrics. *Journal of Applied Econometrics*, 17:175–189, 2002.

[41] B. D. Ripley. *Stochastic Simulation*. Wiley, New York, 1987.

[42] C. P. Robert, G. Casella. *Monte Carlo Statistical Methods*. Springer Texts in Statistics. Springer, New York, 2000.

[43] E. R. Tufte. *Envisioning Information*. Graphics Press, 1990.

[44] E. R. Tufte. *Visual & Statistical Thinking: Displays of Evidence for Decision Making*. Graphics Press, 1997.

[45] E. R. Tufte. *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphic Press, 1997.

[46] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 2 ed., 2001.

[47] K. L. P. Vasconcellos, S. G. Silva. Corrected estimates for student t regression models with unknown degrees of freedom. *Journal of Statistical Computation and Simulation*, 2005. In press.

[48] W. N. Venables, B. D. Ripley. *S Programming*. Springer-Verlag, New York, 2000.

[49] W. N. Venables, B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, 4 ed., 2002.

[50] W. N. Venables, D. M. Smith. *An Introduction to R*. Network Theory Limited, UK, 2001.