57ªRBras

Reunião Anual da Região Brasileira da

Sociedade Internacional de Biometria

MINICURSO

Métodos Computacionais para Inferência com Aplicações em R

Wagner Hugo Bonat

Elias Teixeira Krainski

Paulo Justiniano Ribeiro Jr

Walmes Marques Zeviani

Métodos Computacionais para Inferência com Aplicações em R

Wagner Hugo Bonat
Elias Teixeira Krainski
Paulo Justiniano Ribeiro Jr
Walmes Marques Zeviani

Laboratório de Estatística e Geoinformação (LEG)
http://www.leg.ufpr.br
Departamento de Estatística
Universidade Federal do Paraná (UFPR)

Complementos online: http://www.leg.ufpr.br/mcie
Contato: mcie @ leg.ufpr.br

57ª Reunião anual da RBRAS
Região Brasileira da
Sociedade Internacional de Biometria
ESALQ/USP
Piracicaba - SP - Brasil
05 a 09 de Maio de 2012

Prefácio

A ideia de verossimilhança como instrumento para avaliar a evidência contida nos dados está no centro dos fundamentos da metodologia estatística. Embora formalizada nos trabalhos de R.A. Fisher nos anos 20, apenas muitas décadas depois e principalmente com as possibilidades abertas pela computação estatística, que pôde ser explorada, investigada, aplicada, modificada e expandida nas mais diferentes formas.

A necessidade de computação em estatística está presente desde sua origem, seja de forma manual ou, na agora onipresente, forma eletrônica com o uso de computadores. O desenvolvimento de linguagens e aplicativos para computação estatística ampliam rápida e largamente as possibilidades de geração e tratamento de dados. Linguagens interpretadas, direcionadas e/ou adaptáveis para computação estatística diminuem dramaticamente a distância entre programação e uso de aplicativos permitindo usuários investigar possibilidades e conceitos, experimentar ideias, adaptar códigos, implementar protótipos com grande flexibilidade ainda que sem um investimento proibitivo no domínio dos recursos utilizados. Em particular os projetos de *software livre* cumprem tal papel sem impor obstáculos ao usuário. Neste contexto o *Projeto* R *de Computação Estatística* iniciado na década de 90 e com a primeira versão lançada no ano 2000, tem uma impactante contribuição e larga abrangência que, em muito ultrapassa os limites da área de estatística. A linguagem já imprimiu uma marca indelével no conjunto de recursos disponíveis para interessados em computação e métodos estatísticos.

O presente texto situa-se na interface entre métodos de inferência estatística baseada em verossimilhança e métodos computacionais (com implementações em ambiente R). Sem nos aprofundarmos em nenhuma das duas áreas, procuramos ilustrar suas conexões através de diversos exemplos básicos de modelagem estatística, na expectativa que o texto possa servir como material introdutório ao leitor e facilitar seu caminho para construir suas próprias implementações.

O material foi motivado por nossa experiência em estudos e disciplinas

do LEG/UFPR nos últimos anos. Procuramos mesclar a discussão de princípios básicos de inferência estatística, com ênfase em métodos baseados na função de verossimilhança, com a implementação computacional. Nossa estratégia usual é a de escrever nossas próprias funções, na forma de protótipos, para melhor desenvolver a intuição sobre as características dos modelos e métodos estatísticos em discussão. Desta forma nossas funções são predominantemente ilustrativas, privilegiando a facilidade de leitura e entendimento e não devem ser vistas como implementações definitivas nem tampouco tentam explorar os mais eficientes usos da linguagem, ainda que alguns cuidados para evitar problemas numéricos sejam tomados na definição de certas operações. Por vezes os resultados são comparados com os fornecidos por funções do R e alguns de seus pacotes. Seguimos a sugestão de que *"programming is the best way to debug your ideias"*.

Nosso público alvo principal são alunos de graduação com alguma exposição anterior a conceitos de inferência estatística e ao uso do ambiente R. Outros potenciais interessados são alunos em início de pós-graduação e/ou profissionais que tenham interesse em se familiarizar com elementos de programação em R para inferência estatística. Incentivamos os leitores do material a nos enviar comentários, sugestões e correções.

O texto é permeado de códigos em linguagem R que são identificados pelo uso de fonte estilo `VERBATIM como esta`. Um tratamento especial é dado a funções em R que são definidas dentro de caixas em destaque. Tipicamente estas definem funções implementando alguma metodologia ou alguma função de verossimilhança a ser chamada por funções otimizadoras. Um material *online* complementa o texto com exemplos e informações adicionais e está disponível em *http://www.leg.ufpr.br/mcie*.

Todo o material é produzido utilizando *software* livre. As implementações de métodos e algoritmos é toda feita no ambiente **R** de computação estatística. O texto é escrito utilizando LaTeX e a integração com o **R** pelo mecanismo **Sweave**. Os recursos são utilizados em sistema operacional **LINUX**, incluindo a página *web* disponibilidade em ambiente **Dokuwiki** em um servidor **Apache**.

A escrita do material foi motivada pelas oportunidades de apresentar em 2012 minicursos no programa de verão/2012 do DEX/UFLA, durante a 57ªRBras e no 20ºSINAPE. Em cada uma destas ocasiões expandimos o material apresentando anteriormente apresentando novos tópicos sequenciais. Somos gratos às comissões organizadoras dos eventos pelas oportunidades e incentivo.

W.H.B., P.J.R.Jr, E.T.K. & W.M.Z.
Curitiba, abril, 2012

Sumário

Prefácio **iii**

1 Verossimilhança **1**
1.1 Estimação pontual 4
1.2 Intervalos de confiança 7
1.3 Propriedades do estimador 8
1.4 Testes de hipóteses 13
1.4.1 Teste da razão de verossimilhança 14
1.4.2 Teste de Wald 15
1.4.3 Teste escore 15
1.5 Exemplo 1 - Estimação pontual 15
1.6 Exemplo 2 - Intervalos de confiança 18
1.7 Exemplo 3 - Testes de hipóteses 25
1.8 Exemplo 4 - Reparametrização 29
1.9 Exemplo 5 - Distribuição Gaussiana 38
1.9.1 Dados intervalares 43
1.10 Exemplo 6 - Distribuição Gama 46
1.10.1 Parametrizações para Gama 54
1.11 Exemplo 7 - Distribuição Binomial Negativa 60
1.12 Tratando tudo numericamente 62

2 Modelos de regressão **69**
2.1 Regressão Poisson 72
2.2 Regressão Simplex 76
2.3 Modelo contagem-Gama 82

3 Modelos de regressão com efeitos aleatórios **91**
3.1 Modelo geoestatístico 93
3.2 Verossimilhança Marginal 99
3.2.1 Simulação de modelo Poisson com intercepto aleatório 101
3.3 Técnicas de integração numérica 103
3.3.1 Método Trapezoidal 105

3.3.2 Método de Simpson 1/3 106
3.3.3 Quadratura de Gauss-Hermite 108
3.3.4 Adaptativa Gauss-Hermite e Aproximação de Laplace 111
3.3.5 Integração Monte Carlo 114
3.4 Modelo Poisson com Intercepto aleatório 117
3.5 Poisson com efeito aninhado 121
3.6 Modelo Beta longitudinal 128
3.7 Modelo de Teoria de Resposta ao Item 132
3.8 Modelo linear dinâmico 136
3.8.1 Filtro de Kalman e verossimilhança 137
3.8.2 Exemplo simples 138
3.8.3 Exemplo de regressão dinâmica 142
3.9 Referências 148

Referências Bibliográficas **150**

Capítulo 1

Verossimilhança

A abordagem estatística para análise e resumo de informações contidas em um conjunto de dados, consiste na suposição de que existe um mecanismo estocástico gerador do processo em análise. Este mecanismo é descrito através de um modelo probabilístico, representado por uma distribuição de probabilidade. Em situações reais a verdadeira distribuição de probabilidade geradora do processo é desconhecida, sendo assim, distribuições de probabilidade adequadas devem ser escolhidas de acordo com o tipo de fenômeno em análise. Por exemplo, se o fenômeno em estudo consiste em medir uma característica numérica de um grupo de indivíduos em uma escala contínua, distribuições com este **suporte** devem ser escolhidas. O **suporte** de uma distribuição de probabilidade informa qual o domínio da função, ou seja, quais são os valores que a variável aleatória pode assumir. Considere o caso da distribuição Gaussiana, o **suporte** é a reta real, no caso da distribuição Gama o suporte é apenas os reais positivos. Um cuidado adicional deve ser dado quando a variável de interesse é discreta, por exemplo contagens, onde é comum atribuir uma distribuição de Poisson que tem **suporte** nos naturais positivos.

Em quase todos os problemas de modelagem estatística não existe uma única distribuição de probabilidade que pode representar o fenômeno. Porém, na maioria das situações assume-se que a distribuição de probabilidade geradora do processo é conhecida, com exceção dos valores de um ou mais **parâmetros** que a indexam. Por exemplo, considere que o tempo de vida de um tipo de componente eletrônico tem distribuição exponencial com **parâmetro** λ, mas o valor exato de λ é desconhecido. Se o tempo de vida de vários componentes de mesmo tipo são observados, com estes dados e qualquer outra fonte relevante de informação que esteja disponível, é possível fazer **inferência** sobre o valor desconhecido do parâmetro λ. O

processo de **inferência** consiste em encontrar um valor mais plausível de ser λ, bem como, informar um intervalo para o qual acredita-se conter o verdadeiro valor de λ, além de decidir ou opinar se λ é igual, maior ou menor que algum valor previamente especificado. Em alguns problemas há ainda interesse em fazer previsões sobre possíveis valores do processo, por exemplo, em outros tempos ou locais.

Em implementações computacionais para inferência estatística, deve-se sempre estar atento ao **espaço paramétrico** (Θ) de um modelo probabilístico. No caso, do tempo de vida de componentes eletrônicos, assumindo que a distribuição exponencial é adequada e esta sendo indexada pelo parâmetro λ, de acordo com a construção do modelo exponencial, tem-se que o **espaço paramétrico** de λ é dado pelo conjunto dos reais positivos. Em outras situações o espaço paramétrico pode ser todo o conjunto dos reais como no caso de média μ de um modelo Normal, com média μ e variância σ^2, enquanto que para σ^2 o espaço paramétrico restringe-se aos reais positivos. Outro caso comum são modelos em que o parâmetro representa alguma proporção e tem como espaço paramétrico o intervalo $[0,1]$ Estas restrições precisam ser levadas em consideração no processo de inferência e são de fundamental importância para o sucesso de muitos algoritmos de maximização numérica. Não raramente nas implementações computacionais são feitas reparametrizações com novos parâmetros com valores na reta real, com resultados transformados de volta ao espaço original ao final.

Partindo destes conceitos, um fenômeno aleatório ou estocástico é descrito minimamente por uma distribuição de probabilidade, que por sua vez é descrita por seus parâmetros e respectivos campos de variação **(espaço paramétrico)**. Além, do campo de variação da própria variável aleatória que deve ser compatível com o suporte da distribuição atribuída ao fenômeno. Por exemplo, não é correto atribuir uma distribuição de Poisson para a altura (medida contínua) de trabalhadores, uma vez que o campo de variação da variável de interesse (resposta) não é compatível com o suporte da distribuição de probabilidade.

Considere o caso onde deseja-se fazer uma pesquisa a respeito da intenção de voto em um determinado candidato. Suponha que n eleitores obtidos aleatoriamente são questionados sobre a sua intenção em votar (1) ou não votar (0) em determinado candidato. Neste caso, tem-se como possíveis resultados do questionamento a resposta (1) ou (0). Deseja-se saber a probabilidade de um eleitor ao acaso manifestar a intenção de votar neste candidato. Dado a estrutura do experimento, pode-se supor que o modelo de Bernoulli seja adequado para esta situação. Este modelo tem como suporte o 0 e 1, compatível com o experimento, além disso, tem como seu parâmetro indexador p que representa a probabilidade de sucesso, ou seja, votar no candidato. Para completar a especificação este parâmetro

tem como seu **espaço paramétrico** o intervalo unitário. Com este conjunto de suposições e conhecimentos a respeito do modelo probabilístico, tem-se total condições de fazer **inferência** sobre o parâmetro p.

Neste texto será dada ênfase ao método de máxima verossimilhança, que fornece **estimadores** com propriedades desejáveis para os parâmetros desconhecidos de um modelo probabilístico. Este método é baseado na **função de verossimilhança**, e fornece uma abordagem integrada para a obtenção de estimativas pontuais e intervalares, além da construção de testes de hipóteses. Toda a metodologia será descrita através de exemplos abordando diversos aspectos teóricos, com ênfase na implementação computacional para estimação de parâmetros desconhecidos desde modelos mais simples até modelos mais estruturados.

Definição 1.1 (Função de Verossimilhança)**.** *Suponha que os dados $\underline{y}$ são uma realização de um vetor aleatório $\underline{Y}$ com função de probabilidade ou densidade probabilidade $f(\underline{Y},\underline{\theta})$. A **função de verossimilhança** (ou simplesmente verossimilhança) para $\underline{\theta}$ dado os valores observados $\underline{y}$ é a função $L(\underline{\theta},\underline{y})$.*

A função de verossimilhança é dada pela expressão da distribuição conjunta de todas as variáveis aleatórias envolvidas no modelo, porém vista como função dos parâmetros, uma vez que tendo os dados sido observados, são quantidades fixas. Para cada particular valor do parâmetro (escalar ou vetor), a verossimilhança é uma medida de compatibilidade, plausibilidade ou similaridade com a amostra observada.

A expressão da verossimilhança $L(\underline{\theta},\underline{y})$ pode ser mais cuidadosamente definida considerando a natureza das variáveis. Para modelos discretos não há ambiguidade e o valor da função de verossimilhança é a probabilidade do dado observado,

$$L(\theta) = P_\theta[Y = y].$$

Já para modelos contínuos a probabilidade de um particular conjunto de valores dos dados é nula. Entretanto, na prática medidas contínuas são tomadas com algum grau de precisão em um intervalo $(y_{iI} \leq y_i \leq y_{iS})$ e a verossimilhança para um conjunto de observações é:

$$L[\theta] = P_\theta[y_{1I} \leq y_1 \leq y_{1S}, \ldots, y_{1I} \leq y_1 \leq y_{1S}]. \tag{1.1}$$

Esta definição é geral e pode ser utilizada tanto para dados considerados pontuais como dados intervalares, como no caso de dados censurados.

Vamos supor agora uma situação mais simples e comum na qual todos os dados são medidos a um grau de precisão comum. Neste caso que cada dado é medido em um intervalo $(y_i - \delta/2 \leq Y_i \leq y_i + \delta/2)$. Supondo ainda, sem perda de generalidade, observações independentes a verossimi-

lhança é dada por:

$$\begin{aligned} L[\theta] &= \prod_{i=1}^{n} P_\theta[y_i - \delta/2 \leq Y_i \leq y_i + \delta/2] \\ &= \prod_{i=1}^{n} \int_{y_i-\delta/2}^{y_i+\delta/2} f(y_i, \underline{\theta}) d(y_i). \end{aligned}$$

Se o grau de precisão é alto (δ é pequeno) em relação a variabilidade dos dados a expressão se reduz a

$$L[\theta] \approx \left(\prod_{i=1}^{n} f(y_i, \underline{\theta}) \right) \delta^n,$$

e se δ não depende dos valores do parâmetros temos a verossimilhança como produto das densidades individuais,

$$L[\theta] \approx \prod_{i=1}^{n} f(y_i, \underline{\theta}), \tag{1.2}$$

e de forma mais geral para observações não independentes com a densidade multivariada:

$$L[\theta] \approx \prod_{i=1}^{n} f(\underline{y}, \underline{\theta}). \tag{1.3}$$

No caso onde os elementos de $\underline{y}$ são independentes a verossimilhança é simplesmente um produto das distribuições de cada variável Y_i individualmente, ou seja, $L(\underline{\theta}, \underline{y}) = \prod_{i=1}^{n} f(y_i, \underline{\theta})$. Neste caso, o procedimento de inferência pode ser bastante facilitado tanto analítica como computacionalmente. Porém, cabe ressaltar que isso não é uma exigência, e situações onde as amostras não são independentes são tratadas da mesma forma, escrevendo a verossimilhança de uma forma adequada, considerando a distribuição conjunta do vetor $\underline{Y}$.

O texto concentra-se exclusivamente no uso da função de verossimilhança como base para inferências estatística, sejam na obtenção de estimativas pontuais, intervalares ou testes de hipótese. Começamos revisando conceitos de estimação e vendo como se relacionam com a verossimilhança.

1.1 Estimação pontual

Seja $Y_1, Y_2, \ldots, Y_n$ variáveis aleatórias com função probabilidade para variáveis aleatórias discretas ou densidade de probabilidade para contínuas, e em ambos os casos denotada por $f(\underline{Y}, \underline{\theta})$, em que $\underline{\theta}$ é um vetor de parâmetros desconhecidos (um único elemento de $\underline{\theta}$ será denotado por θ), aos

quais se deseja estimar através de amostras $y_1, y_2, \ldots, y_n$ realizações das variáveis aleatórias $Y_1, Y_2, \ldots, Y_n$. Denota-se de forma simplificada, $Y_i \sim f(\underline{\theta})$ com $i = 1, \ldots, n$.

Definição 1.2 (Estatística). *Uma **estatística** é uma variável aleatória $T = t(\underline{Y})$, onde a função $t(\cdot)$ não depende de θ.*

Definição 1.3 (Estimador). *Uma estatística T é um estimador para θ se o valor realizado $t = t(\underline{y})$ é usado como uma estimativa para o valor de θ.*

Definição 1.4 (Distribuição amostral). *A distribuição de probabilidade de T é chamada de **distribuição amostral** do estimador $t(\underline{Y})$.*

Definição 1.5 (Viés). *O **viés** de um estimador T é a quantidade*

$$B(T) = E(T - \theta).$$

O estimador T é dito não viciado para θ se $B(T) = 0$, tal que $E(T) = \theta$. O estimador T é assintoticamente não viciado para θ se $E(T) \to \theta$ quando $n \to \infty$.

Definição 1.6 (Eficiência relativa). *A **eficiência relativa** entre dois estimadores T_1 e T_2 é a razão $er = \frac{V(T_1)}{V(T_2)}$.*

Definição 1.7 (Erro quadrático médio). *O **erro quadrático médio** de um estimador T é a quantidade*

$$EQM(T) = E((T - \theta)^2) = V(T) + B(T)^2.$$

Definição 1.8 (Consistência). *Um estimador T é **médio quadrático consistente** para θ se o $EQM(T) \to 0$ quando $n \to \infty$. O estimador T é **consistente em probabilidade** se $\forall \epsilon > 0$, $P(|T - \theta| > \epsilon) \to 0$, quando $n \to \infty$.*

Estas definições introduzem conceitos e propriedades básicas para uma estatística ser um estimador adequado para um determinado parâmetro. Fracamente falando, o desejo é obter um estimador que seja assintoticamente não-viciado, ou seja, conforme o tamanho da amostra aumenta ele se aproxima cada vez mais do verdadeiro valor do parâmetro. Além disso, é interessante que ele seja eficiente, ou seja, apresente a menor variância possível entre todos os estimadores de θ. Esta definição de eficiência, introduz o conceito de variância minima. Sendo assim, para saber se um estimador é eficiente é necessário conhecer um limite inferior para a variância de um estimador, uma vez que tal quantidade exista e seja passível de calcular, ao propor um estimador para θ, basta calcular a sua variância e comparar com a menor possível, se ele atingir este limite será eficiente. Além disso, tomando sua esperança pode-se concluir sobre o seu viés dependendo da situação em termos assintóticos. O Teorema 1.1, ajuda a responder sobre a

eficiência de um estimador qualquer. Mas antes precisamos de mais algumas definições.

Como dito, a verossimilhança é uma medida de compatibilidade da amostra observada com um particular vetor de parâmetros, desta forma é natural definir como estimador para o vetor de parâmetros $\underline{\theta}$, aquele particular vetor digamos, $\underline{\hat{\theta}}$, que tenha a maior compatibilidade com a amostra, ou em outras palavras o vetor que maximiza a função de verossimilhança ou compatibilidade. O particular valor assumido pela função de verossimilhança não é importante, o que interessa para **inferência** são os valores relativos de $L(\underline{\theta}, \underline{y})$ para diferentes conjuntos de $\underline{\theta}$.

Definição 1.9. *Seja $L(\underline{\theta}, \underline{y})$ a função de verossimilhança. O valor $\hat{\underline{\theta}} = \hat{\underline{\theta}}(\underline{y})$ é a estimativa de máxima verossimilhança para $\underline{\theta}$ se $L(\hat{\underline{\theta}}) \geq L(\underline{\theta})$, $\forall \underline{\theta}$.*

Definição 1.10. *Se $\hat{\underline{\theta}}(\underline{y})$ é a estimativa de máxima verossimilhança, então $\hat{\underline{\theta}}(\underline{Y})$ é o estimador de máxima verossimilhança.*

Nesta etapa é preciso ter cuidado com a notação. Veja que $\hat{\underline{\theta}}(\underline{y})$ é um vetor de escalares, enquanto que $\hat{\underline{\theta}}(\underline{Y})$ é um vetor de variáveis aleatórias. Daqui em diante usaremos apenas $\hat{\underline{\theta}}$, para ambos sendo que o contexto indicará o real sentido de $\hat{\underline{\theta}}$. A função de verossimilhança contêm toda a informação proveniente dos dados sobre o vetor de parâmetros $\underline{\theta}$. Apesar disso, a $L(\underline{\theta})$ é computacionalmente incoveniente, uma vez que esta função apresentará valores muito próximos de zero. Por razões meramente computacionais é mais comum usar a função de log-verossimilhança, definida por:

Definição 1.11 (Log-verossimilhança)**.** *Se $L(\underline{\theta})$ é a função de verossimilhança, então $l(\underline{\theta}) = \log L(\underline{\theta})$ é a função de log-verossimilhança.*

Segue do fato da função logaritmo ser monótona crescente que maximizar $L(\underline{\theta})$ e $l(\underline{\theta})$ levam ao mesmo ponto de máximo. Neste ponto estamos habilitados a enunciar um dos teoremas mais fortes da inferência estatística.

Teorema 1.1 (Limite inferior de Cramer Rao)**.** *Se T é um estimador não-viciado para θ e $l(\theta, \underline{Y})$ é duas vezes diferenciável com respeito a θ, então*

$$V(T) \geq \frac{1}{E(-l''(\theta, \underline{Y}))}.$$

Este teorema informa o limite inferior para a variância de um estimador T qualquer. O estimador de máxima verossimilhança apresenta propriedades ótimas e uma delas é a eficiência, ou seja, assintoticamente o EMV atinge o limite inferior de Cramer-Rao. Antes de discutirmos as propriedades dos estimadores de máxima verossimilhança, vamos apresentar uma

forma de introduzir a incerteza associada a estimativa de um parâmetro qualquer. Lembre-se que o estimador é um variável aleatória, a estimativa é uma realização desta variável aleatória. Sendo assim, quando reportamos apenas a estimativa pontual, estamos ignorando a incerteza associada a esta estimativa. Uma forma, tradicional de se medir e informar a incerteza associada é com a construção de intervalos de confiança.

1.2 Intervalos de confiança

Definição 1.12 (Intervalo de confiança)**.** *Um intervalo de verossimilhança para θ é um intervalo da forma $\theta : L(\theta) \geq rL(\hat{\theta})$ ou equivalentemente, $\theta : D(\theta) \geq c^*$, com $D(\theta) = 2(l(\hat{\theta}) - l(\theta))$ e $c^* = -2\log(r)$.*

Esta definição é bastante geral para o caso uni-paramétrico, para o caso multi-parâmetros os princípios se mantém e trocamos o intervalo de confiança por uma região de confiança, o que será abordado mais adiante. Nesta definição o valor de r precisa ser especificado entre 0 e 1, para intervalos não vazios, logo $c^* > 0$. Quanto maior o valor de c^* mais largo será o intervalo, algumas vezes o intervalo pode ser a união de sub-intervalos disjuntos. Apesar do valor de c^* ser a necessário para a construção dos intervalos ainda não temos condições de especificar um valor para ele.

Usando esta definição pode-se pensar em duas formas básicas de construção de intervalos de confiança. A primeira é considerar a quantidade $\frac{L(\theta)}{L(\hat{\theta})} \geq r$ que é a verossimilhança relativa, ou seja, compara cada valor de θ com o máximo. Nestas condições a verossimilhança relativa toma sempre valores entre 0 e 1 e o intervalo é a região do espaço paramétrico para qual os valores associados de verossimilhança sejam uma fração não menor que r do máximo valor. Por exemplo, definindo $r = 0.8$ estamos deixando que faça parte do intervalo de confiança valores que tenham até 80% de compatibilidade com a amostra, da mesma forma poderíamos definir $r = 0.20$ ou 0.5, dependendo de nosso critério. Royall (1997) propõe que este valor seja definido por analogias com resultados considerados aceitáveis em experimentos simples como lançamento de uma moeda. Uma forma equivalente é utilizar a função *deviance* definindo o intervalos pelos valores que satisfazem $D(\theta) = 2(l(\hat{\theta}) - l(\theta)) \geq -2\log(r)$. Esta é uma outra forma usar a verossimilhança relativa, agora em termos de diferença em log-verossimilhança. Neste caso a região de confiança pode ser definida como anteriormente ou valendo-se de propriedades frequentistas desta quantidade conforme veremos na sequência.

Em ambas abordagens surge o problema de que após definir o valor $c^* = -2log(r)$, é necessário encontrar as raízes da função de verossimilhança relativa ou da *deviance* que fornecem os limites do intervalo de con-

fiança para um c^* qualquer especificado. Encontrar as raízes da função comumente envolve métodos numéricos, uma vez que na maioria das situações práticas não é possível obter expressões fechadas para os limites do intervalo.

Dado esta restrição é comum fazer uma expansão em séries de Taylor para a $l(\theta)$ em torno de $\hat{\theta}$ de forma a facilitar a obtenção do intervalo de confiança.

$$D(\theta) = 2[l(\hat{\theta}) - l(\theta)] = 2[l(\hat{\theta}) - [l(\hat{\theta}) + (\theta - \hat{\theta})l'(\hat{\theta}) + \frac{1}{2}(\theta - \hat{\theta})^2 l''(\hat{\theta})]].$$

Como por definição de EMV $l'(\hat{\theta}) = 0$, eliminando termos a aproximação quadrática define a região

$$D(\theta) = -(\theta - \hat{\theta})^2 l''(\hat{\theta}) \geq c^*.$$

que define então intervalos de confiança da forma,

$$\tilde{\theta} = \hat{\theta} \pm \sqrt{\frac{c^*}{l''(\hat{\theta})}}.$$

Isto corresponde a fazer uma aproximação quadrática da função *deviance*, que torna o intervalo fácil de ser obtido. Estendendo para o caso de multiplos parâmetros, tem-se que uma região de confiança para $\underline{\theta}$ é dada pelo conjunto $\underline{\theta} \in \Theta : D(\underline{\theta} \geq c^*)$. Portanto, as duas formas de interpretar o intervalo de confiança discutidas no caso uniparamétrico podem ser estendidas para o caso multiparamétrico, sem problemas. Novamente a questão que surge é a definição de um valor para c^*. Pela abordagem frequentista é desejável que o intervalo tenha uma interpretação em termos de probabilidades ou frequência e isto é atingido através das propriedades assintóticas dos estimadores de máxima verossimilhança, que serão apresentadas na próxima Seção.

1.3 Propriedades do estimador

Apesar de definirmos a função de verossimilhança como uma quantidade fixa avaliada em $\underline{y}$, devemos lembrar que ela é baseada em apenas uma realização do vetor aleatório $\underline{Y}$, sendo assim, estudar o comportamento probabilístico dos estimadores de máxima verossimilhança é de fundamental importância para a construção de intervalos de confiança e testes de hipóteses. Para isto, vamos precisar de mais algumas definições.

Definição 1.13 (Função escore)**.** *Sendo $l(\underline{\theta})$ a função de log-verossimilhança, o vetor escore é definido por*

$$U(\underline{\theta}) = \left(\frac{\partial l(\underline{\theta})}{\partial \theta_1}, \ldots, \frac{\partial l(\underline{\theta})}{\partial \theta_d} \right)^\top,$$

é o vetor gradiente da função de log-verossimilhança.

Definimos as matrizes de informação *observada* e *esperada* (matriz de informação de Fisher).

Definição 1.14 (Matriz de informação Observada)**.** *Sendo $l(\underline{\theta})$ a função de log-verossimilhança, a matriz de informação Observada é definida por*

$$I_O(\underline{\theta}) = \begin{bmatrix} -\frac{\partial^2 l(\underline{\theta})}{\partial \theta_1^2} & \cdots & \cdots & -\frac{\partial^2 l(\underline{\theta})}{\partial \theta_1 \partial \theta_d} \\ \vdots & \ddots & -\frac{\partial^2 l(\underline{\theta})}{\partial \theta_i \partial \theta_j} & \vdots \\ \vdots & -\frac{\partial^2 l(\underline{\theta})}{\partial \theta_j \partial \theta_i} & \ddots & \vdots \\ -\frac{\partial^2 l(\underline{\theta})}{\partial \theta_d \partial \theta_1} & \cdots & \cdots & -\frac{\partial^2 l(\underline{\theta})}{\partial \theta_d^2} \end{bmatrix}.$$

Definição 1.15 (Matriz de informação Esperada)**.** *Sendo $l(\underline{\theta})$ a função de log-verossimilhança, a matriz de informação Esperada é definida por*

$$I_E(\underline{\theta}) = \begin{bmatrix} E\left[-\frac{\partial^2 l(\underline{\theta})}{\partial \theta_1^2}\right] & \cdots & \cdots & E\left[-\frac{\partial^2 l(\underline{\theta})}{\partial \theta_1 \partial \theta_d}\right] \\ \vdots & \ddots & E\left[-\frac{\partial^2 l(\underline{\theta})}{\partial \theta_i \partial \theta_j}\right] & \vdots \\ \vdots & E\left[-\frac{\partial^2 l(\underline{\theta})}{\partial \theta_j \partial \theta_i}\right] & \ddots & \vdots \\ E\left[-\frac{\partial^2 l(\underline{\theta})}{\partial \theta_d \partial \theta_1}\right] & \cdots & \cdots & E\left[-\frac{\partial^2 l(\underline{\theta})}{\partial \theta_d^2}\right] \end{bmatrix}.$$

Dois importantes resultados da função *escore* e da matriz de informação observada é que $E[U(\underline{\theta})] = 0$ e $V[U(\underline{\theta})] = E[I_O(\underline{\theta})] = I_E[\underline{\theta}]$.

Note que a variância do vetor $U(\underline{\theta})$ é a matriz com entradas

$$\begin{bmatrix} Cov(U_1, U_1) & \cdots & \cdots & Cov(U_1, U_d) \\ \vdots & \ddots & Cov(U_i, U_j) & \vdots \\ \vdots & Cov(U_j, U_i) & \ddots & \vdots \\ Cov(U_d, U_1) & \cdots & \cdots & Cov(U_d, U_d) \end{bmatrix}.$$

onde $Cov(U_i, U_i) = V(U_i)$. Uma propriedade importante de $I_O(\hat{\underline{\theta}})$ e $I_E(\hat{\underline{\theta}})$ é que elas são matrizes definida positiva, as quais mensuram a curvatura

observada/esperada na superfície de log-verossimilhança. Com estas definições, pode-se escrever a função *deviance* aproximada em termos multiparâmetros da seguinte forma:

$$D(\underline{\theta}) \approx (\underline{\theta} - \underline{\hat{\theta}})^\top I_O(\underline{\hat{\theta}})(\underline{\theta} - \underline{\hat{\theta}}).$$

Assim $D(\underline{\theta})$ será positiva desde que $I_O(\underline{\hat{\theta}})$ seja uma matriz positiva definida. Uma vez definida todas as quantidades envolvidas na situação, estamos aptos a enunciar Teorema a seguir.

Teorema 1.2 (Distribuição assintótica do EMV)**.** *Para um problema de estimação regular, no limite com $n \to \infty$, se $\underline{\theta}$ é o verdadeiro vetor de parâmetros, então*

$$\underline{\hat{\theta}} \sim NM_d(\underline{\theta}, I_E(\underline{\theta})^{-1}),$$

ou seja, a distribuição assintótica de $\underline{\hat{\theta}}$ é uma normal multivariada com matriz de variância/covariância dada pela inversa da matriz de informação esperada.

Corolário - Qualquer termo assintoticamente equivalente a $I_E(\underline{\theta})$ pode ser usado no Teorema 1.2. Assim,

$$\underline{\hat{\theta}} \sim NM_d(\underline{\theta}, I_E(\underline{\hat{\theta}})^{-1})$$

$$\underline{\hat{\theta}} \sim NM_d(\underline{\theta}, I_O(\underline{\theta})^{-1})$$

$$\underline{\hat{\theta}} \sim NM_d(\underline{\theta}, I_O(\underline{\hat{\theta}})^{-1}).$$

Teorema 1.3 (Distribuição assintótica da deviance)**.** *Para um problema regular de estimação, no limite com $n \to \infty$, se $\underline{\theta}$ é o verdadeiro valor do parâmetro, então*

$$D(\underline{\theta}) = 2[l(\underline{\hat{\theta}}) - l(\underline{\theta})] \sim \chi^2_d$$

ou seja, a função deviance segue uma distribuição Qui-Quadrado com d graus de liberdade, onde d é a dimensão do vetor $\underline{\theta}$.

De acordo com os teoremas apresentados, podemos chegar a algumas das principais propriedades dos estimadores de máxima verossimilhança:

- O estimador de máxima veossimilhança $\underline{\hat{\theta}}$ de $\underline{\theta}$ é assintóticamente não-viciado, isto é, $E(\underline{\hat{\theta}}) \to \underline{\theta}$.
- Assintoticamente $V(\underline{\hat{\theta}}) \to I_E(\underline{\theta})^{-1}$, o qual por uma versão multivariada do limite de Cramér-Rao é o melhor possível, mostrando que o EMV é eficiente para o vetor $\underline{\theta}$.
- Denote $J = I_E(\underline{\theta})^{-1}$, então $V(\underline{\hat{\theta}}) = J$, sendo que, J é uma matriz simétrica e definida positiva, com elementos $J_{i,j} = Cov(\underline{\hat{\theta}}_i, \underline{\hat{\theta}}_j)$ então $J_{i,i}$ é a variância de $\underline{\hat{\theta}}_i$. Denota-se $J_{i,i}^{\frac{1}{2}}$ de desvio padrão de $\underline{\hat{\theta}}_i$.

- Podemos construir intervalos de $100(1-\alpha)\%$ de confiança para θ_i na forma $\hat{\theta}_i \pm z_{\frac{\alpha}{2}} J^{\frac{1}{2}}_{i,i}$. Intervalos desta forma serão denominados, intervalos de Wald ou baseados em aproximação quadrática da verossimilhança.

- Para regiões de confiança baseados na *deviance* considera-se $[\underline{\theta} \in \Theta : D(\underline{\theta}) \leq c^*]$, para algum valor c^* a ser especificado. Pode-se escolher c^* baseado em justificativas assintóticas de que $D(\underline{\theta}) \sim \chi^2_d$ é uma escolha razoável para $c^* = c_\alpha$ com $P(\chi^2_d \geq c_\alpha) = \alpha$, por exemplo se $\alpha = 0.05$, então $c_\alpha = 3.84$. Isto gera uma região de $100(1-\alpha)\%$ de confiança. Estes intervalos serão denominados de intervalos *deviance*.

De acordo com as propriedades apresentadas tem-se duas formas básicas de construir intervalos de confiança. A primeira mais simples é baseada na aproximação quadrática da log-verossimilhança e a segunda olhando diretamente para a função *deviance*. A segunda opção é bastante complicada computacionalmente, uma vez que ela gera uma equação não linear que precisa ser resolvida numericamente, o que pode ser custoso. A primeira opção é bastante direta, uma vez obtida a matriz de segundas derivadas basta invertê-la e tirar a raiz dos termos da diagonal para se obter o intervalo de confiança para cada parâmetro, marginalmente. Esta abordagem apesar de muito fácil apresenta restrições, por exemplo, não respeita restrições naturais do espaço paramétrico como em parâmetros de variância e correlação pode resultar em limites irreais. Por aproximar sempre por uma forma quadrática os intervalos serão sempre simétricos, o que normalmente não funciona bem para parâmetros de variância e correlação. Em modelos com efeitos aleatórios um interesse natural está em parâmetros de variância, precisão e correlação testar hipóteses sobre a existência de tais efeitos normalmente é feito olhando para as estimativas de variâncias que indexam o modelo, logo esta abordagem é restrita nesta classe mais geral de modelos estatísticos.

É importante deixar claro que a segunda opção resulta em uma região (conjunta) uma elipse, enquanto que pela aproximação é possível obter um intervalo marginal para cada parâmetro, porém baseado em uma aproximação quadrática da superfície de log-verossimilhança. Esta opção é o intuito mais comum, porém não pode ser obtida diretamente apenas com o Teorema 1.3. Por exemplo, suponha que tem-se interesse em um determinado componente do vetor de parâmetros, digamos θ_i baseado na aproximação quadrática podemos facilmente construir um intervalo de confiança, tendo como $\hat{\theta}_L$ e $\hat{\theta}_U$ o seu limite inferior e superior respectivamente. Pelo Teorema 1.3 para o caso em que a dimensão de $\underline{\theta}$ é maior que um, não temos um intervalo desta forma, temos uma elipse o que apesar de mais informativo tem menos apelo prático e apresenta dificuldades de interpretação. Uma

forma intuitiva de obter um intervalo da forma $\hat{\theta}_L$ e $\hat{\theta}_U$ é fixar o restante do vetor de parâmetros nas suas estimativas de máxima verossimilhança e obter os limites em uma direção de cada vez. Esta abordagem tem uma clara restrição que é não levar em consideração a não ortogonalidade nem a incerteza associada ao restante do vetor de parâmetros para a construção do intervalo.

Dada a estrutura do problema temos um método simples via aproximação quadrática, porém não funciona bem quando a superfície de log-verossimilhança é assimétrica. Por outro lado, o método da *deviance* não apresenta esta restrição mais só fornece regiões de confiança, e não limites do tipo $\hat{\theta}_L$ e $\hat{\theta}_U$ para cada parâmetro. Duas abordagens básicas para este problema podem ser consideradas: a primeira é fazer uma reparametrização do modelo, nos parâmetros que apresentam forte assimetria ou são restritos, para torná-los irrestritos e aproximadamente simétricos, obter a variância baseada na aproximação quadrática nesta reparametrização e depois converter para a escala original. Quando este procedimento é satisfatório o custo computacional é pequeno.

Para formalizar esta situação, considere o problema de obter a estimativa pontual e intervalar para um parâmetro de interesse $\phi = g(\underline{\theta})$, onde $g(\cdot)$ é uma função. Desde que $L(\phi) = L(g(\underline{\theta}))$, a função de verossimilhança para ϕ é obtida da função de verossimilhança de $\underline{\theta}$ por uma transformação de escala. Consequentemente, $\hat{\phi} = g(\hat{\underline{\theta}})$, quando o intervalo de confiança digamos $\hat{\theta}_L$ e $\hat{\theta}_U$ for obtido diretamente pela função de verossimilhança, log-verossimilhança ou *deviance*, o intervalo para ϕ pode ser obtido simplesmente transformando os limites obtidos para θ, no caso unidimensional. Esta propriedade é conhecida como invariância do estimador de máxima verossimilhança. Porém, quando o intervalo for obtido pela aproximação quadrática isso não é válido e um Teorema adicional é necessário para esta transformação.

Teorema 1.4. *Considere obter um intervalo de confiança para $\phi = g(\underline{\theta})$ por invariância temos que $\hat{\phi} = g(\theta)$ e a variância de $\hat{\phi}$ é dada por*

$$V(\hat{\phi}) = V(g(\underline{\theta})) = \nabla g(\underline{\theta})^\top I_E(\underline{\theta})^{-1} \nabla g(\underline{\theta})$$

com

$$\nabla g(\underline{\theta}) = \left(\frac{\partial g(\underline{\theta})}{\partial \theta_1}, \ldots, \frac{\partial g(\underline{\theta})}{\partial \theta_d} \right)^\top$$

A partir do Teorema 1.4 é imediato o seguinte resultado.

Teorema 1.5. *Para um problema de estimação regular se $\phi = g(\underline{\theta})$ são os verdadeiros valores dos parâmetros, então quando $n \to \infty$ tem-se que*

$$\underline{\hat{\phi}} \sim NM_d(\phi, \nabla g(\underline{\theta})^\top I_E(\underline{\theta})^{-1} \nabla g(\underline{\theta}))$$

Pelo Teorema 1.5, podemos construir intervalos de confiança da mesma forma anterior, porém usando a nova matriz de variância e covariância ponderada pelo gradiente da função $g(\cdot)$, e assim passar de uma reparametrização para outra torna-se uma tarefa trivial. Apesar deste procedimento ser bastante útil, nem sempre é fácil encontrar uma transformação $g(\cdot)$ que torne a log-verossimilhança simétrica. A forma mais efetiva de construir intervalos de confiança para parâmetros de difícil estimação é o intervalo baseado em perfil de verossimilhança.

Seja $\underline{\theta} = (\underline{\phi}^\top, \underline{\lambda}^\top)^\top$, o vetor de parâmetros particionado nos vetores $\underline{\phi}$ e $\underline{\lambda}$, vamos chamar a primeira componente de interesse e a segunda de incômodo, no sentido que desejamos intervalos ou regiões de confiança para $\underline{\phi}$, que pode ser apenas um escalar. Seja $L(\underline{\phi}, \underline{\lambda})$ a verossimilhança para $\underline{\phi}$ e $\underline{\lambda}$. Denota-se $\hat{\underline{\lambda}}_\phi$ a estimativa de máxima verossimilhança de $\underline{\lambda}$ para dado valor de $\underline{\phi}$.

Definição 1.16 (Verossimilhança perfilhada)**.** *A verossimilhança perfilhada de $\underline{\phi}$ é definida por*

$$\tilde{L}(\underline{\phi}) = L(\underline{\phi}, \hat{\underline{\lambda}}_\phi)$$

A forma apresentada na definição 1.16 sugere um procedimento de maximização em duas etapas. A primeira consiste em obter $\hat{\underline{\lambda}}_\phi$ que maximiza $l(\underline{\phi}, \underline{\lambda}) = \log L(\underline{\phi}, \underline{\lambda})$ com respeito a $\underline{\lambda}$ supondo $\underline{\phi}$ fixo. A seguir maximiza-se $\tilde{l}(\underline{\phi})$. Assim, uma região ou intervalo de confiança para $\underline{\phi}$ pode ser obtida usando que

$$\tilde{D}(\underline{\phi}) = 2[\tilde{l}(\hat{\underline{\phi}}) - \tilde{l}(\underline{\phi})] \sim \chi^2_d$$

onde d é a dimensão de $\underline{\phi}$. Note que esta forma de construção não impõe nenhum tipo de aproximação, ela pode resultar em intervalos muito assimétricos. Porém, é altamente cara computacionalmente, uma vez que precisamos resolver numericamente uma equação não-linear que para cada avaliação necessita de um algoritmo numérico de maximização.

1.4 Testes de hipóteses

Definição 1.17. *Chamamos de hipótese estatística qualquer afirmação acerca da distribuição de probabilidade de uma ou mais variáveis aleatórias.*

Definição 1.18. *Chamamos de teste de uma hipótese estatística a função de decisão $\chi \to \{a_0, a_1\}$, em que a_0 corresponde à ação de considerar a hipótese H_0, como verdadeira e a_1 corresponde à ação de considerar a hipótese H_1 como verdadeira.*

Na definição acima, χ denota o espaço amostral associado à amostra $Y_1, Y_2, \ldots, Y_n$. A função de decisão d divide o espaço amostral χ em dois

conjuntos,

$$A_0 = \{(y_1, \ldots, y_n \in \chi; d(y_1, \ldots, y_n) = a_0\}$$

e

$$A_1 = \{(y_1, \ldots, y_n \in \chi; d(y_1, \ldots, y_n) = a_1\}$$

onde $A_0 \cup A_1 = \chi$ e $A_0 \cap A_1 = \emptyset$. Como em A_0 temos os pontos amostrais que levam à aceitação de H_0, vamos chamar de A_0 de região de aceitação e, por analogia, A_1 de região de rejeição de H_0, também chamada de região crítica.

Um teste de hipótese pode resultar em um de dois tipos de erros. Tradicionalmente, esses dois tipos de erros recebem os nomes de erro Tipo I (α) e erro tipo II (β). O erro tipo I ocorre quando rejeitamos H_0 esta é verdadeira. O erro tipo II ocorre quando aceitamos H_0 e esta é falsa. Em termos de probabilidade temos,

$\alpha = P(Y \in A_1|\theta_0)$ e $\beta = P(Y \in A_0|\theta_1)$.

Definição 1.19. *O poder do teste com região critica A_1 para testar $H_0 : \theta = \theta_0$ contra $H_1 : \theta = \theta_1$ é dado por*

$$\pi(\theta_1) = P(Y \in A_1|\theta_1)$$

Note que $\pi(\theta_1) = 1 - \beta$, e β é a probabilidade do erro tipo II.

1.4.1 Teste da razão de verossimilhança

Definição 1.20. *A estatística do teste da razão de verossimilhança para testar $H_0 : \theta \in \Theta_0$ versus $H_1 : \theta \in \Theta_0^c$ é*

$$\lambda(\underline{y}) = \frac{sup_{\Theta_0} L(\theta|\underline{y})}{sup_{\Theta} L(\theta|\underline{y})}$$

O teste da razão de verossimilhança (TRV) é qualquer teste que tenha uma região de rejeição da forma $\underline{y} : \lambda(\underline{y}) \leq c$ onde c é qualquer número que satisfaça $0 \leq c \leq 1$.

Para testar $H_0 : \theta = \theta_0$ versus $H_1 : \theta \neq \theta_0$, suponha $Y_1, \ldots, Y_n$ sejam iid $f(\underline{y}|\theta)$, $\hat{\theta}$ seja o EMV de θ, e $f(\underline{y}|\theta)$ satisfaça as condições de regularidade. Desse modo, de acordo com H_0, pelo Teorema 1.3 à medida que $n \to \infty$

$$-2 \log \lambda(\underline{y}) \to \chi_1^2.$$

1.4.2 Teste de Wald

Suponha que deseja-se testar a hipótese bilateral $H_0 : \theta = \theta_0$ versus $H_1 : \theta \neq \theta_0$. Um teste aproximado poderia ter como base a estatística $z_n = (W_n - \theta_0)/S_n$ e rejeitaria H_0 se, e somente se, $z_n < -z_{\alpha/2}$. Se H_0 for verdadeira, então $\theta = \theta_0$ e Z_n converge em distribuição para $Z \sim N(0,1)$. Portanto, a probabilidade do Erro Tipo I, $P_{\theta_0}(Z_n < -z_{\alpha/2} \text{ ou } Z_n > z_{\alpha/2}) \to P(Z < -z_{\alpha/2} \text{ ou } Z > z_{\alpha/2}) = \alpha$, este é, assintoticamente, um teste de tamanho α. Em geral, um teste de Wald é um teste com base em uma estatística da forma,

$$Z_n = \frac{W_n - \theta_0}{S_n}$$

onde θ_0 é um valor hipotético do parâmetro θ, W_n é um estimador de θ e S_n é um erro padrão de W_n, uma estimativa do desvio padrão de W_n. Se W_n for o EMV para θ, então, $\sqrt{I_O(\hat{\theta})}$ é um erro padrão razoável para W_n.

1.4.3 Teste escore

Definição 1.21. *A estatística de escore é definida como*

$$U(\theta) = \frac{\partial}{\partial\theta} l(\theta|\underline{Y})$$

Sabemos que para todo θ, $E_\theta(U(\theta)) = 0$. Em particular, se estivermos testando $H_0 : \theta = \theta_0$ e se H_0 for verdadeira, então $U(\theta)$ tem média 0. Além disso,

$$V_\theta(U(\theta)) = -E_\theta\left(\frac{\partial^2}{\partial\theta^2} l(\theta|\underline{Y})\right) = I_E(\theta)$$

o número de informações é a variância da estatística escore. A estatística de teste para o teste de escore é

$$Z_S = U(\theta_0)/\sqrt{I_E(\theta_0)}.$$

Se H_0 for verdadeira, Z_S tem média 0 e variância 1.

1.5 Exemplo 1 - Estimação pontual

Neste exemplo consideramos um problema para no qual o estimador de máxima verossimilhança pode ser obtido analiticamente e ilustramos as propriedades básicas do estimador. Começamos mostrando quatro representações alternativas da verossimilhança.

Seja $Y_i \sim P(\lambda)$ com $i = 1, \ldots, n$, variáveis aleatórias independentes e denote $\overline{Y} = \sum_{i=1}^{n} Y_i/n$. A função de verossimilhança é o produto das n distribuições de Poisson com parâmetro λ comum a todas. Então a função de verossimilhança para o modelo é dada pela expressão a seguir, notando-se que, obtida uma determinada amostra, o termo no denominador é uma constante.

$$L(\lambda) = \prod_{i=1}^{n} \frac{\exp\{-\lambda\}\lambda^{Y_i}}{Y_i!} = \frac{\exp\{-n\lambda\}\lambda^{\sum_{i=1}^{n} Y_i}}{\prod_{i=1}^{n} Y_i!}.$$

Um representação alternativa é a função de verossimilhança relativa. Sendo, $\hat{\lambda}$ o EMV para λ a função de verossimilhança relativa é dada por $LR(\lambda) = \frac{L(\lambda)}{L(\hat{\lambda})}$ que para esse exemplo tem a expressão a seguir. Os valores assumidos por esta função estão sempre no intervalo unitário o que facilita a construção e visualização de gráficos. Note-se ainda que nesta representação o termo contante do denominador é cancelado.

$$LR(\lambda) = \exp\{-n(\lambda - \hat{\lambda})\}(\lambda/\hat{\lambda})^{n\overline{Y}}.$$

Outra possibilidade é usar a função de log-verossimilhança $l(\lambda) = \log L(\lambda)$ que normalmente é preferida para se trabalhar analítica e computacionalmente do que a $L(\lambda)$. Para o exemplo a expressão é como se segue com o último termo constante para uma determinada amostra.

$$l(\lambda) = -n\lambda + n\overline{Y}\log(\lambda) - \sum_{i=1}^{n} \log(Y_i!).$$

Por fim, podemos ainda utilizar a função *deviance* dada por, $D(\lambda) = 2\{l(\hat{\lambda}) - l(\lambda)\}$, que é comumente reportada por algoritmos e utilizada na obtenção de intervalos de confiança e testes de hipótese, devida a suas propriedades assintóticas. Assim como na verossimilhança relativa, a sua expressão elimina o termo constante ficando na forma:

$$D(\lambda) = 2n\{(\lambda - \hat{\lambda}) - \overline{Y}\log(\lambda/\hat{\lambda})\}.$$

A Figura 1.1, apresenta os gráficos dessas quatro formas de visualização da função de verossimilhança para os dados a seguir.

```
> set.seed(20)
> (y <- rpois(20,lambda=10))
 [1] 13  8 15  5  8 12 12  9  6  9  9  8 14  5  9  7  9 11 10  9
```

Apesar das quatro formas serem equivalentes a forma mais conveniente para encontrar o estimador de máxima verossimilhança é a log-verossimilhança. Cálculos analíticos com a função de verossimilhança podem ser mais trabalhosos e sua computação mais sensível a valores que

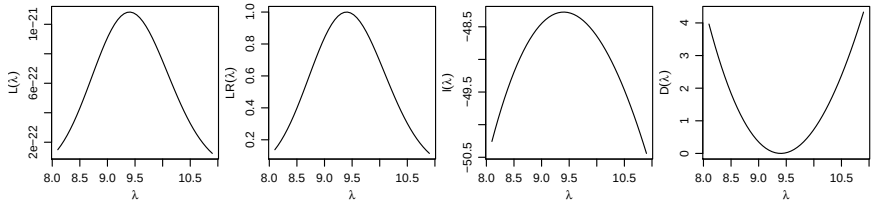

Figura 1.1: Diferentes formas de visualizar a função de verossimilhança - Distribuição Poisson.

podem gerar problemas numéricos, por exemplo excedendo a capacidade de representação de números. As funções de verossimilhança relativa e *deviance* exigem em sua definição o valor da estimativa que, para a *deviance* corresponde à raiz, ou seja, onde a função toca o eixo x.

Seguindo com o exemplo vamos encontrar o estimador de máxima verossimilhança para λ, analiticamente maximizando a função de log-verossimilhança. Partindo da verossimilhança temos,

$$\begin{aligned}
L(\lambda) &= \prod_{i=1}^{n} \frac{\exp\{-\lambda\}\lambda^{Y_i}}{Y_i!} \\
l(\lambda) &= -n\lambda + \log(\lambda)\sum_{i=1}^{n} Y_i - \sum_{i=1}^{n} \log Y_i! \\
U(\lambda) &= -n + \frac{1}{\lambda}\sum_{i=1}^{n} Y_i \\
\hat{\lambda} &= \frac{\sum_{i=1}^{n} Y_i}{n} \\
I_O(\lambda) &= \frac{\sum_{i=1}^{n} Y_i}{\lambda^2} \; ; \; I_O(\hat{\lambda}) = \frac{n^2}{\sum_{i=1}^{n} Y_i} \; ; \; I_E(\lambda) = \frac{n}{\lambda} \\
V(\hat{\lambda}) &= I_E(\lambda)^{-1} \approx I_O(\lambda)^{-1} \approx I_O(\hat{\lambda})^{-1}
\end{aligned}$$

Como o estimador de máxima verossimilhança é uma função de uma variável aleatória ele também é uma variável aleatória. Conforme as propriedades apresentadas o EMV é assintoticamente não viciado e sua distribuição amostral é assintoticamente gaussiana. Para exemplificar estas propriedades vamos fazer um pequeno estudo de simulação, para verificar como se comporta o viés e a distribuição do EMV conforme aumenta o tamanho da amostra.

Para isto, simulamos 1.000 conjuntos de dados de acordo com o modelo Poisson com $\lambda = 3.5$ e $\lambda = 10$. Vamos retirar amostras de tamanho $5, 50$ e

100, em cada amostra calcular o EMV. A Figura 1.2 apresenta os resultados deste estudo de simulação. Pelas propriedades do EMV temos que $\hat{\lambda} \sim N(\lambda, \frac{\lambda^2}{n\bar{y}})$. Na Figura 1.2 sobrepomos o histograma das estimativas obtidas nas simulações com a gráfico da distribuição assintótica (normal).

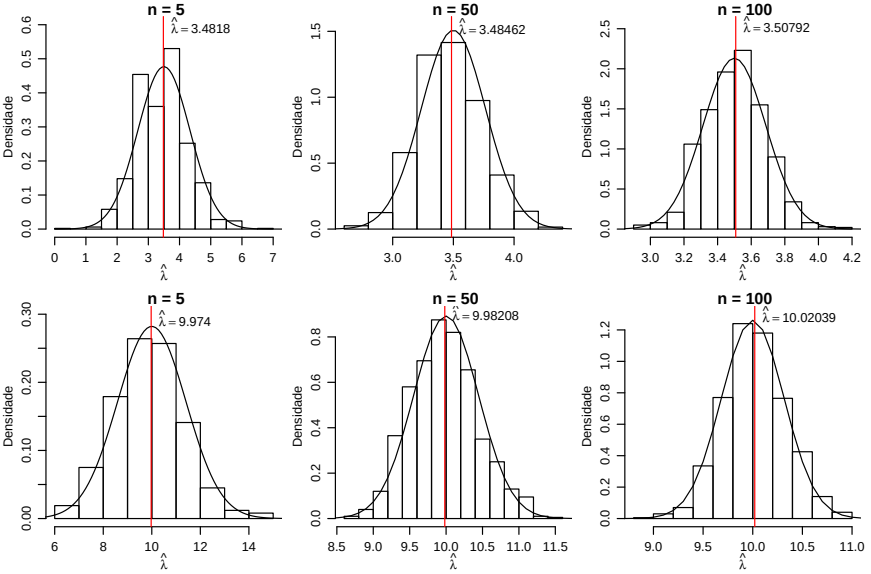

Figura 1.2: Distribuição assintótica estimador de máxima verossimilhança- Distribuição Poisson.

Como é possível visualizar na Figura 1.2 a distribuição empírica apresenta um comportamento muito próximo da distribuição teórica, mesmo para valores baixos de λ e amostras pequenas $n = 5$ e $n = 50$, o viés vai diminuindo conforme a amostra aumenta. É também evidente que com uma amostra maior a variância do EMV vai diminuindo, até no caso limite quando $n \to \infty$ atinge o 0 mostrando a consistência do EMV. É interessante observar que mesmo com uma amostra pequena, os resultados válidos assintoticamente já apresentam resultados excelentes. É claro que este é um exemplo simples, porém como veremos mesmo em modelos mais complexos, como nos modelos lineares generalizados (MLG) estes resultados permanecem igualmente bons.

1.6 Exemplo 2 - Intervalos de confiança

Como vimos na Seção 1.2 há três formas básicas para obtenção dos intervalos baseados na função de verossimilhança. A intuitivamente mais

simples é baseada na verossimilhança relativa, isto é, o intervalo é definido pelo conjunto de valores do parâmetro que tenham uma verossimilhança não inferior a um certo percentual da verossimilhança máxima. O intervalo obtido desta forma não tem interpretações probabilísticas. A segunda forma baseia-se na função *deviance*. Considerando a sua distribuição assintótica, define-se um ponto de corte nesta função e o intervalo é dado pelos conjunto dos valores para o parâmetro que possuem *deviance* inferior a esse valor. A terceira utiliza uma aproximação quadrática da superfície de log-verossimilhança. As últimas duas opções associam uma interpretação frequentista ao intervalo de confiança. Desta forma, os objetivos deste exemplo são: mostrar como obter os intervalos de confiança por cada uma das três abordagens, explorar as relações existentes e discutir algumas possíveis dificuldades computacionais que podem ocorrer quando trabalhamos diretamente com a função *deviance*.

Considere o caso onde amostras independentes de uma variável aleatória Y_i com $i = 1, 2, \ldots, n$ cada uma com distribuição exponencial de parâmetro θ esteja disponível, vamos denotar isto por, $Y_i \sim \text{Exp}(\theta)$. O objetivo é estimar o parâmetro θ e um intervalo de confiança, digamos $\hat{\theta}_L$ e $\hat{\theta}_U$ que, sob o enfoque frequentista, tenha probabilidade $1 - \alpha$ de conter θ. Note que a estrutura probabilística está em $\hat{\theta}_L$ e $\hat{\theta}_U$.

O primeiro passo é sempre escrever a função de verossimilhança,

$$L(\theta) = \prod_{i=1}^{n} \theta \exp\{-\theta y_i\} = \theta^n \exp\{-\theta \sum_{i=1}^{n} y_i\} = \theta^n \exp\{-n\overline{y}\theta\}.$$

Consequentemente, a função de log-verossimilhança é

$$l(\theta) = n \log \theta - \theta \sum_{i=1}^{n} y_i = n \log \theta - \theta n \overline{y}.$$

Derivando a log-verossimilhança em θ chegamos a função escore

$$U(\theta) = n\theta^{-1} - n\overline{y}.$$

Para encontrar a estimativa de máxima verossimilhança basta igualar a função escore a 0 e isolar o θ isto resulta

$$\hat{\theta} = \frac{1}{\overline{y}}.$$

A segunda derivada da função de log-verossimilhança tem um papel fundamental na construção de intervalos de confiança por aproximação quadrática da função de verossimilhança, uma vez que ela mede a curvatura da função no entorno do ponto de máximo. Em termos estatísticos,

trabalhamos com a informação observada ou esperada, uma vez que esta quantidade descreve o comportamento assintótico dos estimadores de máxima verossimilhança, já que sua inversa fornece a variância do estimador. Neste caso a função de log-verossimilhança é côncava e polinomial de ordem infinita. As informações esperada e observada são iguais dependem do valor do parâmetro e na prática é necessário usar a informação estimada pela amostra.

$$I_O(\theta) = I_E(\theta) = n\theta^{-2} \quad \text{e} \quad I_O(\hat{\theta}) = n\hat{\theta}^{-2}.$$

Para encontrar o intervalo de confiança vamos começar pela função *deviance*, lembrando que de acordo com o Teorema 1.3, a *deviance* segue uma distribuição χ^2 com d graus de liberdade, no exemplo $d = 1$. Desta forma o valor de c^* é obtido definindo algum quantil $q_{1-\alpha}$ da distribuição quiquadrado com 1 grau de liberdade e $1 - \alpha$ de confiança. Como exemplo, fixando $\alpha = 0.05$ toma-se o valor do quantil $c^* = 3.84$. Com isto, temos os elementos necessários para construir o intervalo de confiança baseado na função *deviance* que é dado pelo conjunto de valores que obedecem:

$$\begin{aligned} D(\theta) &= 2[l(\hat{\theta}) - l(\theta)] \\ &= 2[n\log\hat{\theta} - \hat{\theta}n\overline{y} - [n\log\theta - \theta n\overline{y}]] \\ &= 2n[\log\left(\hat{\theta}/\theta\right) + \overline{y}(\theta - \hat{\theta})] \leq c^* \end{aligned} \tag{1.4}$$

Os limites $(\hat{\theta}_L, \hat{\theta}_U)$ são encontrados resolvendo a equação em 1.4, mas mesmo em uma situação simples como a deste exemplo a obtenção das raízes requer a solução de um sistema não-linear que não tem solução analítica. Desta forma, algum método numérico é necessário para encontrar os limites do intervalo.

Uma outra opção para encontrar o intervalo de confiança é fazer uma aproximação quadrática utilizando séries de Taylor para a função $l(\theta)$ em torno do ponto $\hat{\theta}$ e usar esta aproximação no lugar da função de log-verossimilhança original para obter o intervalo de confiança. Neste caso, os limites do intervalo são as raízes de um polinômio de segundo grau e os intervalos são da forma

$$\hat{\theta} \pm z_{\frac{\alpha}{2}}\sqrt{V(\hat{\theta})},$$

o que corresponde a usar o resultado do Teorema 1.2.

Nesse exemplo com a distribuição exponencial, temos que a $V(\hat{\theta}) = I_O(\hat{\theta})^{-1} = \hat{\theta}^2/n$, logo o intervalo de confiança é dado por

$$\hat{\theta}_L = \hat{\theta} - z_{\frac{\alpha}{2}}\hat{\theta}/\sqrt{n} \quad \text{e} \quad \hat{\theta}_U = \hat{\theta} + z_{\frac{\alpha}{2}}\hat{\theta}/\sqrt{n}.$$

Neste caso a construção do intervalo de confiança fica bastante facilitada. Conhecendo o valor de $z_{\frac{\alpha}{2}}$ o intervalo se resume a uma conta simples,

ou seja, não é necessário utilizar métodos numéricos para obtenção dos limites do intervalo. Como a distribuição amostral de $\hat{\theta}$ é assintoticamente gaussiana podemos escolher o valor de $z_{\frac{\alpha}{2}}$ como o quantil da distribuição normal, com $1-\alpha$ % de confiança, escolhendo $\alpha = 0.05$ % , temos aproximadamente que $z_{\frac{\alpha}{2}} = 1.96$. No caso de $d = 1$ o quantil da distribuição $\chi^2_{(1)}$ é o quadrado de um score z da distribuição normal e o método também corresponde a obter o intervalo como na função *deviance* porém utilizando a sua aproximação quadrática dada por:

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

Desta forma as raízes que definem os limites dos intervalos equivalentes aos anteriores:

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

A última opção que vamos considerar é a construção do intervalo definindo um valor limite para a verossimilhança relativa. Um intervalo baseado em verossimilhança relativa é da forma $LR(\theta) = \frac{L(\theta)}{L(\hat{\theta})} \geq r$, onde r representa um percentual do máximo da função de verossimilhança, para o qual admite-se ainda obter uma estimativa aceitável de θ. Por exemplo, definindo $r = 0.25$, o intervalo é definido pelo valores que apresentam uma verosimilhança que seja ao menos 25% da máxima para o conjunto de dados. Para escolher um valor de r comparável com o escolhido para a *deviance*, note o seguinte:

$$\begin{aligned}
\frac{L(\theta)}{L(\hat{\theta})} &\geq r \\
\log\left[\frac{L(\theta)}{L(\hat{\theta})}\right] &\geq \log r \\
2[l(\theta) - l(\hat{\theta})] &\geq 2 * \log r \\
-3.84 &\geq 2 * \log r \\
r &\approx 0.146
\end{aligned}$$

A Tabela 1.1 mostra a relação entre alguns valores de corte da verossimilhança relativa e os valores correspondentes em log-verossimilhança e em *deviance*. Na prática os níveis de confiança são aproximados e válidos apenas assintoticamente.

Com isso, podemos observar que usar a verossimilhança relativa ou a *deviance* são formas equivalentes de construir o intervalo de confiança. A diferença está na interpretação que é associada aos intervalos. Uma discussão

Tabela 1.1: Relação entre valores de corte para obtenção de intervalos de confiança com funções $LR(\theta)$, $l(\theta)$ e $D(\theta)$

r	c	c^*	$P[\|Z\| < \sqrt{c^*}]$
50%	0,693	1,386	0,761
26%	1.661	3,321	0,899
15%	1,897	3,794	0,942
3,6%	3,324	6,648	0,990

aprofundada pode ser encontrada em Royall (1997). A vantagem em usar a *deviance* é que conhecemos a sua distribuição assintótica o que permite a construção de intervalos de confiança com a desejada estrutura probabilística. Da mesma forma que para a *deviance*, no exemplo, a obtenção do intervalo pela verossimilhança relativa também requer resolver um sistema não-linear utilizando algum método numérico para encontrar as raízes da equação $LR(\theta) \geq r$.

Considere dados gerados com os comando a seguir.

```
> set.seed(123) ; y <- rexp(20, rate=1)
```

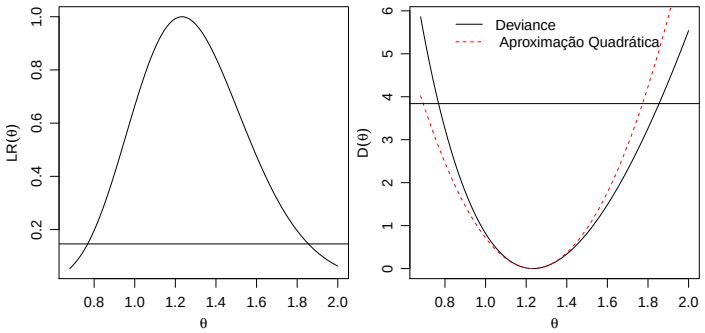

Figura 1.3: Verossimilhança relativa (esquerda) e função deviance exata e aproximada (direita) – Distribuição exponencial.

Como é possível ver na Figura 1.3 a função *deviance* é razoavelmente aproximada por uma forma quadrática mas esta aproximação vai depender do valor do parâmetro e do tamanho da amostra. Para obter os limites inferior e superior do intervalo de confiança pela aproximação quadrática temos:

$$\hat{\theta}_L = \hat{\theta} - z_{\frac{\alpha}{2}}\sqrt{\hat{\theta}^2/n} \quad \text{e} \quad \hat{\theta}_U = \hat{\theta} + z_{\frac{\alpha}{2}}\sqrt{\hat{\theta}^2/n}$$

Para a amostra simulada utilizada para gerar os gráficos temos os seguintes valores:

$$\hat{\theta}_L = 1.23(1 - 1.96\sqrt{1/20}) \quad \text{e} \quad \hat{\theta}_U = 1.23(1 + 1.96\sqrt{1/20}),$$

que resulta no seguinte intervalo de confiança:

$$(\hat{\theta}_L = 0.69 \quad ; \quad \hat{\theta}_U = 1.77).$$

Usando a função *deviance* precisamos resolver o sistema não-linear. Em R podemos facilmente resolver este problema usando as funções do pacote **rootSolve** Soetaert (2009). O primeiro passo é escrever a função *deviance*,

Código 1.1: IC baseado na *deviance* – distribuição exponencial.

```
> ICdevExp <- function(theta, theta.hat, y, nivel=0.95){
+   n <- length(y)
+   dv <- 2*n*( log( theta.hat/theta) + mean(y)*(theta- theta.hat))
+   return(dv - qchisq(nivel,df=1))
+ }
```

Uma vez com a função escrita podemos encontrar suas raízes usando a função `uniroot.all()`.

```
> require(rootSolve)
> uniroot.all(ICdevExp, interval=c(0,10), theta.hat=1/mean(y),y=y)
[1] 0.7684028 1.8547415
```

A Figura 1.3 mostra o intervalo aproximado pela forma quadrática com um deslocamento para a esquerda quando comparado com o intervalo baseado na função *deviance*. É importante ter bastante cuidado na interpretação destes intervalos. De acordo com a interpretação frequentista de probabilidade, se realizarmos o mesmo experimento um grande número de vezes e em cada um destes experimentos calcularmos o respectivo intervalo de confiança esperamos que $(1-\alpha)$ 100% dos intervalos construídos contenham o verdadeiro valor do parâmetro. Isto pode ser ilustrado com o seguinte estudo de simulação em que geramos 100 amostras cada uma de tamanho $n = 70$, com valor do parâmetro igual a 1. Para cada amostra verificamos a taxa de cobertura, isto é, construímos o intervalo de confiança e ao final verificamos a proporção dos intervalos que contém o valor do parâmetro.

```
> THETA <- 1
> set.seed(12)
> ic <- matrix(NA, ncol=2, nrow=100)
> for(i in 1:100){
+   y <- rexp(70, rate=THETA)
+   est <- 1/mean(y)
+   ic[i,] <- uniroot.all(ICdevExp, int=c(0,5), theta.hat=est, y=y)
+ }
> mean(apply(ic, 1, function(x) (x[1] < THETA & x[2] > THETA)))
[1] 0.95
```

No código acima simulamos a cada passo do laço `for()` uma nova realização da variável aleatória Y, com esta realização calculamos a estimativa

de máxima verossimilhança e o seu respectivo intervalo de confiança baseado na *deviance* e guardamos o resultado em um objeto chamado *ic*. De acordo com a interpretação frequentista dos 100 intervalos de confiança que calculamos esperamos que 95 deles contenham o verdadeiro valor do parâmetro neste caso $\theta = 1$. O gráfico apresentado na Figura 1.4 ilustra os resultados. Neste caso, conforme esperado, temos exatamente que 5 dos intervalos não contem o valor verdadeiro do parâmetro. É claro que por se tratar de um exemplo de simulação variações podem ocorrer.

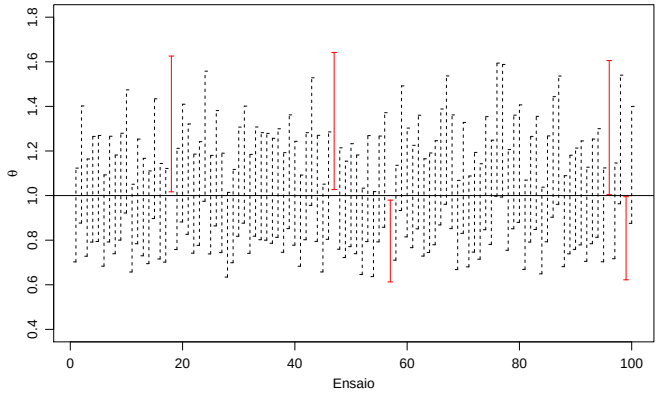

Figura 1.4: Interpretação frequentista de intervalos de confiança.

A taxa de cobertura de um intervalo é a proporção de vezes em que os intervalos contem o verdadeiro valor do parâmetro sobre o total de ensaios simulados. Neste caso obtivemos exatamente 95/100, ou seja, a taxa de cobertura é de 95% igual ao nível nominal especificado para este intervalo. A taxa de cobertura é uma forma muito utilizada para avaliar e comparar métodos de construção de intervalos de confiança. Em modelos mais complexos principalmente os que envolvem efeitos aleatórios, os componentes de variância são de difícil estimação e os seus intervalos de confiança principalmente os construídos baseados em aproximação quadrática da log-verossimilhança apresentam taxa de cobertura abaixo do nível nominal especificado.

Para encerrar este exemplo vamos redefinir a função para obtenção do intervalo de confiança a partir da função *deviance*. Com isto pretendemos chamar a atenção para cuidados em não efetuar cálculos desnecessários nos códigos. Na função original `n <- length(y)` e `mean(y)` são avaliados a cada chamada da função. Entretanto, para uma determinada amostra, estas quantidades são constantes. Em procedimentos numéricos a função é avaliada muitas vezes e portanto estas constantes são recalculadas desnecessariamente a cada avaliação. É melhor então definir a função já recebendo estas

constantes que são estatísticas suficientes que resumem a amostra. Usamos ainda o fato que $\hat{\theta} = 1/\overline{y}$.

Código 1.2: Redefinição da função para obter IC baseado na *deviance* – distribuição exponencial.

```
> ICdevExp <- function(theta, amostra, nivel=0.95){
+   ## amostra é um vetor com elementos n e mean(y), nesta ordem
+   n <- amostra[1]
+   med <- amostra[2]
+   dv <- 2*n*(-log(med*theta) + med*theta - 1)
+   return(dv - qchisq(nivel, df=1))
+ }
> am <- c(length(y), mean(y))
> uniroot.all(ICdevExp, interval=c(0,10), amostra=am)
[1] 0.8756117 1.3998842
```

1.7 Exemplo 3 - Testes de hipóteses

Em situações práticas podemos estar interessados em testar se o parâmetro de um dado modelo é igual, maior ou menor que algum valor de interesse. Conforme descrito na Seção 1.4, um teste de hipóteses é qualquer afirmação acerca da distribuição de probabilidade de uma ou mais variáveis aleatórias. Considere a situação onde observamos uma amostra aleatória de tamanho n de uma população com distribuição de Poisson de parâmetro λ. Suponha que o interesse é testar sob a hipótese nula $H_0 : \lambda = \lambda_0$ contra uma hipótese alternativa $H_1 : \lambda \neq \lambda_0$. Vimos na Seção 1.4 três formas de construir testes de hipóteses que podem ser usadas para concluir sobre as hipóteses levantadas. Como exemplo didático, vamos obter as três estatísticas de testes para o caso onde $X_i \sim P(\lambda)$. Antes de construir os testes propriamente dito vamos obter algumas quantidades relevantes que facilitam a construção dos testes.

Como as amostras são independentes a função de verossimilhança deste modelo é dada por,

$$L(\lambda) = \prod_{i=1}^{n} \frac{\exp\{\lambda\}\lambda_i^x}{x_i!} = \frac{\exp\{-n\lambda\}\lambda^{\sum_{i=1}^n x_i}}{\prod_{i=1}^n x_i!}.$$

A função de log-verossimilhança,

$$l(\lambda) = -\lambda n + \sum_{i=1}^{n} \log\lambda - \sum_{i=1}^{n} \log x_i!.$$

A função escore é obtida derivando a log-verossimilhança em λ,

$$U(\lambda) = -n + \frac{\sum_{i=1}^{n} x_i}{\lambda}$$

que sendo resolvida fornece a estimativa de máxima verossimilhança

$$\hat{\lambda} = \frac{\sum_{i=1}^{n} x_i}{n} = \overline{x}.$$

Além disso, temos que a informação observada é dada por

$$I_O(\lambda) = \frac{\sum_{i=1}^{n} x_i}{\lambda^2}.$$

Para obter informação esperada avaliamos a esperança da informação observada

$$I_E(\lambda) = E(I_O(\lambda)) = E(\frac{\sum_{i=1}^{n} x_i}{\lambda^2}) = \frac{n}{\lambda}.$$

Vamos começar pelo teste de razão de verossimilhança. A estatística do teste de razão de verossimilhança neste caso toma a seguinte forma:

$$\lambda(\underline{x}) = \frac{L(\lambda_0|\underline{x})}{L(\hat{\lambda}|\underline{x})},$$

que é a verossimilhança relativa. Note que $-2\log\lambda(\underline{x})$ é exatamente a função *deviance* usada para construir intervalos de confiança. Então,

$$-2\log\lambda(\underline{x}) = -2\log\left(\frac{\exp\{-n\lambda_0\}\lambda_0^{\sum_{i=1}^{n} x_i}}{\exp\{-n\hat{\lambda}\}\hat{\lambda}^{\sum_{i=1}^{n} x_i}}\right) = 2n\left[(\lambda_0 - \hat{\lambda}) - \hat{\lambda}\log\left(\frac{\lambda_0}{\hat{\lambda}}\right)\right].$$

A hipótese nula $H_0 : \lambda = \lambda_0$ será rejeitada se, e somente se, $-2\log\lambda(\underline{x}) \geq \chi^2_{1,\alpha}$. A probabilidade de Erro do Tipo I será aproximadamente α.

O segundo método para construção de testes de hipóteses é o método de Wald. Se $\hat{\lambda}$ é o estimador de máxima verossimilhança a estatística do teste de Wald é dada por

$$T_w = \frac{\hat{\lambda} - \lambda}{\sqrt{V(\hat{\lambda})}} \sim N(0,1).$$

Sabemos que $V(\hat{\lambda}) = I_E(\hat{\lambda})^{-1}$, então $V(\hat{\lambda}) = \frac{\hat{\lambda}}{n}$. Logo o teste de Wald no caso Poisson resume-se a

$$T_w = \frac{\overline{x} - \theta}{\sqrt{\overline{x}/n}} \sim N(0,1).$$

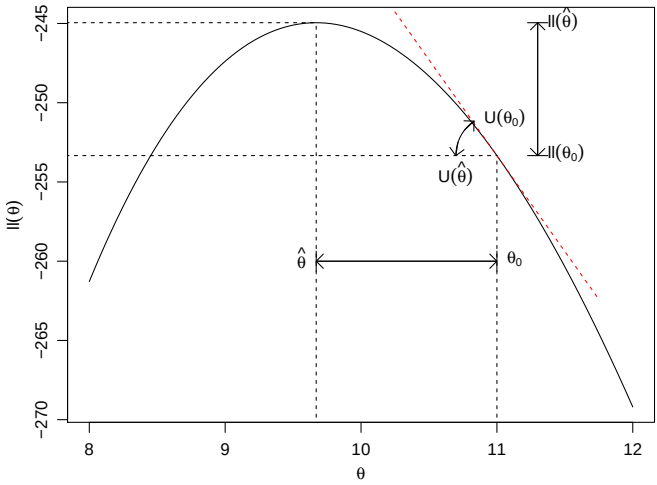

Figura 1.5: Diferentes formas de construir teste de hipótese baseado em verossimilhança.

Dado a distribuição assintótica do teste para encontrar a região de rejeição basta encontrar o quantil da distribuição gaussiana padrão correspondente ao nível de confiança desejado.

A terceira opção é a estatística de teste *escore* que é dada por,

$$T_E = \frac{U(\lambda_0)}{\sqrt{I_E(\lambda_0)}}.$$

Substituindo as quantidades previamente encontradas temos para o caso Poisson,

$$T_E = \frac{-n + \sum_{i=1}^{n} x_i/\lambda_0}{\sqrt{n/\lambda_0}} \sim N(0,1).$$

A Figura 1.5 ilustra a construção dos testes de hipóteses e os relaciona à função de verossimilhança do modelo. O teste da razão de verossimilhança compara valores no eixo vertical, ou seja, compatibilidades com os dados. O teste escore avalia o quanto a curvatura da função no ponto especificado pelo teste se afasta de zero, o valor no máximo. O teste de Wald avalia a distância entre o valor especificado no teste e o EMV, padronizando esta distância pela curvatura da função ao redor de seu máximo.

Para exemplificar a execução do teste de hipóteses vamos utilizar um conjunto de dados simulados com $n = 100$ amostras aleatórias de uma Poisson com $\lambda = 10$. Vamos supor que o interesse seja testar sob $H_0 : \lambda = 8$ contra $H_1 : \lambda \neq 8$. As funções abaixo montam a estrutura do teste de hipótese de acordo com cada abordagem.

Código 1.3: Função genérica para aplicar o teste de razão de verossimilhanças.

```
> trv <- function(Est, H0, alpha, ...){
+   critico <- qchisq(1-alpha, df=1)
+   est.calc <- Est(H0, ...)
+   print(ifelse(est.calc < critico, "Aceita H0", "Rejeita H0"))
+   return(c(est.calc,critico))}
```

Código 1.4: Função genérica para aplicar o teste de Wald.

```
> wald <- function(H0, EMV, V.EMV, alpha){
+   critico <- qnorm(1-alpha/2)
+   Tw <- (EMV - H0)/sqrt(V.EMV)
+   print(ifelse(Tw < critico, "Aceita H0", "Rejeita H0"))
+   return(c(Tw,critico))
+ }
```

Código 1.5: Função genérica para aplicar o teste de escore.

```
> escore <- function(H0, U, Ie, alpha, ...){
+   critico <- qnorm(1-alpha/2)
+   Te <- U(H0,...)/sqrt(Ie(H0,...))
+   print(ifelse(Te < critico, "Aceita H0", "Rejeita H0"))
+   return(c(Te,critico))
+ }
```

A seguir aplicamos as funções para testes com dados da amostra simulada.

```
> set.seed(123)
> x <- rpois(100, lambda=10)
> ## Estatistica do TRV caso Poisson
> Est <- function(H0, x){
+   n <- length(x)
+   EMV <- mean(x)
+   lv <- 2*n*(( H0 - EMV) + EMV*log(EMV/H0))
+ return(lv)}
> ## Procedendo com o TRV
> trv(Est = Est, H0=8, alpha = 0.05, x=x)
[1] "Rejeita H0"
[1] 32.660809  3.841459

> ## Teste de Wald
> wald(H0=8, EMV = mean(x), V.EMV = mean(x)/length(x),alpha=0.05)
[1] "Rejeita H0"
[1] 5.370358 1.959964
```

O teste escore é ligeiramente mais complicado uma vez que é necessário programar a função escore e uma função para avaliar a informação esperada.

Código 1.6: Função escore para a distribuição de Poisson.

```
> fc.escore <- function(lambda,x){
+   n <- length(x)
+   esco <- -n + sum(x)/lambda
+   return(esco)}
```

Código 1.7: Informação esperada para a distribuição Poisson.

```
> Ie <- function(lambda,x){
+   n <- length(x)
+   I <- n/lambda
+   return(I)}
```

```
> escore(H0 = 8, U = fc.escore, Ie = Ie, alpha=0.05, x=x)
[1] "Rejeita H0"
[1] 5.904342 1.959964
```

Neste caso os três testes levam a mesma conclusão. Em geral, o teste escore é o mais complicado de se obter, uma vez que precisa da função escore e da informação esperada ou observada. Apesar destas quantidades poderem ser obtidas numericamente em geral o esforço computacional é maior que pelas outras duas abordagens. O teste de Wald é o mais utilizado pelo menos de forma inicial uma vez que sua construção é simples. O teste de razão de verossimilhança é também muito utilizado, tanto para testar valores para um determinado parâmetro quanto para a escolha de modelos estatísticos. Na situação em que usamos métodos numéricos para maximização da função de log-verossimilhança um subproduto é a informação observada que pode ser usada diretamente na estatística de teste de Wald. Além disso, quando comparamos modelos aninhados a diferença entre os valores da log-verossimilhança dos modelos, permite a construção do teste da razão de verossimilhança de forma bastante direta, porém requer duas otimizações, uma para cada modelo, enquanto que o de Wald apenas uma.

1.8 Exemplo 4 - Reparametrização

Em diversas aplicações o interesse principal pode ser sobre alguma função de um parâmetro do modelo. Por exemplo em uma distribuição exponencial de parâmetro θ o interesse pode estar na probabilidade de obter um

valor maior que k expressa por $\psi = \exp\{-\theta k\}$, o que pode ser visto como uma reparametrização. Além disto, por vezes pode ser mais simples estimar em uma certa parametrização do que em outra. Por exemplo, no caso de parâmetros de variância em geral é mais estável numericamente estimar a sua raiz ou o log da raiz. Note também que reparametrizações mudam as regiões de busca em algoritmos de maximização. Por exemplo, em parâmetros de variância digamos σ^2 tem como seu intervalo de busca o $\Re^+$, enquanto que se ao invés de estimar σ^2 estimarmos um $\phi = \log\sqrt(\sigma^2)$ o intervalo de busca será toda a reta real, o que normalmente é mais conveniente quando trabalha-se com algoritmos de maximização numérica.

Como exemplo ilustrativo deste problema, seja $X_i : i = 1,\ldots,n$ variáveis aleatórias independentes com função densidade de probabilidade dada por:

$$f(x;\theta) = 2\theta x \exp\{-\theta x^2\} \quad : \quad x \geq 0.$$

Considere que seja de interesse a reparametrização $\theta = \frac{1}{\mu}$. Vamos primeiro fazer todo o processo de inferência considerando que queremos estimar o parâmetro θ. Na sequencia consideramos todo o processo considerando o parâmetro μ. Para finalizar mostramos como passar de uma reparametrização para outra, ou seja, como obter as estimativas tanto pontuais quanto intervalares para μ a partir das estimativas de θ. Começamos escrevendo a função de verossimilhança,

$$L(\theta) = \prod_{i=1}^{n} 2\theta x_i \exp\{-\theta x_i^2\} = (2\theta)^n \prod_{i=1}^{n} \log x_i \exp\{-\theta \sum_{i=1}^{n} x_i\},$$

logo a função de log-verossimilhança,

$$l(\theta) = n\log 2 + n\log\theta + \sum_{i=1}^{n} \log x_i - \theta \sum_{i=1}^{n} x_i^2.$$

Derivando em relação a θ chegamos a função escore,

$$U(\theta) = \frac{n}{\theta} - \sum_{i=1}^{n} x_i^2.$$

Igualando a zero encontramos a estimativa de máxima verossimilhança,

$$\hat{\theta} = \frac{n}{\sum_{i=1}^{n} x_i^2}.$$

Para a construção do intervalo de confiança usando a aproximação quadrática precisamos da segunda derivada, da qual derivamos a informação observada e/ou esperada, neste caso temos,

$$I_O(\theta) = -\frac{\partial l(\theta)}{\partial\theta} = \frac{n}{\theta^2}.$$

Para obter a informação esperada basta obter a esperança da informação observada,

$$I_E(\theta) = E(I_O(\theta)) = E\left(\frac{n}{\theta^2}\right) = \frac{n}{\theta^2}.$$

Note que neste caso em particular $I_O(\theta) = I_E(\theta)$, porém em geral isso não é válido . Com as expressões anteriores podemos estimar o parâmetro θ e encontrar um intervalo de confiança aproximado usando a aproximação quadrática. Para obter intervalos de confiança baseado na função *deviance* precisamos resolver a seguinte equação não-linear,

$$D(\theta) = 2\left[n\log\left(\frac{\hat{\theta}}{\theta}\right) + (\hat{\theta} - \theta)\sum_{i=1}^{n} x_i^2\right] \leq c^*.$$

Isto pode ser resolvido usando algum método numérico conforme será explicado na sequência. Por agora, podemos resolver conforme no Exemplo 1.6, usando a função `uniroot.all()`, que implementa o Newton-Raphson para uma função qualquer. Com isso, temos todo o processo de inferência completo, podemos estimar pontualmente, obter intervalo de confiança aproximado ou baseado na *deviance*.

Mas não estamos interessados em θ, mas sim em μ. Podemos reescrever então a verossimilhança como função de μ.

$$L(\mu) = \prod_{i=1}^{n} 2\mu^{-1}x_i \exp\{-\frac{x_i^2}{\mu}\} = (2\mu^{-1})^n \exp\{-\frac{1}{\mu}\sum_{i=1}^{n}\} \prod_{i=1}\log x_i.$$

A log-verossimilhança é dada por,

$$l(\mu) = n\log 2 - n\log\mu - \mu^{-1}\sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n}\log x_i.$$

Derivando em μ obtemos a função escore,

$$U(\mu) = -\frac{n}{\mu} + \mu^{-2}\sum_{i=1}^{n} x_i^2.$$

Igualando a zero chegamos a estimativa de máxima verossimilhança,

$$\hat{\mu} = \frac{\sum_{i=1}^{n} x_i^2}{n}.$$

Para a informação observada tem-se

$$\begin{aligned} I_O(\mu) &= -\frac{\partial^2 l(\mu)}{\partial\mu} = -\frac{\partial}{\partial\mu} - \frac{n}{\mu} + \mu^{-2}\sum_{i=1}^{n} x_i^2 \\ &= n\mu^{-2} - 2\mu^{-3}\sum_{i=1}^{n} x_i^2. \end{aligned}$$

Para obter a informação esperado precisamos tomar a esperança de $I_O(\mu)$,

$$I_E(\mu) = E(I_O(\mu)) = E\left(n\mu^{-2} - 2\mu^{-3}\sum_{i=1}^{n} X_i^2\right).$$

Neste passo é necessário calcular a $E(X_i^2)$ que é a solução da integral $E(X_i^2) = \int_0^\infty X_i^2 2\mu^{-1} X_i \exp\{-\frac{X_i^2}{\mu}\}dx = 2\mu$ que neste caso é possível de ser obtida analiticamente. Substituindo na equação acima temos,

$$I_E(\mu) = E(-n\mu^{-2} + 2\mu^{-3}n2\mu) = n\mu^{-2}$$

Neste caso a informação observada é diferente da esperada e que isto afeta a construção do intervalo de confiança baseado na aproximação quadrática, uma vez que muda a estimativa de variância do estimador de máxima verossimilhança. As duas formas são equivalentes assintoticamente, porém na situação real temos apenas uma amostra finita observada. Na maioria das situações em modelagem estatística quando métodos numéricos são utilizados não temos a opção de escolher entre a informação observada e esperada, quando a segunda derivada é obtida numericamente estamos diretamente usando a informação observada. Podemos dizer que usar a informação observada é acreditar plenamente na amostra observada, enquanto usar a informação esperada estamos emprestando mais informação do modelo. Pensamos na informação observada como uma medida de curvatura local, por outro lado a informação esperada é uma medida de curvatura global.

A construção de intervalos de confiança baseados diretamente na função *deviance* é feita resolvendo a seguinte equação não linear,

$$D(\mu) = 2\left[n\log\left(\frac{\mu}{\hat{\mu}}\right) + (\mu^{-1} - \hat{\mu}^{-1})\sum_{i=1}^{n} x_i^2\right].$$

Para exemplificar considere que a seguinte amostra foi observada $x_i = 0.19; 1.68; 2.81; 0.59; 1.18$. Vamos fazer um gráfico da verossimilhança em cada parametrização. Começamos escrevendo as funções em R.

```
> L.theta <- function(theta,x){
+    n <- length(x)
+    return((2*theta)^n * prod(x) * exp(-theta*sum(x^2)))
+   }
> L.mu <- function(mu,x){
+    n <- length(x)
+    return((2*mu^-1)^n * prod(x) * exp( -(1/mu)*sum(x^2)))
+    }
```

Vamos entrar com os dados no R em forma de vetor e calcular as estimativas de máxima verossimilhança para θ e μ.

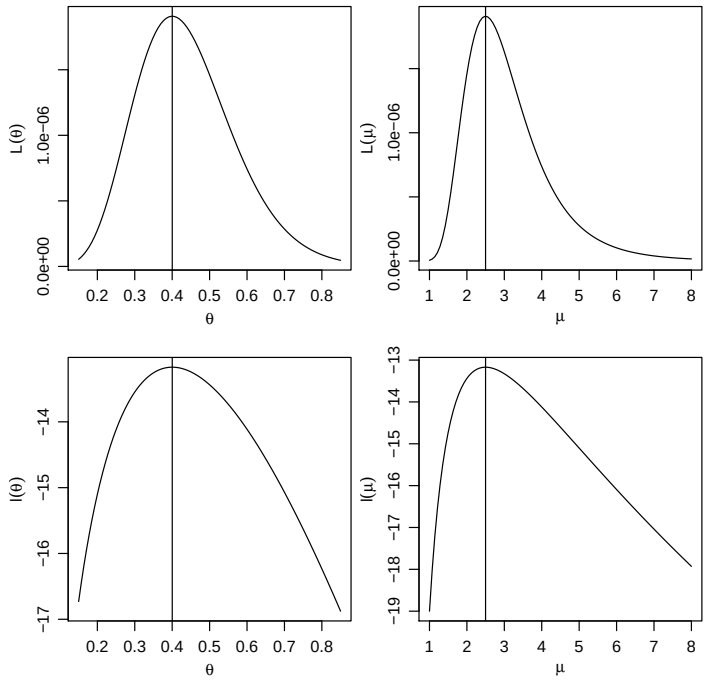

Figura 1.6: Verossimilhança e log-verossimilhança para θ e μ.

```
> x <- c(0.19,1.68,2.81,0.59,1.18,0.19,1.68,2.81,0.59,1.18)
> theta <- length(x)/sum(x^2)
> mu <- sum(x^2)/length(x)
> c(theta, mu)
```

```
[1] 0.4001569 2.4990200
```

Entretanto, na prática não conveniente nem necessário reescrever a função de verossimilhança para cada parâmetro de interesse. Pela propriedade de invariância pode-se obter μ partindo de θ e vice-versa. Os gráficos da função de verossimilhança e log-verossimilhança nas duas parametrizações são apresentados na Figura 1.6.

Para a construção dos intervalos de confiança baseados na *deviance* em cada parametrização basta resolver as respectivas equações não lineares. Além disso, podemos obter os intervalos de confiança aproximados usando a informação observada e/ou esperada, dependendo da situação. A Figura 1.6, apresenta as funções *deviance* exatas e aproximadas em cada parametrização. Para isto, precisamos criar cada uma das funções, para depois poder avalia-las.

Código 1.8: Funções *deviance* para o parâmetro θ e μ.

```
> dev.theta <- function(theta, theta.hat, x, desloca=0){
+   saida <- 2*(length(x)*log(theta.hat/theta) -
+                         (theta.hat - theta)*sum(x^2))
+   return(saida - desloca)
+ }
> dev.mu <- function(mu, mu.hat, x, desloca=0){
+   saida <- 2*(length(x)*log(mu/mu.hat) +
+               ((1/mu)-mu.hat^-1)*sum(x^2))
+   return(saida - desloca)
+ }
> dev.app.theta <- function(theta, theta.hat, x){
+   return((theta - theta.hat)^2 * (length(x)/theta.hat^2))
+ }
> dev.app.mu.obs <- function(mu, mu.hat,  x){
+   Io <- -((length(x)/mu.hat^2) - (2*sum(x^2)/mu.hat^3))
+   return((mu - mu.hat)^2 * Io)
+ }
> dev.app.mu.esp <- function(mu, mu.hat, x){
+   Ie <- length(x)/(mu.hat^2)
+   return((mu - mu.hat)^2 *Ie)
+ }
```

Como é possível ver na Figura 1.7 a função *deviance* apresenta um comportamento bastante assimétrico o que torna a aproximação quadrática ruim. Neste caso o pequeno tamanho da amostra prejudica a aproximação quadrática.

De acordo com as propriedades do estimador de máxima verossimilhança, sabemos que $\hat{\theta} \sim N(\theta, I_E(\theta)^{-1})$, assintoticamente podemos usar ao invés de $I_E(\theta)$ tanto a $I_E(\hat{\theta})$ como também $I_O(\hat{\theta})$. Sabemos também que para θ a informação esperada e a observada coincidem. Então o intervalo assintótico fica dado por:

$$\hat{\theta}_L = \hat{\theta} - z_{\frac{\alpha}{2}}\sqrt{\hat{\theta}^2/n} \quad \text{e} \quad \hat{\theta}_U = \hat{\theta} - z_{\frac{\alpha}{2}}\sqrt{\hat{\theta}^2/n}.$$

O mesmo argumento assintótico é usado para construir o intervalo para μ, porém aqui a informação esperada difere da observada o que neste caso não faz tanta diferença, temos que a informação esperada é dada por $I_E(\hat{\mu}) = \frac{\hat{\mu}^2}{n}$ enquanto que a informação observada é dada por $I_O(\hat{\mu}) = -\frac{n}{\hat{\mu}^2} + \frac{2\sum_{i=1}^{N} x_i^2}{\hat{\mu}^3}$, e o intervalo é construído exatamente igual ao anterior.

Para os intervalos baseados diretamente na função *deviance* precisamos resolver as respectivas equações não lineares. Isto pode ser resolvido numericamente usando a função `uniroot.all()`. Vamos obter os intervalos de

confiança e fazer uma comparação qualitativas em ambas as parametrizações. Para isto criamos uma função genérica que recebe a estimativa de máxima verossimilhança, a informação esperada ou observada e o nível de confiança do intervalo e retorna os seus limites inferior e superior.

Código 1.9: Função genérica para construir intervalo de confiança de Wald.

```
> ic.assintotico <- function(EMV, Io, alpha){
+   return(EMV + c(-1, 1) * qnorm(1-alpha/2)*sqrt(Io^(-1)))
+ }
```

Usando a função criada obtemos os intervalos assintóticos para θ e μ.

```
> n <- length(x)
> theta.est <- n/sum(x^2)
> ic.theta <- ic.assintotico(EMV = theta.est,
+             Io = (n / theta.est^2), alpha=0.05)
> mu.est <- sum(x^2)/n
> ic.mu.obs <- ic.assintotico(EMV = mu.est,
+              Io = -n/mu.est^2 + (2*sum(x^2))/(mu.est^3), alpha=0.05)
> ic.mu.esp <- ic.assintotico(EMV = mu.est,
+              Io = n/mu.est^2, alpha=0.05)
```

Vamos também criar uma função genérica à qual tem como seu argumento principal a função *deviance* do modelo.

Código 1.10: Função genérica para construir intervalo de confiança baseado na *deviance*.

```
> ic.deviance <- function(dev,intervalo,...){
+   ic <- uniroot.all(dev, interval=intervalo, ...)
+   return(ic)
+ }
```

Usamos a função criada para obter os intervalos de confiança,

```
> ic.dev.theta <- ic.deviance(dev=dev.theta, intervalo=c(0,10),
+             theta.hat=theta.est, x=x, desloca=qchisq(0.95, df=1))
> ic.dev.mu <- ic.deviance(dev=dev.mu, intervalo=c(0,100),
+             mu.hat=mu.est, x=x, desloca=qchisq(0.95, df=1))
```

A seguir comparamos os intervalos obtidos e notamos que a discrepância entre intervalos pela deviance exata e aproximada é maior para μ. Isto é explicado pelo fato da verossimilhança de μ ser mais assimétrica do que a de θ. Este comportamento pode ser visualizado na Figura 1.7. Portanto, intervalos baseados em aproximação quadráticas terão taxa de cobertura mais aproximadas das nominais A recomendação é a de que a codificação da função de verossimilhança seja sempre feita na parametrização que fornece a função mais próxima de um comportamento quadrático e isto

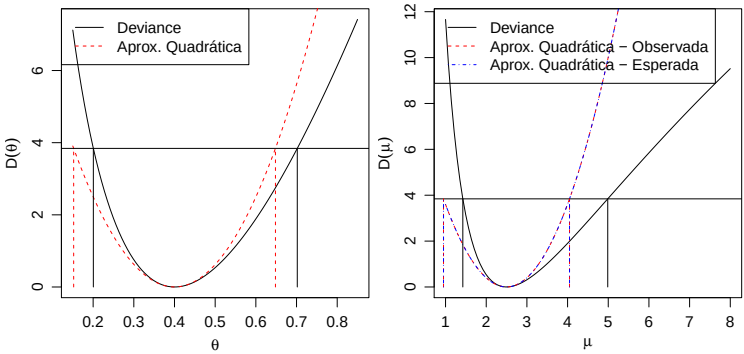

Figura 1.7: Deviances exatas e aproximadas para cada parametrização e intervalos de confiança.

é especialmente relevante se intervalos aproximados (assintóticos) vão ser utilizados.

```
> rbind(ic.theta, ic.dev.theta)
```

```
                [,1]      [,2]
ic.theta     0.1521416 0.6481721
ic.dev.theta 0.2005100 0.7018490
```

```
> rbind(ic.dev.mu, ic.mu.obs,ic.mu.esp)
```

```
              [,1]     [,2]
ic.dev.mu 1.4248010 4.987302
ic.mu.obs 0.9501398 4.047900
ic.mu.esp 0.9501398 4.047900
```

Em situações prática não é necessário percorrer todo o caminho feito neste exemplo, que desenvolvemos aqui de forma completa para ilustrar e reforçar conceitos, já que, reparametrização é muito comum quando ajusta-se modelos mais complexos. Considere que temos apenas as estimativas pontuais e intervalares de θ e desejamos obter as estimativas pontual e intervalar para μ. Temos que $\theta = 1/\mu$, logo $\mu = 1/\theta$ por invariância $\hat{\mu} = 1/\hat{\theta}$. Os intervalos obtidos diretamente baseado na função *deviance* também são invariantes então basta aplicar a transformação inversa igual a estimativa pontual. Quando usamos a aproximação quadrática os intervalos não são invariantes, é neste ponto que usamos o Teorema 1.4. No exemplo, obtemos o intervalo aproximado para θ usando por exemplo a informação esperada e desejamos o intervalo para μ. Usando o Teorema 1.4 sabemos que $\mu = g(\theta)$ e conhecemos a $V(\hat{\theta}) = \hat{\theta}^2/n$, partindo disto precisamos encontrar a variância para $\hat{\mu}$, o Teorema 1.4 diz que $V(\hat{\mu}) = g'(\hat{\theta})^2 I_E(\hat{\theta})^{-1}$, onde $g'(\cdot)$ representa a primeira derivada da função $g(\cdot)$. Sendo assim,

$$\hat{\mu} = \frac{1}{\hat{\theta}} = g(\hat{\theta}) \rightarrow g'(\hat{\theta}) = -\frac{1}{\hat{\theta}^2}.$$

Logo, temos que

$$\begin{aligned} V(\hat{\mu}) &= \left(-\frac{1}{\hat{\theta}^2}\right)^2 \frac{\hat{\theta}^2}{n} \\ &= \frac{1}{\hat{\theta}^4}\frac{\hat{\theta}^2}{n} = \frac{1}{\hat{\theta}^2 n} \\ &= \frac{1}{0.400^2 * 10} = 0.625 \end{aligned}$$

Fazendo as contas no R,

```
> V.mu <- 1/(theta.est^2 * n)
> ic.mu.theta <- c(1/theta.est - qnorm(0.975)*sqrt(V.mu),
+                  1/theta.est + qnorm(0.975)*sqrt(V.mu))
```

Comparando com o intervalo obtido anteriormente,

```
> cbind(ic.mu.theta, ic.mu.esp)
```

```
     ic.mu.theta ic.mu.esp
[1,]   0.9501398 0.9501398
[2,]   4.0479002 4.0479002
```

Os intervalos são idênticos porém com um esforço de obtenção muito menor do que redefinir a verossimilhança. O Teorema 1.4 as vezes é chamado de *método Delta* para obter variância de estimadores. Estes resultados dos estimadores de máxima verossimilhança são muito utilizados quando estamos programando modelos mais complexos, principalmente modelos com efeitos aleatórios, onde diversos parâmetros de variância/precisão serão estimados. De forma geral, estes tipos de estimadores vão apresentar na parametrização original, uma distribuição amostral bastante assimétrica, além de problemas de representação numérica, tornando o uso de algoritmos numéricos difícil. Reparametrizações em termos de raízes e logaritmo são muito utilizadas para estabilizar os algoritmos numéricos e tornar a distribuição amostral dos estimadores mais próxima da gaussiana, o que também ajuda para a construção de intervalos baseados na aproximação quadrática. No caso de intervalos baseados na função deviance (exata), pela invariância a transformação de uma parametrização para outra é feita diretamente aplicando-se a transformação nos limites do intervalo. Para o caso da aproximação quadrática, a volta para a parametrização original pode ser feita pelo método Delta. Desta forma é possível manter as interpretações desejadas em termos dos parâmetros de interesse do modelo.

Vimos neste exemplo também, que mesmo em situações simples os intervalos baseados na função *deviance* são de difícil obtenção e requerem algoritmos numéricos para sua obtenção. Porém de forma geral são mais realistas, no sentido de representar melhor a incerteza associada a estimação do parâmetro e possuírem propriedades ótimas.

1.9 Exemplo 5 - Distribuição Gaussiana

Suponha que $Y_1, Y_2, \ldots, Y_n$ são variáveis aleatórias independentes e identicamente distribuídas com distribuição gaussiana de média μ e variância σ^2. Denote isto por $Y_i \sim N(\mu, \sigma^2)$. Note que neste caso o vetor de parâmetros $\underline{\theta} = (\mu, \sigma)^\top$. Onde $\mu \in \Re$ e $\sigma \in \Re^+$ são os respectivos espaços paramétricos. O objetivo é estimar μ e σ além de encontrar intervalos ou regiões de confiança. Note que agora temos dois parâmetros e estamos trabalhando com uma superfície de log-verossimilhança. Os princípios vistos no caso uniparamétrico se mantém mas a construção de gráficos e a obtenção das estimativas são mais trabalhosas. Vejamos alguns fatos relevantes deste exemplo. Como sempre, começamos escrevendo a função de verossimilhança,

$$L(\mu, \sigma) = (2\pi)^{-n/2}\sigma^{-n} \exp\{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu)^2\}.$$

A log-verossimilhança é dada por,

$$l(\mu,\sigma) = -\frac{n}{2}\log 2\pi - n\log\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu)^2.$$

A função *escore* toma a forma de um sistema de equações,

$$U(\mu) = \frac{\partial l(\mu,\sigma)}{\partial \mu} = \frac{\sum_{i=1}^{n} y_i}{\sigma^2} - \frac{n\mu}{\sigma^2}$$
$$U(\sigma) = -\frac{n}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{n}(y_i - \mu)^2.$$

Neste caso podemos facilmente resolver este sistema chegando as estimativas de máxima verossimilhança,

$$\hat{\mu} = \frac{\sum_{i=1}^{n} y_i}{n} \quad \text{e} \quad \hat{\sigma} = \frac{\sum_{i=1}^{n}(y_i - \mu)^2}{n}.$$

A matriz de informação observada fica da seguinte forma,

$$I_O(\mu,\sigma) = \begin{bmatrix} -\frac{\partial^2 l(\mu,\sigma)}{\partial \mu^2} & -\frac{\partial^2 l(\mu,\sigma)}{\partial \mu \partial \sigma} \\ -\frac{\partial^2 l(\mu,\sigma)}{\partial \mu \partial \sigma} & -\frac{\partial^2 l(\mu,\sigma)}{\partial \sigma^2} \end{bmatrix}.$$

Temos então,

$$\frac{\partial^2 l(\mu,\sigma)}{\partial \mu^2} = \frac{\partial U(\mu)}{\partial \mu} = -\frac{n}{\sigma^2}$$
$$\frac{\partial^2 l(\mu,\sigma)}{\partial \sigma^2} = \frac{\partial U(\sigma)}{\partial \sigma} = -\frac{2n}{\sigma^2}$$
$$\frac{\partial^2 l(\mu,\sigma)}{\partial \mu \partial \sigma} = \frac{\partial U(\sigma)}{\partial \sigma} = -\frac{2}{\sigma^3}\sum_{i=1}^{n}(y_i - \overline{y}) = 0.$$

Logo,

$$I_O(\hat{\mu},\hat{\sigma}) = \begin{bmatrix} \frac{n}{\hat{\sigma}^2} & 0 \\ 0 & \frac{2n}{\hat{\sigma}^2} \end{bmatrix}.$$

Neste caso a matriz de informação observada coincide com a matriz de informação esperada. Além disso, note a importante propriedade de ortogonalidade entre os parâmetros, indicada pelos termos fora da diagonal da matriz de informação serem zero. A derivada cruzada entre dois parâmetros ser zero, é condição suficiente para que estes parâmetros sejam ortogonais. A ortogonalidade é uma propriedade muito conveniente e simplifica as inferências, uma vez que podemos fazer inferência para um parâmetro sem nos preocupar com os valores do outro.

Para construção dos intervalos de confiança, a maneira mais direta é usar os resultados assintóticos dos estimadores de máxima verossimilhança, neste caso temos que a distribuição assintótica de $\underline{\hat{\theta}} = (\hat{\mu},\hat{\sigma})^\top$ é

$$\begin{bmatrix} \hat{\mu} \\ \hat{\sigma} \end{bmatrix} \sim NM_2\left(\begin{bmatrix} \mu \\ \sigma \end{bmatrix}, \begin{bmatrix} \hat{\sigma}^2/n & 0 \\ 0 & \hat{\sigma}^2/2n \end{bmatrix}\right)$$

Intervalos de confiança de Wald podem ser obtidos por:

$$\hat{\mu} \pm z_{\alpha/2}\sqrt{\hat{\sigma}^2/n}$$

e para σ temos

$$\hat{\sigma} \pm z_{\alpha/2}\sqrt{\hat{\sigma}^2/2n}.$$

A função de verossimilhança deste exemplo é simétrica e quadrática na direção de μ, e portanto a aproximação é exata. Porém a verossimilhança é assimétrica na direção de σ. Destacamos ainda que a assimetria é maior σ^2, um pouco menos acentuada em σ e ainda menos acentuada em $\log(\sigma)$. Nos remetemos a discussão na Sessão 1.8 para mais detalhes e consequências. Na prática, se intervalos baseados na aproximação quadrática serão utilizados o mais recomendado então é parametrizar a função para uma forma mais próxima a simetria com $\psi = \log(\sigma)$, obter intervalos para ψ e

depois transformá-los para escala original do parâmetro, por transformação direta se verossimilhança em ψ for muito próxima à simetria ou caso contrário pelo método delta.

Outra opção é obter uma região de confiança baseada na *deviance*,

$$\begin{aligned} D(\mu,\sigma) &= 2[l(\hat{\mu},\hat{\sigma}) - l(\mu,\sigma)] \\ &= 2[n\log\left(\frac{\sigma}{\hat{\sigma}}\right) + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2 - \frac{1}{2\hat{\sigma}^2}\sum_{i=1}^{n}(y_i-\hat{\mu})]. \end{aligned}$$

A *deviance* aproximada tem a seguinte forma

$$D(\mu,\sigma) \approx (\underline{\theta}-\underline{\hat{\theta}})^\top I_o(\underline{\hat{\theta}})(\underline{\theta}-\underline{\hat{\theta}}).$$

Note que neste caso a superfície de log-verossimilhança em duas dimensões esta sendo aproximada por uma elipse. É bastante intuitivo pensar que aproximar uma função em duas dimensões é mais difícil que em uma, sabemos também que esta aproximação será tanto melhor quanto maior for o tamanho da amostra. Para exemplificar esta ideia a Figura 1.8 apresenta o gráfico da função *deviance* bidimensional em (μ,σ) para o caso do modelo gaussiano, de acordo com quatro tamanhos de amostra.

Na Figura 1.8 vemos que com $n = 10$ a verossimilhança exibe forte assimetria na direção do parâmetro σ e a aproximação quadrática é claramente insatisfatória. Com o aumento do tamanho da amostra a aproximação quadrática vai ficando cada vez mais próxima da *deviance* exata, mais uma vez ilustrando o comportamento assintótico da verossimilhança. É importante notar também em modelos com dois ou mais parâmetros a aproximação pode melhorar mais rapidamente para um do que outro. No exemplo a aproximação é exata para μ para qualquer tamanho de amostra. Já para σ é necessário um tamanho de amostra relativamente grande para que a *deviance* em sua direção tome um comportamento próximo do simétrico. A função se aproxima da simetria mais rapidamente se parametrizada com $\log(\sigma)$.

Em outros modelo, como no caso dos MLG a intuição é a mesma ainda que a aproximação para parâmetros de média deixe de ser exata. De forma geral, para parâmetros de média a aproximação quadrática tende a apresentar bons resultados mesmo com amostras reduzidas. O mesmo não pode ser dito para parâmetros de dispersão ou mesmo de correlação. Em outra palavras, estimar média é mais simples que estimar variabilidade, que por sua vez é mais simples do que estimar correlações.

As regiões de confiança são as curvas de nível na superfície de *deviance*. Note que apenas com a *deviance* não temos intervalos marginais como os obtidos pela aproximação quadrática. Uma possível solução é projetar a superfície na direção do parâmetro de interesse na maior amplitude, que

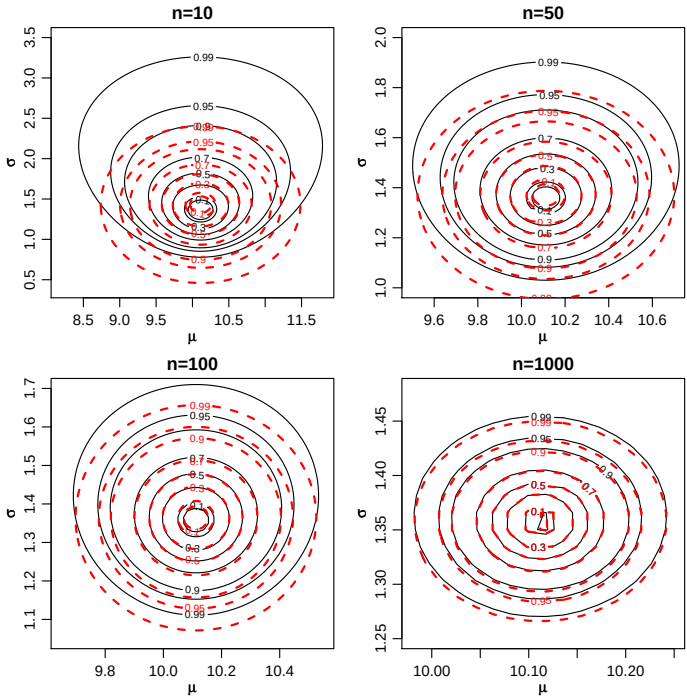

Figura 1.8: Deviance exata (linha sólida) e aproximada (linha tracejada) para diferentes tamanhos de amostra - Distribuição Normal.

é obtida quando fixamos o outro parâmetro na sua estimativa de máxima verossimilhança. Porém esta prática só produz bons resultados quando os parâmetros são ao menos aproximadamente ortogonais. Uma solução mais genérica, ainda que computacionalmente mais trabalhosa é o obtenção das verossimilhanças perfilhadas. No caso particular da distribuição gaussiana, que tem a propriedade de ortogonalidade entre μ e σ, a verossimilhança condicionada na estimativa de máxima verossimilhança coincide com a verossimilhança perfilhada. Para ilustrar este fato considere a obtenção da verossimilhança perfilhada para μ e σ pelas funções a seguir:

Código 1.11: Função para log-verossimilhança perfilhada em relação aos parâmetros μ e σ da distribuição gaussiana.

```
> ## Perfil para mu
> perf.mu <- function(sigma, mu, dados){
+ ll <- sum(dnorm(dados,mean=mu,sd=sigma,log=TRUE))
+ return(ll)}
> ## Perfil para sigma
> perf.sigma <- function(mu, sigma, dados){
+ ll <- sum(dnorm(dados,mean=mu,sd=sigma,log=TRUE))
+ return(ll)}
```

Vamos criar uma malha de valores para avaliação da função e posterior construção dos gráficos. Também vamos obter a log-verossimilhança condicionada na estimativa de máxima verossimilhança, que consiste em apenas avaliar a função para um dos parâmetros com o outro fixado em sua estimativa.

```
> grid.mu <- seq(9,11,l=100)
> grid.sigma <- seq(1,2,l=100)
> ## Condicional para mu
> vero.cond.mu <- sapply(grid.mu,perf.sigma,sigma=sd(y50),dados=y50)
> ## Condicional para sigma
> vero.cond.sigma<-sapply(grid.sigma,perf.mu,mu=mean(y50),dados=y50)
```

Para obter o perfil de verossimilhança, por exemplo para σ precisamos de uma grade de valores de σ e para cada valor nesta grade encontrar o valor digamos $\hat{\mu}_\sigma$ que maximiza a verossimilhança perfilhada. Para evitar a obtenção de muitas equações vamos ilustrar o procedimento numericamente. Para a maximização usamos a função `optimize()` própria para maximização em apenas uma dimensão como é o caso neste exemplo. O código abaixo ilustra o procedimento:

```
> vero.perf.mu <- c()
> for(i in 1:length(grid.mu)){
+ vero.perf.mu[i] <- optimize(perf.mu,c(0,200),
+               mu=grid.mu[i],dados=y50,maximum=TRUE)$objective}
> vero.perf.sigma <- c()
> for(i in 1:length(grid.sigma)){
+ vero.perf.sigma[i] <- optimize(perf.sigma,c(0,1000),
+           sigma=grid.sigma[i],dados=y50,maximum=TRUE)$objective}
```

Os gráficos da verossimilhança condicionada no MLE e perfilhada são sobrepostos na Figura 1.10.

A Figura 1.10 ilustra que a *deviance* perfilhada e a *deviance* condicional coincidem neste caso devido a ortogonalidade. Para obter intervalos de confiança basta definir o corte nestas funções seja por valor relativo da verossimilhança ou usando o quantil da distribuição χ^2 para o nível de confiança desejado e encontrar as raízes da equação, assim como nos exemplos uniparamétricos. A verossimilhança perfilhada permite tratar um pro-

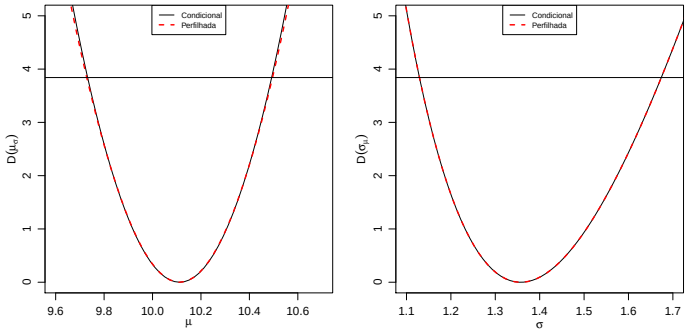

Figura 1.9: Deviance perfilhada e condicional para μ e σ - Distribuição Normal.

blema multiparamétrico como um problema uniparamétrico levando em consideração a forma da verossimilhança na direção de todos os parâmetros do modelo. Porém, esta abordagem pode ser extremamente cara computacionalmente, uma vez que para cada avaliação da verossimilhança perfilhada pode ser necessário uma maximização, que em geral vai requerer algum método numérico.

1.9.1 Dados intervalares

Quando definimos a função de verossimilhança no início deste capítulo, mencionamos que dados são medidos em algum intervalo definido pela precisão da medição. No exemplo anterior fizemos a suposição usual de que este intervalo é pequeno em relação a variação dos dados e portanto os valores dos dados são tratados como pontos na cálculo da função de verossimilhança e utilizamos 1.2

Vamos considerar agora a situação na qual os dados são medidos em intervalos *não desprezíveis*. Desta forma voltamos a definição mais geral da verossimilhança em 1.1 para obter sua expressão.

Como exemplo vamos considerar a distribuição gaussiana $Y_i \sim N(\mu, \sigma^2)$, $\underline{\theta} = (\mu, \sigma)$. Suponha que temos um conjunto de dados que consiste de:

observações "pontuais": `72,6  81,3  72,4  86,4  79,2  76,7  81,3;`

observações intervalares:

uma observação com valor acima de 85,

uma observação com valor acima de 80,

quatro observações com valores entre 75 e 80,

seis observações com valores abaixo de 75.

Supondo independência, a contribuição para verossimilhança das observações pontuais é o valor da densidade no ponto, enquanto que para as intervalares é a probabilidade da observação estar no intervalo. Para os tipos de dados neste exemplo temos:

$$
\begin{aligned}
L(\underline{\theta}) &= f(y_i) \text{ para } y_i \text{ pontual,} \\
L(\underline{\theta}) &= 1 - F(85) \text{ para } y_i > 85, \\
L(\underline{\theta}) &= 1 - F(80) \text{ para } y_i > 80, \\
L(\underline{\theta}) &= F(80) - F(75) \text{ para } 75 < y_i < 80, \\
L(\underline{\theta}) &= F(75) \text{ para } y_i < 85.
\end{aligned}
$$

A seguir escrevemos a função de (negativo da) verossimilhança que recebe como argumentos os parâmetros, os dados pontuais como um vetor e os intervalares como uma matriz de duas colunas na qual cada linha corresponde a um dado.

Código 1.12: Função para log-verossimilhança para dados pontuais e intervalares de distribuição gaussiana.

```
> nllnormI <- function(par, xp, XI) {
+     ll1 <- sum(dnorm(xp, mean = par[1], sd = par[2], log = T))
+     L2 <- pnorm(XI, mean = par[1], sd = par[2])
+     ll2 <- sum(log(L2[, 2] - L2[, 1]))
+     return(-(ll1 + ll2))
+ }
```

Nos comandos a seguir definimos os objetos que contém os dados. A matriz dos dados intervalares é transposta apenas para visualização Usamos estimativas baseadas nos dados completos como valores iniciais e encontramos as estimativas usando todos os dados maximizando e função de verossimilhança numericamente.

```
> y <- c(72.6, 81.3, 72.4, 86.4, 79.2, 76.7, 81.3)
> t(yI <- cbind(c(85, 80, rep(75, 4), rep(-Inf, 6)),
+               c(rep(Inf, 2), rep(80, 4), rep(75, 6))))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[1,]   85   80   75   75   75   75 -Inf -Inf -Inf  -Inf  -Inf  -Inf
[2,]  Inf  Inf   80   80   80   80   75   75   75    75    75    75

> (ini <- c(mean(y), sd(y)))

[1] 78.557143  5.063219

> (ests <- optim(ini, nllnormI, x=y, XI=yI)$par)

[1] 76.67196  5.71692
```

Quando possível, é mais conveniente fazer o gráfico das superfícies de verossimilhança na escala da *deviance* que requer o valor da verossmimilhança avaliado nas estimativas dos parâmetros. Vamos utilizar a função *deviance* genérica definida em **??** que pode ser usada com outras densidades com dois parâmetros. Por conveniência definimos também a função em forma vetorizada que utilizaremos com o comando `outer()` na obtenção das superfícies.

Código 1.13: Função deviance genérica.

```
> devFun <- function(theta, est, llFUN, ...){
+   return(2 * (llFUN(theta, ...) - llFUN(est, ...)))
+ }
> devSurf <- Vectorize(function(x,y, ...) devFun(c(x,y), ...))
```

O gráfico à esquerda de Figura **??** mostra superfícies de verossimilhança na escala da *deviance* e é obtido com os comandos a seguir. O gráfico da direita usa a parametrização $\log(\sigma)$. O aspecto talvez mais importante é notar que, diferentemente dos gráficos 1.8, com dados intervalares a superfície não mais exibe ortogonalidade entre os parâmetros.

```
> mu <- seq(70,82, l=50)
> sigma <- seq(3, 14, l=50)
> devMS <- outer(mu, sigma, FUN=devSurf, llFUN=nllnormI,
+                est=ests, xp=y, XI=yI)
> LEVELS <- c(0.99,0.95,0.9,0.7,0.5,0.3,0.1,0.05)
> contour(mu, sigma, devMS, levels=qchisq(LEVELS,df=2),
+         labels=LEVELS, xlab=expression(mu),ylab=expression(sigma))
> points(t(ests), pch=19, col=2, cex=0.7)
```

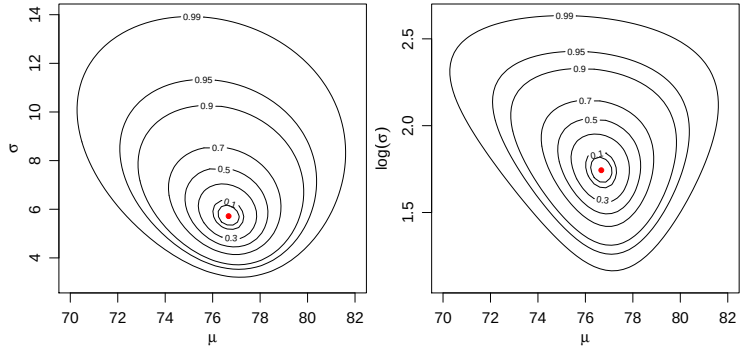

Figura 1.10: Deviance perfilhada e condicional para μ e σ - Distribuição Normal.

No código a seguir redefinimos a função de verossimilhança anterior

acrescentando alguns elementos. Colocamos uma opção para parametrização usando $\log(\sigma)$ através do argumento `logsigma`. Comandos para verificar se argumentos de dados foram informados permitem rodar a função mesmo sem dados pontuais ou intervalares. Finalmente verificamos internamente se a matriz de dados intervalares está especificada corretamente.

Código 1.14: Redefinição da função para log-verossimilhança para dados pontuais e intervalares de distribuição gaussiana.

```
> nllnormI <- function(par, xp, XI, logsigma=FALSE){
+     if(logsigma) par[2] <- exp(par[2])
+         ll1 <- ifelse(missing(xp), 0,
+                     sum(dnorm(xp, mean=par[1], sd=par[2], log=T)))
+     if(missing(XI)) ll2 <- 0
+     else{
+       if(ncol(XI) != 2 || any(XI[,2] <= XI[,1]))
+        stop("XI deve ser matrix com 2 colunas com XI[,2] > XI[,2]")
+       L2 <- pnorm(XI, mean=par[1], sd=par[2])
+             ll2 <- sum(log(L2[,2] - L2[,1]))
+   }
+   return(-(ll1 + ll2))
+ }
```

1.10 Exemplo 6 - Distribuição Gama

Sejam $Y_1, Y_2, \ldots, Y_n$ variáveis aleatórias independentes com distribuição Gama de parâmetros a e s. Nosso objetivo partindo de uma amostra aleatória $y_1, y_2, \ldots y_n$ é fazer inferências sobre os seus dois parâmetros com seus respectivos intervalos de confiança baseados na aproximação quadrática e na função *deviance*. A função de densidade da Gama pode ser escrita na seguinte forma:

$$f(y) = \frac{1}{s^a \Gamma(a)} y^{a-1} \exp\{-y/s\}, \quad \text{para} \quad y \geq 0 \quad \text{e} \quad a,s \geq 0.$$

Nesta parametrização $E(Y) = a * s$ e $V(Y) = a * s^2$. A função de verossimilhança é

$$\begin{aligned} L(a,s) &= \prod_{i=1}^{n} (s^a \Gamma(a))^{-1} y_i^{a-1} \exp^{-y_i/s} \\ &= s^{-na} \Gamma^{-n}(a) \exp^{-\sum_{i=1}^{n} y_i/s} \prod_{i=1}^{n} y_i^{a-1}. \end{aligned}$$

Esta parametrização da função gama é comumente encontrada, entretanto não é a mais conveniente para cálculos numéricos pois os parâmetros

não são ortogonais. Vamos explorar estes fatos seguindo com esta parametrização para ilustrar o comportamento da verossimilhança. Ao final passamos a uma forma reparametrizada mais conveniente para implementação de algoritmos.

A função de log-verossimilhança é dada por

$$l(a,s) = -na\log s - n\log\Gamma(a) - \frac{1}{s}\sum_{i=1}^{n} y_i + (a-1)\sum_{i=1}^{n}\log y_i.$$

As funções *escore* são obtidas derivando a log-verossimilhança em função de cada um dos respectivos parâmetros,

$$U(a) = -n\log(s) - n\frac{\Gamma'(a)}{\Gamma(a)} + \sum_{i=1}^{n}\log y_i$$

$$U(s) = -\frac{na}{s} + \frac{1}{s^2}\sum_{i=1}^{n} y_i.$$

Para obter as estimativas de máxima verossimilhança igualamos essas expressões a zero e resolvemos em relação a e s o sistema de equações:

$$\begin{cases} \log(s) + \Psi(a) & = \frac{1}{n}\sum_{i=1}^{n}\log y_i \\ a \cdot s & = \overline{y} \end{cases}$$

em que $\Psi(\cdot)$ é a função digama (`digamma()` no R) definida por $\Psi(x) = \frac{d}{dx} = \Gamma(x)/\Gamma'(x)$. Este sistema claramente não tem solução analítica em a, porém para s obtemos

$$\hat{s} = \frac{\overline{y}}{a}. \tag{1.5}$$

Substituindo $\hat{s}$ na função de log-verossimilhança, obtemos o que chamamos de log-verossimilhança concentrada em a com a expressão e escore dados por:

$$l_s(a) = -na\log\frac{\overline{y}}{a} - n\log\Gamma(a) - \frac{a}{\overline{y}}\sum_{i=1}^{n} y_i + a\sum_{i=1}^{n}\log y_i - \sum_{i=1}^{n}\log y_i \tag{1.6}$$

$$U_s(a) = -n\log(\overline{y}/a) - n\frac{\Gamma'(a)}{\Gamma(a)} + \sum_{i=1}^{n}\log y_i \tag{1.7}$$

que são funções apenas do parâmetro a. Com a verossimilhança concentrada reduzimos o problema original de maximização em duas dimensões, a uma maximação para apenas uma dimensão, o que é mais eficiente e estável computacionalmente. Para encontrar a estimativa de a ainda precisamos maximizar a log-verossimilhança concentrada numericamente. Para

isto temos diferentes alternativas. Podemos usar um otimizador numérico como o implementado na função `optimize()` (para um parâmetro) ou alguns dos métodos da função `optim()` (para dois ou mais parâmetros) para maximizar 1.6. Alternativamente, podemos obter a estimativa igualando a equacão 1.7 a zero, e resolvendo numericamente, por exemplo com a função `uniroot()` do R. O pacote **rootSolve** implementa algoritmos adicionais incluindo a definição e solução de sistemas de equações.

Mas antes disso, vamos investigar a ortogonalidade entre a e s. Para isto, precisamos obter a matriz de segundas derivadas, neste caso de dimensão 2×2, que fornece a matriz de informação observado e/ou esperada.

Derivando as funções escore temos,

$$\begin{aligned}
\frac{\partial^2 l(a,s)}{\partial a^2} &= -n\left[\frac{\Gamma(a)\Gamma''(a) - \Gamma'(a)^2}{\Gamma(a)^2}\right] \\
\frac{\partial^2 l(a,s)}{\partial s^2} &= \frac{na}{s^2} - \frac{2}{s^3}\sum_{i=1}^{n} y_i \\
\frac{\partial^2 l(a,s)}{\partial a \partial s} &= -\frac{n}{s}.
\end{aligned}$$

Logo, a matriz de informação observada é dada por,

$$I_o(a,s) = \begin{bmatrix} n\left[\frac{\Gamma''(a)}{\Gamma'(a)} - \left(\frac{\Gamma'(a)}{\Gamma(a)}\right)^2\right] & \frac{n}{s} \\ \frac{n}{s} & -na/s^2 + 2/s^3\sum_{i=1}^{n} y_i \end{bmatrix}.$$

A matriz esperada é obtida tomando a esperança da matriz observada, lembrando que $E(Y) = a * s$ temos

$$I_E(a,s) = \begin{bmatrix} n\left[\frac{\Gamma''(a)}{\Gamma(a)} - \left(\frac{\Gamma'(a)}{\Gamma(a)}\right)^2\right] & \frac{n}{s} \\ \frac{n}{s} & \frac{na}{s^2}. \end{bmatrix}.$$

O termos fora da diagonal são não-nulos o que mostra que os parâmetros são não ortogonais. Para visualizarmos o formato da função de log-verossimilhança a Figura 1.11 apresenta a superfície de log-verossimilhança e sua aproximação quadrática em escala de *deviance* para facilitar a construção e visualização do gráfico.

Pelos gráficos podemos ver claramente que quando o tamanho da amostra é pequeno $n = 10$ o formato da log-verossimilhança é extremamente assimétrico e consequentemente a aproximação quadrática é muito ruim. Com o aumento da amostra a aproximação quadrática vai melhorando, até que com $n = 2000$ a diferença é bastante pequena. Os gráficos também mostram a dependência entre os parâmetros a e s, quando o a aumenta necessariamente o s diminui para manter a média que é $a * s$, além disso fica

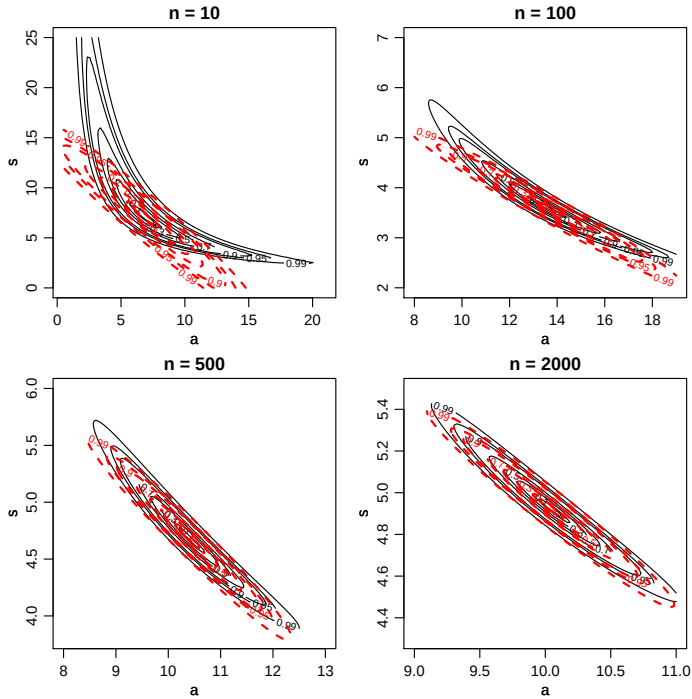

Figura 1.11: Deviance exata e aproximada por tamanho de amostra - Distribuição Gama.

claro também que a incerteza associada a estimativa de a é muito maior quando comparada a estimativa de s.

Agora retornamos à obtenção das estimativas de máxima verossimilhança. Lembrando que a log-verossimilhança concentrada 1.6 é uma função apenas do parâmetro a, ama vez obtido a estimativa $\hat{a}$ podemos substitui-la em $\hat{s} = \frac{\overline{y}}{\hat{a}}$ e obter a estimativa de s. Da mesma forma podemos substituir as estimativas nas matrizes de informação observada e esperada e encontrar intervalos de confiança assintóticos, sabendo que estes intervalos serão consistentes apenas com amostras grandes. Mas para todos estes procedimentos precisamos maximizar a log-verossimilhança concentrada em a. A forma mais comum de fazer isso é usando o algoritmo de Newton-Raphson que utiliza a informação observada, ou uma variante deste chamada de algoritmo Escore de Fisher que substitui a informação observada pela esperada. Vamos abrir um parenteses e explicar rapidamente o algoritmo de Newton-Raphson.

O método de Newton-Raphson é usado para se obter a solução numérica de uma equação na forma $f(x) = 0$, onde $f(x)$ é contínua e diferen-

ciável e sua equação possui uma solução próxima a um ponto dado. O processo de solução começa com a escolha do ponto x_1 como a primeira tentativa de solução. A segunda tentativa, x_2, é obtida a partir do cruzamento com o eixo x da reta tangente a $f(x)$ no ponto $(x_1, f(x_1))$. A tentativa seguinte, x_3 é a intersecção com o eixo x da reta tangente a $f(x)$ no ponto $(x_2, f(x_2))$, e assim por diante. A equação de iteração é dada por:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{1.8}$$

ela é chamada de equação de iteração porque a solução é obtida com a aplicação repetida em cada valor sucessivo de i até que um critério de convergência seja atingido. Diversos critérios de convergência podem ser usados. Os mais comuns são:

- Erro relativo
$$\left| \frac{x_{i+1} - x_i}{x_i} \right| \leq \epsilon$$
- Tolerância em $f(x)$
$$|f(x_i)| \leq \delta$$

Uma função chamada `NewtonRaphson()` é definida em 1.15.

Código 1.15: Algoritmo genérico de Newton-Raphson.

```
> NewtonRaphson <- function(initial, escore, hessiano, tol=0.0001,
+                   max.iter, n.dim, print=TRUE, ...){
+   solucao <- initial
+   for(i in 2:max.iter){
+     solucao <- initial - solve(hessiano(initial, ...),
+                                escore(initial, ...))
+     tolera <- abs(solucao - initial)
+     if(all(tolera < tol) == TRUE)break
+     initial <- solucao
+   }
+   if(print) print(initial)
+   return(initial)
+ }
```

Note que para usar este algoritmo é necessário obter a primeira (escore) e a segunda (hessiano) derivada. Neste exemplo é possível obter expressões analíticas para ambas. Em modelos mais complexos expressões analíticas podem ser substituídas por gradientes e hessianos obtidos por algoritmos numéricos. Além disto, em certos casos o custo computacional em calcular o hessiano analítico pode ser muito maior que o numérico, o que acontece

em alguns modelos multivariados que em geral envolvem muitas inversões de matrizes densa, fazendo com que este algoritmo se torne muito lento.

Cabe ressaltar que o método de Newton-Raphson é um algoritmo para encontrar raízes de uma equação que no caso da função escore leva as estimativas de máxima verossimilhança. Porém existem diversos e poderosos algoritmos de maximização numérica que não exigem derivadas analíticas embora possam ser beneficiados com o uso de resultados destas principalmente a função escore. No R alguns destes maximizadores numéricos estão implementados na função `optim()`.

Continuando com o exemplo da Gama, vamos obter a função escore e o hessiano da função de log-verossimilhança concentrada e usar o algoritmo de Newton-Raphson para obter a estimativa de a. Derivando a log-verossimilhança em a obtemos,

$$U_c(a) = \sum_{i=1}^{n} \log y_i - n \log \frac{\overline{y}}{a} - n\psi_0(a) + \sum_{i=1}^{n} \log y_i$$

onde $\psi_0(a)$ é a função digama que é a derivada primeira do logaritmo da função gama. Derivando novamente em a obtemos,

$$U_c'(a) = \frac{n}{a} - n\psi_1(a)$$

onde $\psi_1(a)$ é a função trigama que é a derivada segunda do logaritmo da função gama.

Escrevendo estas funções em R temos o código 1.16.

Código 1.16: Função escore e hessiana ambas em relação ao parâmetro a da distribuição Gama.

```
> escore <- function(a,y){
+   n <- length(y)
+   u <- -n*log(mean(y)/a) - n*digamma(a) + sum(log(y))
+   return(u)}
> hessiano <- function(a,y){
+   n <- length(y)
+   u.l <- (n/a)-trigamma(a)*n
+   return(u.l)}
```

O passo final para obter a estimativa de máxima verossimilhança de a é usar o algoritmo de Newton-Raphson.

```
> (a.hat <- NewtonRaphson(initial=5,escore= escore ,
+         hessiano= hessiano, max.iter=100, n.dim=1, y=y10))
[1] 13.53677
[1] 13.53677
```

Definindo o argumento `print=TRUE` é possível visualizar todas as tentativas do algoritmo até a convergência. Neste exemplo foi necessário seis iterações para atingir o critério de convergência. Uma limitação deste algoritmo é que o chute inicial não pode estar muito longe da solução, o que pode ser difícil de obter em modelos complexos, nestes casos estimativas grosseiras por mínimos quadrados podem ser de grande valia como valores inicial.

Uma vez estimado o a podemos substituir na expressão de $\hat{s}$ para obtê-lo,

```
> (s.hat <- mean(y10)/a.hat)
[1] 3.614119
```

Para construção dos intervalos assintóticos basta substituir as estimativas nas matrizes de informação observada e/ou esperada. Note que no caso da distribuição Gama a distribuição assintótica do vetor $(\hat{a},\hat{s})^\top$ é a seguinte,

$$\begin{bmatrix}\hat{a}\\ \hat{s}\end{bmatrix} \sim NM_2\left(\begin{bmatrix}a\\ s\end{bmatrix}, \begin{bmatrix}n\psi_1(\hat{a}) & n/\hat{s}\\ n/\hat{s} & n\hat{a}/\hat{s}^2\end{bmatrix}^{-1}\right)$$

poderíamos usar também a matriz de informação observada que é assintoticamente equivalente. O código 1.17 implementa da matriz de informação esperada e observada e constrói os intervalos de confiança assintóticos para a e s.

Código 1.17: Funções para a matriz de informação esperada e informação observada da distribuição Gama.

```
> Ie <- function(a,s,y){
+   n <- length(y)
+   saida <- matrix(c(n*trigamma(a),n/s,
+                     n/s, (n*a)/s^2),2,2)
+   return(saida)}
> Io <- function(a,s,y){
+   n <- length(y)
+   saida <- matrix(c(n*trigamma(a), n/s, n/s,
+                     -(n*a)/s^2 + (2/s^3)*sum(y)),2,2)
+   return(saida)}
```

Avaliando as matrizes no ponto de máximo,

```
> Ie(a = a.hat, s = s.hat, y=y10)
         [,1]     [,2]
[1,]  7.666858  27.66926
[2,] 27.669258 103.63586
> Io(a = a.hat, s = s.hat, y=y10)
         [,1]     [,2]
[1,]  7.666858  27.66926
[2,] 27.669258 103.63586
```

Como é possível observar a matriz de informação esperada e observada apesar de apresentarem formas diferentes levam a resultados idênticos para o tamanho de amostra $n = 100$ considerado aqui. Sendo assim, vamos usar apenas a informação esperada que é mais simples de avaliar. Para obter os intervalos assintóticos basta inverter a matriz de informação e pegar os termos em sua diagonal que corresponde a variância de $\hat{a}$ e $\hat{s}$.

```
> erro.padrao <- sqrt(diag(solve(Ie(a = a.hat, s= s.hat, y=y10))))
> ic.a <- a.hat + c(-1,1)*qnorm(0.975)*erro.padrao[1]
> ic.s <- s.hat + c(-1,1)*qnorm(0.975)*erro.padrao[2]
> ic.a
[1]  9.82995 17.24359
> ic.s
[1] 2.605900 4.622339
```

Outra forma é a construção de intervalos baseados no perfil de verossimilhança. As funções em 1.18 implementam o perfil de verossimilhança para os parâmetros a e s respectivamente.

Código 1.18: Funções para a log-verossimilhança perfilhada em relação aos parâmetros a e s da distribuição Gama.

```
> ## Perfil para a
> perf.a <- function(s, a, dados){
+ ll <- sum(dgamma(dados,shape=a , scale=s, log=TRUE))
+ return(ll)}
> ## Perfil para s
> perf.s <- function(a, s, dados){
+ ll <- sum(dgamma(dados,shape=a,scale=s,log=TRUE))
+ return(ll)}
```

Para as maximizações necessárias vamos utilizar a função `optimize()` própria para maximização em uma dimensão como é o caso aqui. Precisamos também criar uma grade para a avaliação da função. Também será avaliada a verossimilhança condicional para comparação conforme realizado no Exemplo 1.9. A Figura 1.12 apresenta os resultados.

```
> grid.a <- seq(9,18,l=100)
> grid.s <- seq(2,5,l=100)
> ## Perfil para a
> vero.perf.a <- c()
> for(i in 1:length(grid.a)){
+ vero.perf.a[i] <- optimize(perf.a,c(0,200),
+                   a=grid.a[i],dados=y10,maximum=TRUE)$objective}
> ## Perfil para s
> vero.perf.s <- c()
> for(i in 1:length(grid.s)){
+ vero.perf.s[i] <- optimize(perf.s,c(0,1000),
+                   s=grid.s[i],dados=y10,maximum=TRUE)$objective}
```

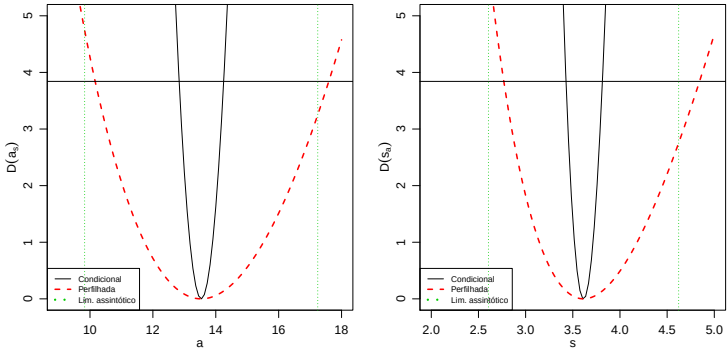

Figura 1.12: Deviance perfilhada, condicional e limites do intervalo de confiança assintótico para a e s - Distribuição Gama.

```
> ## Condicional para a
> vero.cond.a <- sapply(grid.a,perf.a,s=s.hat,dados=y10)
> ## Condicional para sigma
> vero.cond.s <- sapply(grid.s, perf.s, a= a.hat , dados=y10)
```

Como mostra a Figura 1.12 os intervalos obtidos condicionando a log-verossimilhança na estimativa de máxima verossimilhança são claramente mais curtos que o intervalo baseado em perfil de verossimilhança e o intervalo assintótico. Comparando o perfil com o assintótico verificamos que a perfilhada traz intervalos ligeiramente assimétricos e mais largos que a aproximação quadrática, a aproximação é ligeiramente deslocada para esquerda e para direita para os parâmetros a e s respectivamente.

1.10.1 Parametrizações para Gama

Vamos explorar este exemplo para ilustrar o efeito de diferentes parametrizações no formado da verossimilhança da densidade Gama. Aproveitamos ainda esta sessão para ilustrar o uso de algumas sintaxes e formas de programação em R.

Parametrização 1: Esta é a parametrização utilizada anteriormente e também a usada nas funções `*gamma` do R. O parâmetro de forma (*shape*) é $\alpha = a$ e o de escala (*scale*) $\beta = s$. Lembrando as expressões obtidas anteriormente

temos:

$$
\begin{aligned}
Y &\sim G(\alpha,\beta)\\
f(y|\alpha,\beta) &= \frac{1}{\Gamma(\alpha)\,\beta^{\alpha}}\; y^{\alpha-1}\;\exp\{-y/\beta\} \quad y\geq 0\;\; \alpha\geq 0\;\; \beta>0\\
E[Y] &= \alpha\beta \quad Var[Y]=\alpha\beta^2\\
L((\alpha,\beta)|y) &= \left(\frac{1}{\Gamma(\alpha)\,\beta^{\alpha}}\right)^{n}\;\prod_{i=1}^{n} y_i^{\alpha-1}\;\exp\{-y_i/\beta\}\\
l((\alpha,\beta)|y) &= n\left(-\log(\Gamma(\alpha))-\alpha\log(\beta)+(\alpha-1)\overline{\log(y)}-\bar{y}/\beta\right)
\end{aligned}
$$

A função escore é escrita como:

$$
\begin{cases}
\frac{dl}{d\beta} &= n\left(-\frac{\alpha}{\beta}+\frac{\bar{y}}{\beta^2}\right)\\
\frac{dl}{d\alpha} &= n\left(-\frac{\Gamma'(\alpha)}{\Gamma(\alpha)}-\log(\beta)+\overline{\log y}\right)
\end{cases}
$$

Igualando as funções a zero, da primeira equação temos $\hat{\alpha}\hat{\beta}=\bar{y}$. Substituindo β por $\hat{\beta}$ a segunda expressão é escrita como:

$$
n\left(-\frac{\Gamma'(\alpha)}{\Gamma(\alpha)}-\log\left(\frac{\bar{y}}{\hat{\alpha}}\right)+\overline{\log y}\right)=0
$$

O EMV é portanto solução conjunta de

$$
\begin{cases}
\overline{\log y}-\log\beta+ &= \psi(\bar{y}/\beta)\\
\hat{\alpha}\hat{\beta} &= \bar{y}
\end{cases}
$$

em que $\psi(t)=\frac{d}{dt}\log(\Gamma(t))=\frac{\Gamma'(t)}{\Gamma(t)}$ (função `digamma()`no R) e $\overline{\log y}=\sum_{i=1}^{n}\log(y_i)/n$.

Parametrização 2: Esta parametrização utilizada por Rizzo (2008), dentre outros autores, é parametrização original usada na linguagem § e sua primeira implementação o programa S-PLUS. [1]. Neste caso o parâmetro de escala é trocado pela seu inverso, a taxa (*rate*) e denotamos $\lambda=1/\beta$. No R pode-se definir a escala ou taxa.

```
> args(rgamma)
function (n, shape, rate = 1, scale = 1/rate)
NULL
```

[1] S-PLUS e R são duas implementações em *software*, não completamente distintas, da linguagem S.

As expressões ficam então:

$$
\begin{aligned}
Y &\sim \mathrm{G}(\alpha,\lambda)\\
f(y|\alpha,\lambda) &= \frac{\lambda^{\alpha}}{\Gamma(\alpha)}\, y^{\alpha-1}\ \exp\{-\lambda y\} \quad y \geq 0\ \ \alpha \geq 0\ \ \lambda > 0\\
E[Y] &= \alpha/\lambda \quad Var[Y] = \alpha/\lambda^2\\
L((\alpha,\lambda)|y) &= \left(\frac{\lambda^{\alpha}}{\Gamma(\alpha)}\right)^{n} \prod_{i=1}^{n} y_i^{\alpha-1}\ \exp\{-\lambda y_i\}\\
l((\alpha,\lambda)|y) &= n\left(\alpha\log(\lambda) - \log(\Gamma(\alpha)) + (\alpha-1)\overline{\log(y)} - \lambda\bar{y}\right)
\end{aligned}
$$

Função escore:

$$
\left\{
\begin{array}{ll}
\frac{dl}{d\lambda} & = n\left(\frac{\alpha}{\lambda} - \bar{y}\right)\\
\frac{dl}{d\alpha} & = n\left(\log(\lambda) - \frac{\Gamma'(\alpha)}{\Gamma(\alpha)} + \overline{\log y}\right)
\end{array}
\right.
$$

Igualando as funções a zero, da primeira equação temos $\hat{\lambda} = \hat{\alpha}/\bar{y}$. Substituindo λ por $\hat{\lambda}$ a segunda expressão é escrita como:

$$
n\left(\log\left(\frac{\hat{\alpha}}{\bar{y}}\right) + \overline{\log y} - \frac{\Gamma'(\alpha)}{\Gamma(\alpha)}\right) = 0
$$

O EMV é solução conjunta de

$$
\left\{
\begin{array}{ll}
\log\lambda + \overline{\log y} & = \psi(\lambda\bar{y})\\
\bar{y} & = \alpha/\lambda
\end{array}
\right.
$$

Parametrização 3: `aitkin:2010` menciona ainda duas parametrizações sendo a primeira, a mesma do R, citada como a mais usual e uma segunda parametrizada por α e $\mu = \alpha\beta$.

Esta parametrização tem propriedades interessantes para inferência. A primeira é a ortogonalidade na matriz de informação entre r e μ . Além disto em geral μ é o usualmente o parâmetro de interesse para inferências e r é um parâmetro *nuisance*. A parametrização é adequada para modelagem estatística na qual usualmente se propõe um modelo de regressão para

média μ, como por exemplo em modelos lineares generalizados (MLG).

$$
\begin{aligned}
Y &\sim \mathrm{G}(\alpha,\mu)\\
f(y|\alpha,\mu) &= \frac{\alpha^{\alpha}}{\Gamma(\alpha)\,\mu^{\alpha}}\; y^{\alpha-1}\;\exp\{-\alpha y/\mu\} \quad y\geq 0\;\; \alpha\geq 0\;\; \mu\geq 0\\
E[Y] &= \mu \quad Var[Y] = \mu^2/\alpha\\
L((\alpha,\mu)|y) &= \left(\frac{\alpha^{\alpha}}{\Gamma(r)\mu^{\alpha}}\right)^{n}\prod_{i=1}^{n} y_i^{\alpha-1}\;\exp\{-\alpha y_i/\mu\}\\
l((\alpha,\mu)|y) &= n\left(\alpha(\log(\alpha)-\log(\mu))-\log(\Gamma(\alpha))+(\alpha-1)\overline{\log(y)}-\alpha\bar{y}/\mu\right)
\end{aligned}
$$

Função escore

$$
\begin{cases}
\frac{dl}{d\mu} &= n\left(-\frac{\alpha}{\mu}+\frac{\alpha\bar{y}}{\mu^2}\right)\\
\frac{dl}{d\alpha} &= n\left(\log(\alpha)+1-\log(\mu)-\frac{\Gamma'(\alpha)}{\Gamma(\alpha)}+\overline{\log y}+\frac{\bar{y}}{\mu}\right)
\end{cases}
$$

Igualando as funções a zero, da primeira equação temos $\hat{\mu}=\bar{y}$. Substituindo μ por $\hat{\mu}$ a segunda expressão é escrita como:

$$
n\left(\log(\hat{\alpha})+1-\log(\bar{y})-\frac{\Gamma'(\hat{\alpha})}{\Gamma(\hat{\alpha})}+\overline{\log y}+1\right)=0
$$

O EMV é solução conjunta de:

$$
\begin{cases}
\log\hat{\alpha}-\psi(\hat{\alpha}) &= \log\bar{y}-\overline{\log y}-2\\
\hat{\mu} &= \bar{y}
\end{cases}_{S}
$$

Nesta parametrização, a partir das expressões de $\frac{d^2l}{d\mu^2}$ e $\frac{d^2l}{d\alpha^2}$ obtemos que ortogonais na informação já que $I_E(\mu,\alpha)$ e $I_E(\hat{\mu},\hat{\alpha})$ são matrizes diagonais.

Para obter os gráficos de verossimilhança vamos definir uma função em R que será escrita com opção para as três parametrizações mencionadas. Em todas estas parametrizações os parâmetros são não negativos e uma transformação logarítmica fornece parâmetros com suporte na reta real, mas note que isto exclui o valor nulo do espaço paramétrico. Desta forma nossa função permite ainda reparametrizações adicionais trocando os parâmetros por seus logaritmos.

Código 1.19: Funções de verossimilhança de distribuição Gama, para diferentes parametrizações.

```
> neglLik <- function(par, amostra, modelo=2, logpar=FALSE){
+   if(logpar) par <- exp(par)
+   ll <- switch(modelo,
+     "1" = {alpha <- par[1]; beta <- par[2];
+            with(amostra, n*(-alpha*log(beta)-log(gamma(alpha)) +
+                 (alpha-1) * media.logs - media/beta))},
+     "2" = {alpha <- par[1]; lambda <- par[2] ;
+            with(amostra, n*(alpha*log(lambda)-log(gamma(alpha)) +
+                 (alpha-1) * media.logs - lambda * media))},
+     "3" = {alpha <- par[1]; mu <- par[2] ;
+            with(amostra, n*(alpha*(log(alpha) - log(mu)) -
+                 log(gamma(alpha)) +  (alpha-1) * media.logs -
+                 (alpha/mu) * media))})
+   return(-ll)
+ }
```

A função em 1.19 é escrita em termos de estatísticas (suficientes) da amostra para evitar repetições de cálculos e exige que estas quantidades sejam passadas na forma de uma `lista nomeada` pelo argumento `amostra`. A função retorna o negativo da verossimilhança. No exemplo a seguir começamos então simulando um conjunto de dados com $\alpha = 4.5$ e $\beta = 2$, e criamos o objeto com as estatísticas suficientes.

```
> set.seed(201107)
> dadosG <- rgamma(20, shape = 4.5, rate=2)
> am <- list(media=mean(dadosG), media.logs = mean(log(dadosG)),
+            n=length(dadosG))
```

Quando possível, é mais conveniente fazer o gráfico das superfícies de verossimilhança na escala da *deviance*. Para isto precisamos das estimativas dos parâmetros. Neste ponto vamos simplesmente usar os objetos `mod1`, `mod2` e `mod3` que contém as estimativas. Mais adiante vamos discutir em mais detalhes a obtenção das estimativas.

Os comandos a seguir mostram como obter o gráfico da superfície de verossimilhança (*deviance*) para a parametrização 1. Utilizamos a função *deviance* genérica definida em **??** para ser usada com densidades com dois parâmetros. Definimos uma sequencia de valores para cada parâmetro e o valor da *deviance* é calculado em cada ponto da malha que combina os valores usando a função `outer()`. A partir da malha os contornos parametrização 1 na escala original dos parâmetros são desenhados na forma de isolinhas. Comandos análogos são usados para as demais parametrizações nas escalas originais e logarítmicas.

Na Figura fig:devSurf-Gamma mostramos as superfícies para cada parametrização nas escalas originais e logarítmicas dos parâmetros. Simetria e ortogonalidade da superfície facilitam e melhoram o desempenho de algo-

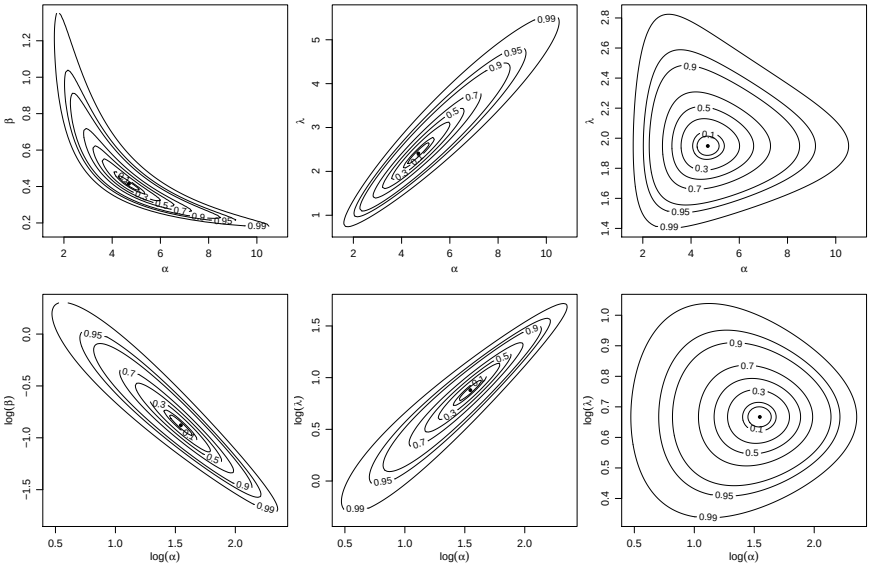

Figura 1.13: Superfícies de deviance sob diferentes parametrizações - Distribuição Gama.

ritmos numéricos, sejam de optimização ou de algorítmos de amostragem como por exemplo os de *MCMC* (cadeias de Markov por Monte Carlo). A superfície de verossimilhança para a parametrização 3 é a que apresenta melhores características. Isto sugere que algorítimos de cálculos de verossimilhança em procedimentos numéricos utilizando a Gama devem ser escritos nesta parametrização transformando ao final os resultados para outras parametrizações de interesse, se for o caso.

```
> alpha <- seq(1.5, 10.9, len = 100)
> beta <- seq(0.17, 1.35, len = 100)
> dev1 <- outer(alpha, beta, FUN = devSurf, est = mod1, llFUN = neglLik,
+     amostra = am, modelo = 1)
> LEVELS <- c(0.99, 0.95, 0.9, 0.7, 0.5, 0.3, 0.1, 0.05)
> contour(alpha, beta, dev1, levels = qchisq(LEVELS, df = 2),
+     labels = LEVELS, xlab = expression(alpha), ylab = expression(beta))
> points(t(mod1), pch 19, cex = 0.6)
```

Neste exemplo passamos detalhadamente por cada passo do processo de inferência. Apresentamos o algoritmo de Newton-Raphson e comparamos três abordagens para a construção de intervalos de confiança. Neste exemplo, ficou claro o uso intensivo de ferramentas do cálculo diferencial para obter a função escore e a matriz de informação. Ficou claro também que nem sempre é fácil obter estas quantidades analiticamente.

1.11 Exemplo 7 - Distribuição Binomial Negativa

Considere $Y_1, Y_2, \ldots, Y_n$ variáveis aleatórias independentes provenientes de uma distribuição Binomial Negativa de parâmetros $\phi \in \Re^+$ e $0 < p \leq 1$ e $y = 0,1,\ldots$. Usamos dados de uma amostra $y_1, y_2, \ldots, y_n$ estimar os parâmetros ϕ e p e possivelmente construir intervalos de confiança e testes de hipóteses. Neste caso temos dois parâmetros, o que leva a uma superfície de verossimilhança. A função de distribuição da Binomial Negativa é dada por:

$$p(y) = \frac{\Gamma(y+\phi)}{\Gamma(\phi)y!}p^{\phi}(1-p)^{y}, \quad \text{para} \quad 0 < p \leq 1 \quad \phi > 0 \quad \text{e} \quad y = 0,1,2,\ldots.$$

Sendo n amostras independentes a função de verossimilhança tem a seguinte forma,

$$\begin{aligned} L(\phi,p) &= \prod_{i=1}^{n} \frac{\Gamma(y_i+\phi)}{\Gamma(\phi)y_i!}p^{\phi}(1-p)_i^{y} \\ L(\phi,p) &= p^{n\phi}(1-p)^{\sum_{i=1}^{n} y_i} \prod_{i=1}^{n} \frac{\Gamma(y_i+\phi)}{\Gamma(\phi)y_i!}. \end{aligned}$$

Logo a função de log-verossimilhança pode ser escrita como,

$$l(\phi,p) = n\phi \log p + \sum_{i=1}^{n} y_i \log(1-p) + \sum_{i=1}^{n} \log\Gamma(y_i+\phi) - n\log\Gamma(\phi) - \sum_{i=1}^{n} \log y_i!.$$

Derivando em relação aos parâmetros ϕ e p chegamos as equações escore.

$$\begin{aligned} U(\phi) &= n\log p + \sum_{i=1}^{n} \psi_0(y_i+\phi) - n\psi_0(\phi) \\ U(p) &= \frac{n\phi}{p} - \sum_{i=1}^{n} y_i/(1-p) \end{aligned}$$

Para obter a matriz de informação precisamos das derivadas segundas que

são:

$$\begin{aligned}
\frac{\partial^2 l(\phi,p)}{\partial \phi^2} = \frac{\partial U(\phi)}{\partial \phi} &= \frac{\partial}{\partial \phi} n \log p \sum_{i=1}^{n} \psi_0(y_i + \phi) - n\psi_0(\phi) \\
&= \sum_{i=1}^{n} \psi_1(y_i + \phi) - n\psi_1(\phi) \\
\frac{\partial^2 l(\phi,p)}{\partial p^2} = \frac{\partial U(p)}{\partial p} &= \frac{\partial}{\partial p} \frac{n\phi}{p} - \sum_{i=1}^{n} y_i/(1-p) \\
\frac{\partial l(\phi,p)}{\partial \phi \partial p} = \frac{\partial U(\phi)}{\partial p} &= \frac{\partial}{\partial p} n \log p + \sum_{i=1}^{n} \psi_0(y_i + \phi) - n\psi_0(\phi).
\end{aligned}$$

Dado estes resultados temos todos os ingredientes para estimar ϕ e p através do algoritmo de Newton-Raphson. Note que pela equação $U(p)$ podemos obter a estimativa de p em função do parâmetro desconhecido ϕ da mesma forma que no modelo Gama. Porém, neste exemplo não vamos usar verossimilhança concentrada para exemplificar o uso do algoritmo de Newton-Raphson em duas dimensões.

A implementação do modelo começa sempre com a construção da função de log-verossimilhança, neste caso tal qual escrevemos no papel. Adiante veremos como usar as funções residentes do *R* para implementar isso de forma mais eficiente. Vamos também simular uma amostra de tamanho $n = 100$ desta distribuição para imitar uma realização do modelo, vamos fixar os valores dos parâmetros como $\phi = 100$ e $p = 0.8$.

```
> set.seed(123)
> y <- rnbinom(100, size=100, p = 0.8)
```

Código 1.20: Função de log-verossimilhança para a distribuição binomial negativa.

```
> ll.neg.binomial <- function(par, y){
+   phi <- par[1]
+   p <- par[2]
+   n <- length(y)
+   ll <- n*phi*log(p) + sum(y)*log(1-p) + sum(log(gamma(y+phi))) -
+         n*log(gamma(phi)) - sum(log(factorial(y)))
+   return(ll)}
```

Pensando em otimizar a função de log-verossimilhança usando o algoritmo de Newton-Raphson o próximo passo é implementar a função escore e na sequência a matriz de segundas derivadas ou Hessiana.

Código 1.21: Funções escore e hessiana para a distribuição binomial negativa.

```
> escore <- function(par, y){
+   phi <- par[1]
+   p <- par[2]
+   n <- length(y)
+   U.phi <- n*log(p) + sum(digamma(y+phi)) - n*digamma(phi)
+   U.p <- (n*phi)/p - sum(y)/(1-p)
+   return(c(U.phi,U.p))}
> Hessiana <- function(par, y){
+   phi = par[1]
+   p = par[2]
+   n <- length(y)
+   II <- matrix(c(sum(trigamma(y+phi)) - n*trigamma(phi),
+                  n/p,n/p, -(n*phi)/p^2 - sum(y)/(1-p)^2),2,2)
+   return(II)}
```

```
> NewtonRaphson(initial=c(50,0.5), escore=escore,hessiano=Hessiana,
+               max.iter=100, tol = 1e-10, n.dim=2, y=y)
[1] 114.1976249   0.8213433
```

Para construção do intervalo de confiança assintótico e/ou baseado em perfil de verossimilhança podemos proceder exatamente igual ao exemplo da distribuição Gama. Deixamos como exercício para o leitor obter estes intervalos e compará-los.

1.12 Tratando tudo numericamente

Vimos nos exemplos anteriores que métodos numéricos são essenciais em inferência baseada em verossimilhança. Mesmo em exemplos simples com apenas dois parâmetros soluções analíticas não podem ser obtidas. Vimos também que o algoritmo de Newton-Raphson é muito poderoso para resolver sistemas do tipo $f(x) = 0$, porém ele necessita do vetor escore e da matriz de derivadas segundas (Hessiana), o que nem sempre pode ser fácil de ser obtido e mesmo em exemplos simples pode demandar trabalho e/ou computacional analítico considerável.

Além disso, vimos que a implementação de intervalos baseados em perfil de verossimilhança é um tanto quanto tediosa, mesmo os intervalos de Wald que são simples exigem de um tempo razoável de programação que precisa ser muito cuidadosa para evitar problemas e exceções numéricas. Nesta seção nos vamos abordar os mesmos três problemas a dois parâmetros já apresentados porém tratando eles de forma inteiramente numérica, o que pode ser útil para investigar, ainda que preliminarmente, o comportamento de modelos existentes ou em desenvolvimento. Para isto, vamos

usar a função `optim()` que implementa quatro poderosos algoritmos de otimização numérica, são eles *Nelder-Mead*, *Gradiente Conjugado*, *Simulating Annealing* e *BFGS*.

Porém, só esta função não resolve o problema da construção dos intervalos de confiança e testes de hipóteses. Para isto, vamos introduzir o pacote **bbmle** que traz uma verdadeira "caixa de ferramentas"para inferência baseada em verossimilhança. Ilustramos o uso mostrando que com pouca programação conseguiremos estimar os parâmetros, construir intervalos de confiança assintóticos e perfilhados, obter testes de hipóteses, além de obter os resultados apresentados como no padrão de funções centrais do R como a `lm()` e `glm()`.

Veremos também como os resultados analíticos já obtidos, principalmente o vetor escore, podem ser usados para acelerar e melhorar a performance dos otimizadores numéricos. Relembre que no Exemplo 1.9, tratamos o modelo gaussiano com média μ e variância σ^2. Podemos escrever o **negativo** da log-verossimilhança deste modelo em R da seguinte forma:

Código 1.22: Função de log-verossimilhança para a distribuição gaussiana em formato vetorial.

```
> ll.gauss <- function(par, dados){
+   ll <- sum(dnorm(dados, par[1], sd=par[2], log=TRUE))
+   return(-ll)}
```

Note que escrevemos o negativo da log-verossimilhança, isso é meramente computacional porque, por padrão, a função `optim()` minimiza a função objetivo. Observe que passamos os parâmetros a serem estimados como um vetor único em que cada posição representa um parâmetro a ser estimado. Para proceder o processo de estimação precisamos de uma amostra, vamos simular uma amostra de tamanho $n = 100$ do modelo Normal com $\mu = 0$ e $\sigma = 1$ apenas para ilustrar o procedimento via a função `optim()`. Na maioria dos algoritmos de otimização numérica será necessário o uso de valores iniciais, para isto é comum utilizar alguma estimativa grosseira ou mesmo fazer várias tentativas e avaliar a sensibilidade do algoritmo as condições iniciais.

Um cuidado que se deve ter quando se usa este tipo de algoritmos numéricos e o espaço de busca, que no caso de inferência estatística corresponde ao espaço paramétrico. Para o parâmetro μ a busca deve ser em toda a reta real, porém para σ apenas nos reais positivos. A maioria dos algoritmos da `optim()` não leva isso em consideração, fazendo com que tente avaliar a função de verossimilhança em pontos não válidos. Este problema pode gerar desde simples mensagens de *warnings* até a falha total do algo-

ritmo. Reparametrizações são muito úteis neste ponto. O único algoritmo dentro da `optim()` que permite busca em espaços determinados é o `L-BFGS-B` é o que vamos usar neste exemplo. Recomendamos que o leitor leia a documentação da função e experimente os outros algoritmos.

```
> set.seed(123)
> dados <- rnorm(100,0,1)
> (est.gauss <- optim(par=c(0.5,10), fn = ll.gauss, dados=dados,
+              method="L-BFGS-B", lower=c(-Inf, 1e-32),
+              upper=c(Inf,Inf), hessian=TRUE))
```

```
$par
[1] 0.09040592 0.90824121

$value
[1] 132.2692

$counts
function gradient
      19       19

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"

$hessian
              [,1]          [,2]
[1,]  1.212265e+02 -4.114042e-06
[2,] -4.114042e-06  2.424550e+02
```

A saída da `optim()` retorna uma lista. O primeiro elemento é o valor que maximiza o negativo da log-verossimilhança em seguida o valor que ela toma neste ponto, muito importante para a construção de testes de hipóteses e comparações de modelos, na sequência diversas informações sobre o procedimento e por fim a matriz Hessiana obtida numericamente, importante para a construção dos intervalos de confiança assintóticos. Deste ponto, ainda é preciso uma quantidade razoável de programação para obtenção de intervalos perfilhados. Uma forma mais rápida de obter praticamente tudo que se deseja do processo de inferência é usar o pacote **bbmle**, que facilita todo o procedimento.

Para usar este pacote precisamos escrever a função de log-verossimilhança ligeiramente diferente. O código abaixo apresenta as funções de log-verossimilhança para os casos Gaussianos, Gama e Binomial Negativo.

Código 1.23: Funções de log-verossimilhança escritas em formato compatível para estimação com a função `mle2()`.

```
> ll.gauss <- function(mu, sigma, dados){
+   ll <- sum(dnorm(dados, mu, sigma, log=TRUE))
+   return(-ll)}
> ll.gamma <- function(a,s, dados){
+   ll <- sum(dgamma(dados,shape=a,scale=s, log=TRUE))
+   return(-ll)}
> ll.negbin <- function(phi, p, dados){
+   ll <- sum(dnbinom(dados,size=phi, p=p, log=TRUE))
+   return(-ll)}
```

A estimação via o pacote **bbmle** é feita através da função `mle2()`, que é apenas uma casca para a `optim()` mas facilita a entrada de dados e formata os resultados de forma conveniente. Veja um exemplo de chamada para o modelo gaussiano.

```
> require(bbmle)
> est.gauss <- mle2(ll.gauss, start=list(mu=0.5,sigma=100),
+                   data=list(dados=dados), method="L-BFGS-B",
+                   lower=list(mu=-Inf, sigma=1e-32),
+                   upper=list(mu=Inf,sigma=Inf))
```

Neste formato os resultados podem ser explorados desde um simples resumo do modelo ajustado via função `summary()`,

```
> summary(est.gauss)
Maximum likelihood estimation

Call:
mle2(minuslogl = ll.gauss, start = list(mu = 0.5, sigma = 100),
    method = "L-BFGS-B", data = list(dados = dados), lower = list(mu = -Inf,
        sigma = 1e-32), upper = list(mu = Inf, sigma = Inf))

Coefficients:
      Estimate Std. Error z value  Pr(z)
mu    0.090407   0.090828  0.9954 0.3196
sigma 0.908243   0.064223 14.1421 <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

-2 log L: 264.5385
```

construção de intervalos de confiança assintóticos,

```
> confint(est.gauss,method="quad")
           2.5 %   97.5 %
mu    -0.0876118 0.268426
sigma  0.7823690 1.034118
```

ou construção de intervalos de confiança perfilhados.

```
> confint(est.gauss)
            2.5 %    97.5 %
mu    -0.08934198 0.2701538
sigma  0.79559101 1.0503091
```

O código baixo mostra o procedimento para o caso Gama, note a definição dos intervalos de busca para o algoritmo `L-BFGS-B`.

```
> set.seed(123)
> dados.gama <- rgamma(100, shape=10, scale=1)
> est.gamma <- mle2(ll.gamma, start=list(a=2,s=10),
+                data=list(dados=dados.gama), method="L-BFGS-B",
+                lower=list(a=1e-32,s=1e-32), upper=list(a=Inf,s=Inf))
> summary(est.gamma)
Maximum likelihood estimation

Call:
mle2(minuslogl = ll.gamma, start = list(a = 2, s = 10), method = "L-BFGS-B",
    data = list(dados = dados.gama), lower = list(a = 1e-32,
        s = 1e-32), upper = list(a = Inf, s = Inf))

Coefficients:
  Estimate Std. Error z value     Pr(z)
a 13.53632    1.89127  7.1573 8.230e-13 ***
s  0.72285    0.10289  7.0256 2.132e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

-2 log L: 474.394
> confint(est.gamma)
       2.5 %     97.5 %
a 10.1649566 17.5943054
s  0.5538588  0.9687489
> confint(est.gamma,method="quad")
      2.5 %     97.5 %
a 9.8295055 17.2431423
s 0.5211908  0.9245057
```

A seguir para o modelo Binomial Negativo.

```
> set.seed(123)
> dados.nbin <- rnbinom(1000, size=10, p=0.8)
> est.nbin <- mle2(ll.negbin,start=list(phi=2,p= 0.5),
+                  data=list(dados=dados.nbin), method="L-BFGS-B",
+                  lower=list(phi=1e-32, p= 1e-32),
+                  upper=list(phi=Inf,p=0.99999))
> summary(est.nbin)
Maximum likelihood estimation

Call:
mle2(minuslogl = ll.negbin, start = list(phi = 2, p = 0.5), method = "L-BFGS-B",
    data = list(dados = dados.nbin), lower = list(phi = 1e-32,
        p = 1e-32), upper = list(phi = Inf, p = 0.99999))

Coefficients:
    Estimate Std. Error z value     Pr(z)
phi 8.634386   1.737165  4.9704 6.682e-07 ***
p   0.772486   0.035581 21.7103 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

-2 log L: 3853.87
```

```
> confint(est.nbin,method="quad")  # assintotico
        2.5 %     97.5 %
phi 5.2296059 12.0391669
p   0.7027476  0.8422245
> confint(est.nbin)                # perfilhada
        2.5 %     97.5 %
phi 6.0825778 13.7049646
p   0.7043744  0.8436871
```

O esforço de programação quando tratamos tudo numericamente, sem utilizar resultados analíticos parciais é menor e adequado em explorações preliminares de modelos. Entretanto, note-se que isto torna a inferência computacionalmente mais cara e potencialmente mais instável. O uso das funções do pacote **bbmle** facilita muito o trabalho de inferência, pois já implemente métodos genéricos, além de formatar adequadamente a saída na forma de outras funções padrão do R. Implementações para uso intensivo podem, em algums situações, ser porteriormente ajustadas para um melhor desempenho se necessário.

Finalmente é importante dizer que as implementaçõa feitas neste Capítulo são ilustrativas e outros algorítmos e funções mais gerais podem ser escritos. Por exemplo, para a classe de modelos na família exponencial, que engloba todas as distribuições usadas aqui, é possível escrever expressões gerais, válidas para todas estas distribuições. Neste caso, as expressões no Método de Newton-Raphson podem ser escritas na forma de um procedimento iterativo de minímos quadrados que são reponderados a cada iteração (*IRWLS - iteractive reweighted least squares*).

Capítulo 2

Modelos de regressão

Modelos estocásticos têm sido amplamente utilizados tanto na comunidade científica como no mundo dos negócios em geral. Estudos de mercado usando modelos altamente estruturados, modelos de predição em séries financeiras, análises de componentes de solos para agricultura de precisão, mapeamento de doenças, modelos ecológicos, ambientais e climáticos são algumas das áreas de aplicações de tais modelos, entre tantas outras. De todos os métodos estatísticos, os modelos de regressão talvez sejam o mais amplamente utilizados na prática. Nestes modelos procura-se explicar, ao menos parcialmente, a variabilidade de uma variável considerada como resposta, por outras variáveis chamadas de explicativas ou covariáveis. Procura-se verificar a existência e quantificar o efeito das covariáveis sobre a resposta, embora a relação não seja necessariamente de causa e efeito.

A área de modelos de regressão teve seu início com o tradicional modelo de regressão linear gaussiano, que, seja por sua aplicabilidade e/ou pela facilidade analítica e computacional, foi e continua sendo largamente utilizado para descrever a associação entre um conjunto de covariáveis e uma variável resposta. Diversas transformações na resposta foram propostas para adaptar a suposição de normalidade. Porém tais modelos não são satisfatórios para respostas discreta como no caso de dados binários, contagens baixas e respostas restritas a certos intervalos como proporções e índices com valores em intervalos limitados.

A modelagem estatística por modelos de regressão recebeu um importante novo impulso desde o desenvolvimento dos modelos lineares generalizados no início da década de 70 cujo marco inicial é o trabalho de Nelder & Wedderburn (1972) que estendeu de uma forma mais geral a classe de modelos de regressão, acomodando sob a mesma abordagem diversas distribuições agrupadas na forma da família exponencial. Variáveis resposta

binárias e de contagens foram as que receberam mais atenção, talvez pela dificuldade em tratar deste tipo de resposta com transformações do modelo linear gaussiano. Desde então modelos de regressão logística e de Poisson, passaram a ser utilizados nas mais diversas áreas de pesquisa. Nas décadas seguintes e até os dias de hoje o modelo linear generalizado (MLG) tornou-se uma estrutura de referência inicial, canônica, a partir da qual diversas modificações, expansões e alternativas são propostas na literatura. Classes mais amplas de distribuições para resposta, preditores não lineares ou aditivamente lineares, modelagem conjunta de média e dispersão, adição de variáveis latentes no preditor são algumas das importantes extensões, às quais adicionam-se métodos para inferência e algorítimos para computação.

Neste capítulo vamos considerar alguns modelos simples de regressão com ilustrações computacionais. No Capítulo seguinte consideramos modelos com extensões com inclusão de efeitos aleatórios.

Considere que $Y_1, Y_2, \ldots, Y_n$ são variáveis aleatórias independentes e identicamente distribuídas e X é a matriz de delineamento do modelo conhecida. Um modelo de regressão em notação matricial pode ser escrito da seguinte forma:

$$Y|X \sim f(\underline{\mu}, \phi); \underline{\mu} \qquad = g(X; \underline{\beta}).$$

A distribuição de probabilidade $f(\underline{\mu}, \phi)$ da variável resposta é descrita por dois conjuntos de parâmetros, os de locação (média) e os de dispersão (precisão / variância). Por simplicidade de apresentação supomos aqui que o parâmetro de dispersão é comum a todas as observações e o que muda é apenas a média. Esta restrição pode ser flexibilizada, supondo que o parâmetro de dispersão, também possa ser descrito por alguma função das covariáveis que é possível requerendo uma generalização da implementação computacional.

Nesta distribuição é muito importante estar atento ao espaço paramétrico de $\underline{\mu}$ e ϕ. O segundo em geral tem como seu espaço paramétrico os reais positivos, já que, o parâmetro de variabilidade tem que ser necessariamente positivo. Para a parte de média devemos ter mais cuidado, uma vez que a função $g(\cdot)$ deve mapear o preditor linear ou não-linear, que pode assumir qualquer valor real, para o espaço paramétrico de $\underline{\mu}$. Por exemplo, se estamos trabalhando com dados de contagens onde o vetor $\underline{\mu}$ representa o parâmetro da distribuição de Poisson. A função $g(\cdot)$ deve então mapear dos reais para os reais positivos. Em outra situação, por exemplo com dados restritos ao intervalo $(0,1)$, a função $g(\cdot)$ deve mapear dos reais para o intervalo unitário e assim de forma análoga para outros espaço paramétricos.

Note que para a declaração completa do modelo são necessárias duas

suposições. A primeira se refere a distribuição de probabilidade atribuída a variável resposta, neste caso $f(\underline{\mu}, \phi)$. A segunda é a definição de uma função $g(\cdot)$ fazendo com que o preditor linear ou não-linear, que pode apresentar qualquer valor nos reais, seja mapeado adequadamente para o espaço paramétrico da parte de média do modelo.

A função $g(\cdot)$ deve ser duplamente diferenciável. Uma restrição adicional comum é que seja estritamente monótona, porém em modelos não-lineares, isso nem sempre é necessário. Esta função tem como seus argumentos a matriz de delineamento X conhecida e os parâmetros $\underline{\beta}$ desconhecidos a serem estimados. Sendo a função $g(\cdot)$ definida e mapeando os valores do preditor, seja ele linear ou não, para espaço paramétrico de $\underline{\mu}$, o modelo está composto e é passível de ter seus parâmetros estimados pela maximização da função de verossimilhança.

Um fato que pode passar desapercebido é com relação ao suporte da distribuição concordar ou não com os possíveis valores do fenômeno em estudo. Por exemplo, ao estudar o peso de animais é comum atribuir o modelo gaussiano, porém este modelo tem como suporte os reais, enquanto que o fenômeno varia apenas nos reais positivos. Nestes casos a usual justificativa prática é que a probabilidade atribuída a parte negativa é tão pequena que é desprezível na prática.

Uma outra situação é quando trabalhamos com uma variável resposta restrita ao intervalo unitário. Podemos definir como função $g(\cdot)$ o inverso da função *logit* e vamos estar mapeando a média do modelo ao intervalo unitário compatível com os dados. Porém, na escolha da distribuição nem sempre isso é feito. É comum por exemplo, encontrar aplicações de modelos não-lineares atribuindo a distribuição gaussiana para $f(\underline{\mu}, \phi)$ que tem suporte nos reais e em desacordo com os dados restritos a algum intervalo. Apesar deste tipo de construção não ser incomum em aplicações, é recomendado atribuir uma distribuição de probabilidade que tenha suporte concordante com o campo de variação da variável resposta. No caso de variável resposta restrita ao intervalo unitário, a distribuição Beta ou Simplex seriam opções razoáveis. Tal fato motiva um dos Exemplo na Sessão 2.2.

Dado a relevância de modelos de regressão em aplicações nas mais diversas áreas do conhecimento, neste capítulo vamos mostrar como implementar a estimação de alguns modelos de regressão tradicionais, como regressão de Poisson, Simplex e não-linear gaussiano. Vamos explorar o algoritmo de Newton-Raphson no modelo de regressão Poisson, para o modelo Simplex, vamos explorar o conceito de verossimilhança concentrada e exemplificar como usar a função *escore* obtida analiticamente dentro do algoritmo *BFGS* implementado na função `optim()`. No último modelo vamos tratar tudo numericamente apenas como uma implementação simplificada. Neste capítulo, como material extra apresentamos a implementação

computacional de modelos dinâmicos e um estudo de caso de modelos de regressão para contagens subdispersas.

2.1 Regressão Poisson

Em situações onde a resposta é discreta na forma de contagens, escolha de distribuição para variável resposta recai, ao menos inicialmente, sobre a distribuição de Poisson. Conforme mencionado anteriormente, um modelo de regressão qualquer é descrito por duas suposições: a distribucional que se refere a distribuição de probabilidade atribuída a variável resposta e a uma função $g(\cdot)$ que liga o preditor à média desta distribuição. No caso do modelo de regressão de Poisson a distribuição tem a seguinte forma,

$$P(Y = y) = \frac{\exp\{-\lambda\}\lambda^y}{y!}, \quad \text{em que} \quad \lambda \geq 0 \quad \text{e } y = 0,1,2\ldots.$$

No modelo Poisson o parâmetro indexador λ corresponde à esperança de Y. O modelo é um caso particular da família de MLG em que temos apenas a componente de média, já que a variância é igual a média. O parâmetro λ deve ser maior que zero e portanto devemos escolher uma função $g(.)$ que mapeie dos reais aos reais positivos e a escolha usual para regressão Poisson é a exponencial. No contexto de modelos lineares generalizados pode-se mostrar que a função de ligação canônica para modelo Poisson é a função logaritmo cuja a inversa é a exponencial, por isso sua popularidade. Resumindo, supomos que $\underline{Y} \sim P(\underline{\lambda})$ e que $\underline{\lambda} = \exp\{X\underline{\beta}\}$ com a suposição adicional que o preditor é linear.

Expressões genéricas para função de verossimilhança, escore e informação, assim como algoritmos para estimação dos parâmetros são disponíveis para toda classe de MLG. Entretanto como nosso objetivo aqui é ilustrar as computações vamos considerar o caso da Poisson individualmente. A função de log-verossimilhança escrita em formato matricial é dada por,

$$l(\underline{\beta}, \underline{y}) = -1^\top g(X\underline{\beta}) + \underline{y}^\top X\underline{\beta} + 1^\top \log(\underline{y}!).$$

Derivando em relação ao vetor de parâmetros $\underline{\beta}$ temos a função escore.

$$U(\underline{\beta}) = \left(\underline{y} - \exp\{X\underline{\beta}\}\right)^\top X$$

Derivando novamente obtemos a matriz de informação observada em que $diag(X\underline{\beta})$ denota uma matrix diagonal com elementos $\underline{\lambda} = X\underline{\beta}$:

$$I_o(\underline{\beta}) = -X^\top [diag(\exp\{X\underline{\beta}\})]X.$$

Com isso temos todos os elementos para montar o algoritmo de Newton-Raphson e encontrar as estimativas de máxima verossimilhança para o vetor β.

Para exemplificar o processo vamos considerar dados simulados supondo que $X\beta$ tem a forma $\beta_0 + \beta_1 x_i$, x_i é uma covariável com valores conhecidos. Nas simulações a covariável assume $n = 10, 50$ e 100 valores igualmente espaçados entre 0 e 5. Assim, o modelo tem apenas dois parâmetros $\beta = (\beta_0, \beta_1)$. O código abaixo apresenta uma função para simular realizações deste modelo e mostramos o comando para gerar o conjunto de dados com $n = 10$.

Código 2.24: Função para simular variáveis aleatórias de uma modelo de regressão de Poisson.

```
> simula.poisson <- function(formula, beta) {
+     X <- model.matrix(formula)
+     lambda <- exp(X %*% beta)
+     y <- rpois(nrow(X), lambda = lambda)
+     return(data.frame(y = y, X))
+ }
```

```
> set.seed(123)
> cov <- seq(0, 5, length=10)
> dados10 <- simula.poisson(~cov, c(2,0.5))
```

A Figura 2.1 apresenta a superfície de log-verossimilhança em escala de *deviance* exata e pela aproximação quadrática, para diferentes tamanhos de amostras. As linhas de contorno definem regiões de confiança que correspondem aos quantis 99, 95, 90, 70, 50, 30, 10 e 5 % de distribuição χ^2_2. O ponto indica o valor verdadeiro do parâmetro. Note-se que os gráficos não estão na mesma escala.

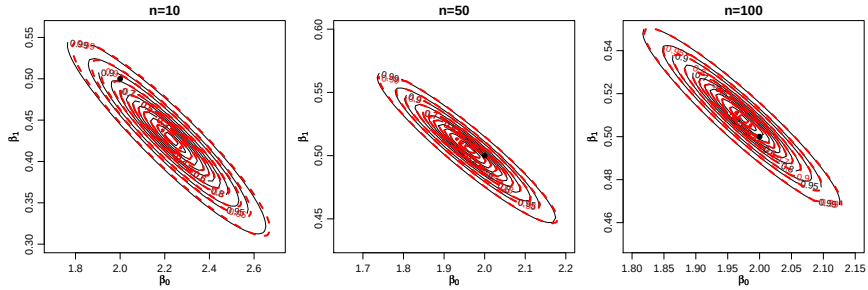

Figura 2.1: Superfície de deviance para diferentes tamanhos de amostra - Regressão Poisson, detalhes no texto.

Podemos perceber que a aproximação quadrática é excelente para os dois parâmetros, o que era esperado, já que, se tratam de parâmetros de

média. Entretanto, mesmo esses parâmetros podem ter comportamento assimétrico em algumas situações, como por exemplo, na presença de dados censurados. A relação entre os dois parâmetros também é clara nos gráficos mostrando que os parâmetros não são ortogonais. Note que com o aumento do tamanho da amostra os contornos vão ficando menores, mais concentrados em torno do verdadeiro valor do parâmetro, sugerindo nesta ilustração a propriedade assintótica de consistência e não-viés.

Seguindo com o processo de inferência podemos usar o algoritmo de Newton-Raphson para encontrar as estimativas de máxima verossimilhança de β_0 e β_1, para isto precisamos escrever duas funções, a `escore()` e a `hessiana()` conforme código 2.25. Definimos a função hessiana de duas maneiras na liguagem R. Na primeira programamos diretamente como na expressão. Na sequência reescrevemos a função evitando usar a matriz completa de pesos já que apenas os elementos fora da diagonal são nulos e portanto desnecessários.

Código 2.25: Função escore e hessiana para a regressão de Poisson.

```
> ## Função escore
> escore <- function(par, formula, dados){
+   mf <- model.frame(formula, dados)
+   X <- model.matrix(formula, data=mf)
+   esco <- crossprod(model.response(mf) - exp(X %*% par), X)
+   return(drop(esco))
+ }
> ## Hessiana ("naïve")
> hessiano <- function(par, formula, dados){
+   X <- model.matrix(formula, data=dados)
+   mat <- matrix(0, nrow(X), nrow(X))
+   diag(mat) <- -exp(X%*%par)
+   H <- t(X) %*% mat %*% X
+   return(H)
+ }
> ## Hessiana (equivalente a anterior)
> hessiano <- function(par, formula, dados){
+   X <- model.matrix(formula, data=dados)
+   H <- crossprod(X * -exp(drop(X%*%par)),  X)
+   return(H)
+ }
```

Usando o algoritmo de Newton-Raphson já apresentado em 1.15, temos as estimativas para o conjunto simulado com $n = 10$.

```
> (beta10 <- NewtonRaphson(initial=c(0,0), escore=escore,
+                          hessiano=hessiano, max.iter=1000,
+                          formula=y~cov, dados=dados10))
```

```
(Intercept)         cov
   2.228564    0.427679
```

Para a construção dos intervalos de confiança assintóticos, basta avaliar a inversa da matriz hessiana no ponto encontrado e os erros padrão das estimativas são dados pelas raízes quadradas dos elementos diagonais.

```
> Io <- -hessiano(par=beta10, formula=y~cov, dados=dados10)
> Io
            (Intercept)      cov
(Intercept)    338.0015 1182.229
cov           1182.2288 4796.178

> (erro.padrao <- sqrt(diag(solve(Io))))

(Intercept)         cov
 0.14650661  0.03889277
```

Lembrando que as variâncias de $\hat{\beta_0}$ e $\hat{\beta_1}$ são dadas pelos termos da diagonal da inversa da matriz de informação observada, temos pela distribuição assintótica do EMV que um intervalo de 95% de confiança para β_0 e β_1 poderia ser obtido por:

```
> beta10[1] + c(-1,1)*qnorm(0.975)*erro.padrao[1]
[1] 1.941417 2.515712

> beta10[2] + c(-1,1)*qnorm(0.975)*erro.padrao[2]
[1] 0.3514506 0.5039075
```

Nossos resultados coincidem com as estimativas retornadas função `glm()` do R.

```
> beta10.glm <- glm(y ~ cov, family=poisson, data=dados10)
> coef(beta10.glm)
(Intercept)         cov
  2.2285674   0.4276769

> summary(beta10.glm)$coef
             Estimate Std. Error  z value     Pr(>|z|)
(Intercept) 2.2285674 0.14650663 15.21138 2.972301e-52
cov         0.4276769 0.03889281 10.99630 3.981440e-28

> confint(beta10.glm, type="quad")
                2.5 %    97.5 %
(Intercept) 1.9327201 2.5074004
cov         0.3525055 0.5050645
```

Os intervalos acima podem ser inadequados quando os parâmetros são não ortogonais por considerar isoladamente a curvatura em cada direção. O grau de não ortogonalidade também é influenciado pela da ordem de magnitude das covariáveis e consequentemente dos parâmetros. Uma primeira alternativa é a obtenção dos intervalos através de verossimilhança (ou deviance) perfilhada, que tipicamente fornece intervalos mais largos e por vezes assimétricos. Esta solução pode ser computacionalmente mais cara ou mesmo proibitiva dependendo da quantidade de observações e parâmetros no modelo.

```
> confint(beta10.glm)
                2.5 %    97.5 %
(Intercept) 1.9327201 2.5074004
cov         0.3525055 0.5050645
```

Uma outra solução é utilizar covariáveis centradas para obter verossimilhanças (aproximadamente) ortogonais. Além disto as covariáveis podem ainda ser escalonadas para que os coeficientes tenham ordem de grandeza comparáveis e a função tenha contornos ao menos aproximadamente circulares. Desta forma os coeficientes são inicialmente estimados pontualmente e por intervalo para variáveis transformadas. Posteriormente as estimativas são transformadas para escala original pela equação que define a reparametrização.

Com isso, exemplificamos a construção dos cálculos computacionais do modelo de regressão de Poisson. Procedemos a estimação por máxima verossimilhança, obtivemos o vetor *escore* e a matriz *hessiana* que possibilitaram o uso do algoritmo de Newton-Raphson. Vimos também que a aproximação quadrática no caso do modelo Poisson é bastante acurada, sendo que em nossos exemplos é visualmente difícil encontrar diferença entre ela e a exata mesmo para a amostra de tamanho 10. Usando resultados assintóticos construímos intervalos de confiança de duas formas diferentes. Além disso, com os resultados obtidos estamos aptos a proceder testes de hipóteses, por exemplo, $H_0 : \beta_1 = 0$ contra $H_1 : \beta_1 \neq 0$. Neste caso poderíamos usar o tradicional teste de Wald, teste escore ou mesmo o teste de razão de verossimilhança.

2.2 Regressão Simplex

Em situações práticas podemos estar diante de variáveis restritas ao intervalo $(0,1)$, como taxas, proporções e índices. O exemplo que motiva essa sessão vem da área de ciências sociais na qual é comum a mensuração de variáveis latentes (não observáveis) que muitas vezes buscam medir qualidade, através de indicadores indiretos sobre o fenômeno latente. Um exemplo deste tipo de análise é o IDH - Índice de Desenvolvimento Humano, preconizado pela ONU - Organização das Nações Unidas, em todo o mundo.

Estes índices que tradicionalmente, por construção, resultam sempre em valores no intervalo unitário, buscam medir indiretamente o nível de desenvolvimento de um país. O índice pode ser adaptado como o IDH-M (Índice de Desenvolvimento Humano - Municipal) para municípios, estados ou qualquer aglomeração de indivíduos, como por exemplo, bairros de uma cidade. A importância em ressaltar que a construção leva a um número no intervalo unitário, é a de que o índice é expresso como uma nota em uma escala comum, permitindo a comparação entre países, estados, municípios ou bairros.

Com foco em tais situações, selecionamos para este exemplo o Índice de Qualidade de Vida de Curitiba (IQVC). Este índice é composto por 19

indicadores separados em 4 áreas temáticas: Habitação, Saúde, Educação e Segurança. Estes indicadores buscam de forma indireta medir o nível de vida da população residente em cada um dos 75 bairros da cidade. A metodologia para sua construção segue as premissas do IDH, e como tal resulta sempre em valores no intervalo unitário. Para sua construção é utilizada a base de microdados do Censo 2000, disponibilizada pelo IBGE (Instituto Brasileiro de Geografia e Estatística). Um interesse comum é relacionar o IQVC com a renda média do bairro medida em salários mínimos vigentes na época da pesquisa, neste caso ano de 2000. Os dados são disponibilizados pelo IPPUC/PR (www.ippuc.org.br) e uma versão resumida está disponível no arquivo *simplex.txt* dos complementos *online*.

Para a construção do modelo de acordo com a abordagem que utilizamos neste material, é necessário fazer duas suposições, como nos MLG's usuais, a primeira a respeito da distribuição de probabilidade atribuída a variável resposta e a segunda referente a função $g(\cdot)$ que vai mapear o preditor linear ou não ao espaço paramétrico induzido pela distribuição suposta. Para este exemplo vamos supor a distribuição como sendo Simplex e a função $g(\cdot)$ como a inversa da função *logit*, comum em modelos lineares generalizados. A apresentação do modelo aqui baseia-se no trabalho de Miyashiro (2008).

A função densidade de probabilidade Simplex é indexada por dois tipos de parâmetros locação $\mu \in (0,1)$ e dispersão $\sigma^2 > 0$. Ambos podem ser decompostos em função de covariáveis observadas. Uma variável aleatória Y que segue o modelo Simplex tem função densidade dada por

$$f(y;\mu,\sigma^2) = [2\pi\sigma^2 y(1-y)^3]^{-1/2} \exp\left\{-d(y;\mu)/2\sigma^2\right\}, \quad y \in (0,1), \quad (2.1)$$

em que

$$d(y;\mu) = \frac{(y-\mu)^2}{y(1-y)\mu^2(1-\mu)^2}.$$

Sejam $Y_1, Y_2, \ldots, Y_n$ variáveis aleatórias independentes, sendo cada $Y_i \sim S(\mu_i, \sigma^2)$, $i = 1,2,\ldots,n$. O modelo de regressão Simplex é definido pela densidade 2.1, sendo as médias μ_i dadas por

$$\mu_i = g(x_i^\top \underline{\beta}) = g(\eta_i)$$

em que $g(\cdot)$ é uma função que transforma valores dos reais ao intervalo unitário, $\underline{\beta} = (\beta_1, \beta_2, \ldots, \beta_p)^\top$ é o vetor de parâmetros da regressão ($\underline{\beta} \in \Re^p$), $x_i^\top = (x_{i1}, x_{i2}, \ldots, x_{ip})^\top$ são os valores conhecidos de p covariáveis e η_i neste caso é o preditor linear. Poderíamos, sem perda de generalidade definir o preditor como

$$h(\mu_i) = x_i^\top \underline{\beta} = \eta$$

onde agora $h(\cdot)$ é uma função de ligação que transforma do $(0,1)$ em $\Re$, dependendo da situação pode ser mais simples definir de uma forma ou de outra. No caso Simplex vamos definir a função $h(\cdot)$ como sendo a função *logit* que facilita bastante a obtenção do vetor *escore*. O objetivo deste exemplo é mostrar como usar os otimizadores numéricos implementados na função `optim()`, porém usando o vetor *escore* obtido analiticamente.

Dada a função densidade de probabilidade, podemos facilmente obter a função de log-verossimilhança,

$$l_i(\mu_i,\sigma^2) = -0.5[\log(2\pi) - \log(\sigma^2) - 3\log\{y_i(1-y_i)\} - d(y_i;\mu_i)/\sigma^2].$$

Temos que $h(\mu_i) = x_i^\top \underline{\beta} = \eta_i$, então para obter o vetor escore precisamos derivar em relação a cada β_p.

$$\frac{\partial l(\underline{\beta},\sigma^2)}{\partial \beta_p} = \sum_{i=1}^{n} \frac{\partial l_i(\mu_i,\sigma^2)}{\partial \mu_i}\frac{\partial \mu_i}{\partial \eta_i}\frac{\partial \eta_i}{\partial \beta_i}.$$

Para obter o vetor escore para a parte de média precisamos de três componentes, o primeiro é:

$$\frac{\partial l(\mu_i,\sigma^2)}{\partial \mu_i} = -\frac{1}{2\sigma^2}\frac{\partial}{\partial \mu_i} d(y_i,\mu_i) = \tag{2.2}$$

$$= \sigma^{-2}\left[\underbrace{\frac{(y_i-\mu_i)}{(1-\mu)^2\mu^2(1-y)y} + \frac{(y_i-\mu_i)^2}{(1-\mu)^2\mu^3(1-y)y} - \frac{(y_i-\mu_i)^2}{(1-\mu_i)^3\mu^2(1-y_i)y_i}}_{u}\right],$$

o segundo,

$$\frac{\partial \mu_i}{\partial \eta_i} = \frac{1}{g'(\mu_i)} = \mu_i(1-\mu_i),$$

e o terceiro,

$$\frac{\partial \eta_i}{\partial \beta_i} = x_{ip}.$$

Em notação matricial, a função escore para $\underline{\beta}$ é dada por

$$U_{\underline{\beta}}(\underline{\beta},\sigma^2) = \sigma^{-2}X^\top T u,$$

onde X é uma matriz $n \times p$, de delineamento do modelo, $T = diag\{1/g'(\mu_1),\ldots,1/g'(\mu_i)\}$ e $u^\top = (u_1,\ldots,u_n)\top$.

A função escore para σ^2 é facilmente obtida, derivando $l(\underline{\beta},\sigma^2)$ em relação a σ^2,

$$U_{\sigma^2}(\underline{\beta},\sigma^2) = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4}\sum_{i=1}^{n} d(y_i;\mu_i).$$

Note que podemos obter $\hat{\sigma}^2$ analiticamente, sua equação é dada por

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} d(y_i, \hat{\mu}_i).$$

Com isso temos todos os elementos necessários para a implementação computacional do modelo de regressão Simplex. Não há uma implementação da densidade Simplex entre as funções básicas do R. Começamos então definindo uma função para densidade da Simplex.

Código 2.26: Função densidade de probabilidade para uma variável aleatória de distribuição Simplex.

```
> dsimplex <- function(y, mu, sigma, log=TRUE){
+   dis <- (y-mu)^2 / (y*(1-y)*mu^2 *(1-mu)^2)
+   dsim <- -0.5*log(2*pi) -0.5*log(sigma) -
+               (3/2)*log(y*(1-y)) - (1/(2*sigma))*dis
+   if(log == FALSE){dsim <- exp(dsim)}
+   return(dsim)
+ }
```

Definimos a função *logit* para $h(\cdot) = g^{-1}(.)$ e sua inversa $g(\cdot)$ fica:

Código 2.27: Inversa da função de ligação *logit* usada no modelo de regressão Simplex.

```
> inv.logit <- function(x, lambda){ 1/(1+exp(-x)) }
```

Dada a aplicação teremos que o vetor $\underline{\beta} = (\beta_0, \beta_1)$ então o modelo de regressão Simplex em R é apresentado no código abaixo. Note que como o parâmetro σ^2 tem solução analitica e depende apenas de μ_i não precisamos maximizar numéricamente em relação a este parâmetro, estamos novamente usando uma log-verossimilhança concentrada. Basta substituirmos seu estimador na expressão da log-verossimilhança.

Código 2.28: Função de log-verossimilhança para o modelo de regressão Simplex.

```
> inv.logit <- function(x, lambda){ 1/(1+exp(-x)) }
> modelo.simplex <- function(b0, b1, formula, data){
+   mf <- model.frame(formula, data=data)
+   y <- model.response(mf)
+   X <- model.matrix(formula, data=mf)
+   mu <- inv.logit(X%*%c(b0,b1))
+   d0 <- ((y-mu)^2)/(y*(1-y)*mu^2*(1-mu)^2)
+   sigma <- sum(d0)/length(y)
+   ll <- sum(dsimplex(y, mu=mu, sigma=sigma))
+   return(-ll)
+ }
```

O próximo passo é implementar a função escore já obtida, note novamente que ela é função apenas de dois parâmetros β_0 e β_1, uma vez que tenhamos estes substituímos na expressão de $\hat{\sigma}^2$ e o usamos na expressão do vetor escore.

Código 2.29: Função escore para o modelo de regressão Simplex.

```
> escore <- function(b0, b1, formula, data){
+   mf <- model.frame(formula, data=data)
+   y <- model.response(mf)
+   X <- model.matrix(formula, data=mf)
+   eta <- X%*%c(b0,b1)
+   mu <- inv.logit(eta)
+   d0 = ((y - mu)^2) / (y*(1-y)*mu^2 * (1-mu)^2)
+   sigma <- sum(d0)/length(y)
+   T <- diag(c(mu*(1-mu)))
+   u <- (1/(sigma*(1-mu)*mu))*(-((y-mu)^2/((1-mu)^2*mu^2)) +
+          (((y - mu)^2)/((1-mu)^2*mu^2))+((y-mu)/((1-mu)^2*mu^2)))
+   es <- (crossprod(X,T) %*% u)/sigma
+   return(as.numeric(-es))
+ }
```

Com isto, estamos aptos a passar a log-verossimilhança e o vetor escore para a optim e obter as estimativas de máxima verossimilhança para β_0 e β_1, com estas substituímos na expressão de $\hat{\sigma}^2$ e concluímos o processo de estimação. Antes disso, precisamos do conjunto de dados que será analisado. O arquivo chamado `simplex.txt` traz o conjunto para a análise. O código abaixo lê a base de dados e apresenta um pequeno resumo. A Figura 2.2 apresenta uma análise exploratória gráfica, com um histograma e um diagrama de dispersão relacionando o IQVC com a renda média do bairro em salários mínimos que foi divido por 10 para evitar problemas numéricos.

```
> dados <- read.table("simplex.txt",header=TRUE)
> head(dados)
```

```
  ID      y MEDIA
1 10 0.6416 1.471
2  6 0.7644 1.529
3 70 0.8580 1.906
4 69 0.6695 2.161
5 62 0.8455 1.730
6 61 0.7861 1.724
```

O conjunto de dados contêm apenas três colunas, a primeira denominada *ID* apenas identifica os diferentes bairros, a coluna *y* apresenta o Índice de Qualidade de Vida de Curitiba (IQVC) e a coluna *MEDIA* apresenta a renda média do bairro em salários mínimos dividido por 10.

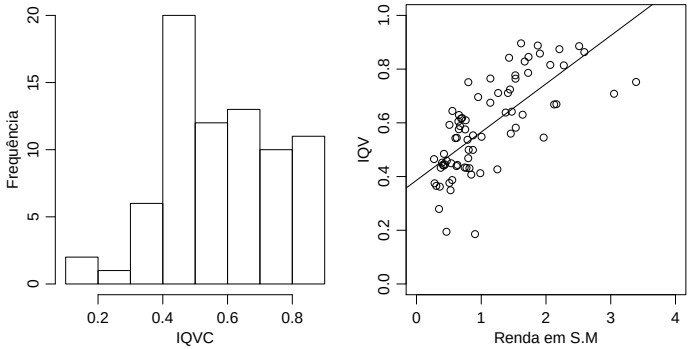

Figura 2.2: Histograma e diagrama de dispersão - IQVC.

Para o ajuste do modelo Simplex, vamos novamente usar as facilidades do pacote **bbmle** que usa internamente a função `optim()`. O ajuste é dado no seguinte código.

```
> require(bbmle)
> simplex.logit.gr <- mle2(modelo.simplex, start=list(b0=0,b1=0),
+                          gr=escore, method="BFGS",
+                          data=list(formula=y~MEDIA, data=dados))
```

A única diferença é no uso do argumento *gr* onde é passada a função escore com exatamente os mesmos argumentos adicionais usados na função de log-verossimilhança. A função `optim()` tem quatro otimizadores básicos: *Nelder-Mead*, *gradiente conjugado*, *BFGS* e *simulating annealing*. O uso de gradiente analítico só é possível nos algoritmos *BFGS* e *gradiente conjugado*.

O resumo tradicional do modelo pela função `summary()`, traz os erros padrões que são usados para a construção de intervalos de confiança assintóticos, que podem ser obtidos diretamente pela função `confint(..., method="quad")`.

```
> summary(simplex.logit.gr)
```

```
Maximum likelihood estimation

Call:
mle2(minuslogl = modelo.simplex, start = list(b0 = 0, b1 = 0),
    method = "BFGS", data = list(formula = y ~ MEDIA, data = dados),
    gr = escore)

Coefficients:
    Estimate Std. Error z value     Pr(z)
b0 -0.399817   0.113700 -3.5164 0.0004374 ***
b1  0.683628   0.071231  9.5974 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

-2 log L: -103.1299
> confint(simplex.logit.gr, method="quad")
        2.5 %     97.5 %
b0 -0.6226643 -0.1769699
b1  0.5440182  0.8232383
```

Deixamos como exercício para o leitor obter o intervalo de confiança para σ^2 e os perfis de verossimilhança para β_0 e β_1. Para finalizar o exemplo, podemos visualizar o ajuste sobreposto aos dados. Para isto, vamos escrever uma função genérica para gerar os preditos.

Código 2.30: Função para calcular os valores preditos para um modelo de regressão Simplex.

```
> my.predict <- function(modelo, formula, newdata){
+   X <- model.matrix(formula,data=newdata)
+   beta <- coef(modelo)[1:ncol(X)]
+   preditos <- inv.logit(X%*%beta)
+   return(preditos)
+ }
```

Usando a função `my.predict()` obtemos os valores preditos pelo modelo para rendas em uma sequência de valores.

```
> preditos <- my.predict(simplex.logit.gr, formula=~MEDIA,
+                        newdata= data.frame(MEDIA=seq(0,4,l=500)))
```

Por fim, plotamos um diagrama de dispersão com as observações e a média predita pelo modelo para a sequência de valores de renda.

2.3 Modelo contagem-Gama

Contagens são variáveis aleatórias que assumem valores inteiros não negativos. Esses valores representam o número de vezes que um evento ocorre em um domínio fixo contínuo, como um intervalo de tempo ou espaço, ou discreto, como a avaliação de um indivíduo ou setor censitário.

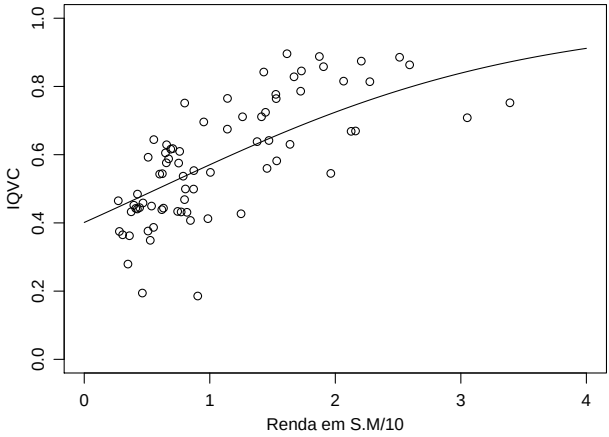

Figura 2.3: Diagrama de dispersão e modelo ajustado - IQVC 2000.

Em análise de dados de contagem o uso do modelo de regressão Poisson tem ocorrência predominante. Esse modelo tem sido uma escolha natural e imediata para essa classe de variável aleatória devido primeiramente à compatibilidade de suporte. Além disso, dados de contagem apresentam certa assimetria em relação à média e variabilidade que depende desse valor médio, duas características que são inerentes ao modelo de regressão Poisson. Entretanto, também inerente à esse modelo, está a forte suposição de equidispersão, ou seja, a da variância esperada ser igual à média esperada.

Afastamentos dessa suposição são frequentemente observadas em análise de dados, principalmente a superdispersão. Superdispersão corresponde à variância ser maior que a média. Isso pode ser resultado de

- ausência de covariáveis importantes, sejam ausentes por que não foram observadas ou ausentes porque não foram declaradas, como por exemplo a omissão da interação entre covariáveis;
- excesso de zeros, que na verdade trata-se de uma mistura de distribuições em que parte dos zeros são gerados pelo processo Poisson e parte é gerado por um processo Bernoulli;
- diferentes amplitudes de domínio, quando eventos são contados em intervalos e tempo ou espaço em que o tamanho desses intervalos é variável e não considerado na análise, geralmente como *offset*.
- efeito aleatório à nível de observações ou grupos de observações, quando existe no preditor teórico um componente estocástico, à nível de observações ou grupos de observação (blocos), ...

Note que em todas as situações acima foram introduzidos processos que geram mais variabilidade (mistura com Bernoulli, efeito aleatório) ou que não permitem explicar a variabilidade existente (desconhecimento dos domínios, falta de covariáveis ou termos no modelo) de uma variável aleatória Poisson.

O contrário à superdispersão é a subdispersão, relatada com menor frequência na literatura. Processos que reduzam a variabilidade de variável aleatória Poisson são pouco conhecidos. Nenhum dos procedimentos descritos acima para superdispersão ou modelos de efeitos aleatórios como os discutidos no Capítulo 3 empregado para modelar subdispersão.

Uma forma de relaxar a suposição de equidispersão é generalizar o processo Poisson. Para isso exploramos a relação que existe entre a distribuição Exponencial e a distribuição de Poisson. De maneira bem simplificada, suponha que temos eventos que ocorrem sob um domínio unidimensional de forma que o intervalo entre eventos tenha distribuição Exponencial. Se dividirmos o domínio em intervalos de igual tamanho, o número de eventos por intervalo terá distribuição de Poisson. Então, para generalizarmos a distribuição do número de eventos, basta atribuirmos outra distribuição para o intervalo entre eventos. Exatamente empregando esse raciocínio que Winkelmann (1995) generalizou a distribuição Poisson empregando a distribuição Gama para o intervalo entre eventos. Entretanto, qualquer outra distribuição de suporte positivo poderia ser empregada. O notável e vantajoso da distribuição Gama é que a distribuição Exponencial é um caso particular dela. Ou seja, a distribuição Gama tem a seguinte função densidade de probabilidade

$$\text{Gama}(y;\alpha,\beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} \exp\{-\beta x\}\, x^{\alpha-1},$$

e para $\alpha = 1$ temos a função densidade de probabilidade da distribuição Exponencial como caso particular

$$\text{Exp}(y;\beta) = \beta \exp\{-\beta x\}.$$

Para entender como as distribuições estão de fato ligadas, pense assim: a variância da Gama é α/β^2, logo, menor valor de α corresponde a menor variância, quer dizer que se concentra uma grande proporção de valores ao redor da média, então intervalos entre eventos serão mais regulares e se dividimos o domínio em intervalos iguais temos um número pouco variável de eventos por intervalo (subdispersão). Por outro lado, maior valor de α corresponde à maior variância, logo, os intervalos entre eventos são mais variáveis/irregulares, de forma que ao dividir o domínio em intervalos temos um número de eventos por intervalo com maior variância (superdispersão).

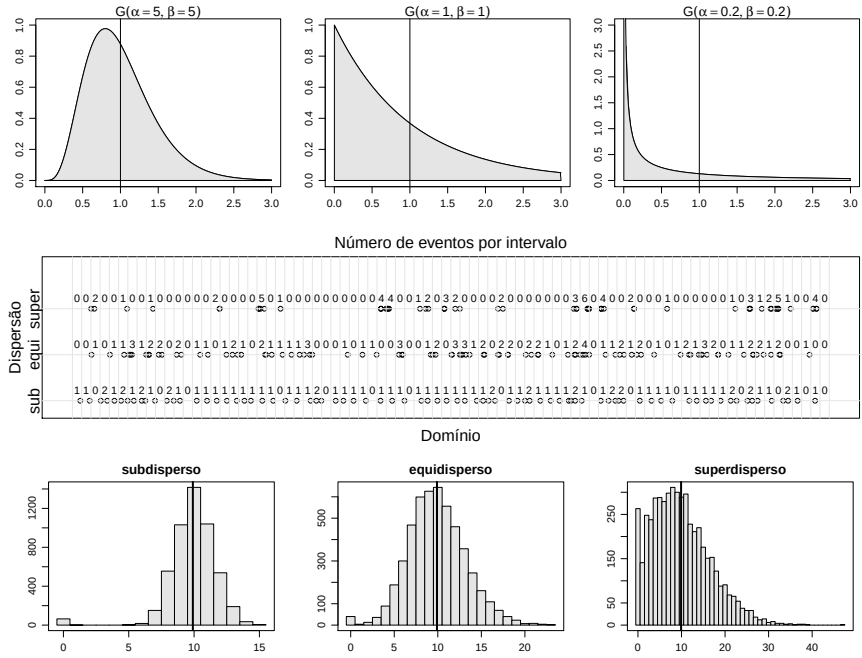

Figura 2.4: (topo) Funções densidade de probabilidade para diferentes distribuições de intervalos entre eventos. (centro) Eventos distribuídos ao longo de um domínio dividido em intervalos com o número de eventos por intervalo representado. (base) Histogramas do número de eventos por intervalo para cada distribuição de intervalo entre eventos.

Na Figura 2.4 foram representados eventos ocorrendo ao longo de um domínio onde a distribuição dos intervalos entre eventos é Gama com média fixa e igual à 1. Em cada um dos casos os parâmetros usados foram sub:$(\alpha = 5, \beta = 5)$, equi:$(\alpha = 1, \beta = 1)$, super:$(\alpha = 0.2, \beta = 0.2)$ que correspondem às variâncias de 0.2, 1 e 5 respectivamente. As funções densidade de probabilidade estão no topo da Figura. No centro da Figura temos o domínio dividido em intervalos unitários e o número de eventos por intervalo representado. Notamos uma distribuição mais uniforme no sub que nos demais o que está de acordo com a argumentação do parágrafo anterior. Para obter os histogramas foram geradas amostras de 50000 elementos e o intervalo teve amplitude de 10 unidades para quantificação do número de eventos (por isso a média passou a ser 10).

Winkelmann (1995) obteve que a função de probabilidades de uma variável aleatória de contagem (N) pode ser escrita como função da distri-

buição acumulada dos intervalos entre eventos (T). Assim, para intervalos Gama, a distribuição de probabilidades do número de eventos por intervalo unitário é dada por

$$P(N = n) = G(1, \alpha n, \beta) - G(1, \alpha(n+1), \beta). \tag{2.3}$$

A distribuição de N é superdispersa para $0 < \alpha < 1$ e subdispersa para $\alpha > 1$.

Uma vez conhecida a função de probabilidade de uma variável aleatória, podemos fazer a estimação dos parâmetros pelo método da máxima verossimilhança. Tomando a equação 2.3 e estendendo para um modelo de regressão obtemos a seguinte função de log-verossimilhança

$$l(y_i; x_i, \alpha, \gamma) = \sum_{i=1}^{n} \log \left(G(1, \alpha y_i, \alpha \exp(x_i^\top \gamma)) - G(1, \alpha(y_i+1), \alpha \exp\{x_i^\top \gamma)\} \right). \tag{2.4}$$

em que y_i é o número de eventos observado na observação i, x_i um vetor de covariáveis conhecidas, γ é o vetor de parâmetros associado as covariáveis x_i. No modelo acima estamos descrevendo a média de T por um modelo de regressão uma vez que $E(T|x_i) = \alpha/\beta = \exp(-x_i^\top \gamma)$ que manipulando resulta em $\beta = \alpha \exp(x_i^\top \gamma)$. A média de N por sua vez pode ser obtida pela seguinte expressão

$$E(N|x_i) = \sum_{i=1}^{\infty} G(\alpha i, \alpha \exp(x_i^\top \gamma)). \tag{2.5}$$

Uma variável de contagem importante do estudo de populações de seres vivos é o número de descendentes produzidos durante a vida. Nas espécies vegetais de importância agrícola, fatores que alteram o número de descendentes também têm impacto na produtividade das culturas. Na cultura do algodão (*Gossypium hirsutum*) a produtividade é função principalmente do número de capulhos produzidos. Ataques de pragas desfolhadoras, por causarem redução da área foliar ou facilitar o desenvolvimento de doenças, podem reduzir a capacidade produtiva da cultura. Para avaliar os efeitos da desfolha, um experimento foi conduzido em casa de vegetação. Em vasos com duas plantas de algodão foram aplicados os níveis de desfolha artificial 0, 25, 50, 75 e 100% de remoção da área foliar (feito com tesoura). A desfolha foi estudada nos estágios fenológicos vegetativo, presença de botão floral, início do florescimento, presença de maça e presença de capulho. Cada combinação desses níveis teve cinco repetições em delineamento completamente casualizado. Ao final do ciclo da cultura, o número de capulhos por vaso foi registrado. Os dados estão no complemento *online* do texto.

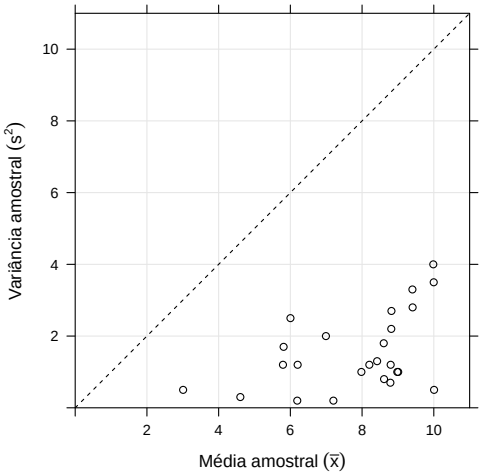

Figura 2.5: Variância amostral em função da média amostral para o número de capulhos do algodão.

A subdispersão é uma característica facilmente observada no número de capulhos. Ao dispor a variância amostral como função das médias amostrais percebemos que todos os valores ficaram abaixo da linha 1:1 (Figura 2.5), portanto, uma evidência contra a suposição de equidispersão. Nessa situação, o emprego do modelo de regressão Poisson não é adequado e o modelo de regressão contagem-Gama, desenvolvido por Winkelmann (1995), será aplicado. A função de log-verossimilhança do modelo está disponível no código 2.31.

Código 2.31: Função de log-verossimilhança para o modelo de regressão contagem-Gama.

```
> ll <- function(theta, y, X){
+   ## theta: vetor de parâmetros/estimativas
+   ## y: vetor de dados observados
+   ## X: matrix de covariáveis associadas aos valores observados
+   eXb <- exp(X%*%theta[-1]) #*theta[1]
+   ## retorna a log verossimilhança para a amostra
+   sum(log(pgamma(1, theta[1]*y, theta[1]*eXb)-
+           pgamma(1, theta[1]*(y+1), theta[1]*eXb)))
+ }
```

Análise exploratória dos dados indica que o valor médio do número de capulhos depende do estágio e do nível de desfolha de forma não aditiva.

Assim, usamos um modelo de interação entre os fatores com polinômio de segundo grau para efeito do nível de desfolha. Dessa maneira, a matriz do modelo é construida por meio da função `model.matrix()`. Esse modelo considera a estimação de um vetor de 11 parâmetros: intercepto único, efeito linear e efeito quadrático de desfolha por nível de estágio. A estimação usando `optim()` exige valores iniciais para iniciar o algoritmo. Um conjunto de bons valores iniciais são as estimativas dos parâmetros para o modelo de regressão de Poisson, que obtemos no R com a função `glm()`.

```
> X <- model.matrix(~est:(des+I(des^2)), data=cap)
> ## estimação modelo Poisson
> rp <- glm(nc~est:(des+I(des^2)), data=cap, family=poisson)
> ## estimação modelo Contagem Gamma
> op <- optim(c(alpha=1, coef(rp)), ll, y=cap$nc,
+             X=X, hessian=TRUE, method="BFGS",
+             control=list(fnscale=-1))
> # 2*diferença da log-verossimilhança
> dll <- c(diff.ll=2*abs(op$value-c(logLik(rp)))); dll
 diff.ll
94.83326
```

Ao comparar as verossimilhanças do modelo Poisson com o contagem-Gama estamos testanto a hipótese do parâmetro $\alpha = 1$. Vemos que essa hipótese foi rejeitada. De forma complementar ao teste de hipótese, podemos obter o perfil de log-verossimilhança para o parâmetro α e construir intervalos de confiança. Para isso escrevemos outra função de log-verossimilhança onde α é fixo e a estimação se dá apenas para os demais parâmentros do modelo (código 2.32). Para fins de comparação, construimos intervalos de confiança baseados na aproximação quadrática da função de log-verossilhança para avaliar a qualidade da aproximação. Nos códigos abaixo tais procedimentos são ilustrados. A função `uniroot.all()` do pacote **rootSolve** é empregada para obter os intervalos de confiança baseados no perfil da deviance.

Código 2.32: Função de perfil de log-verossimilhança para o parâmetro α do modelo de regressão contagem-Gama.

```
> ll.alpha <- function(theta, alpha, y, X){
+   ## theta: vetor de parâmetros/estimativas
+   ## alpha: escalar do parâmetro alpha fixo
+   ## y: vetor de dados observados
+   ## X: matrix de covariáveis associadas aos valores observados
+   eXb <- exp(X%*%theta) #*theta[1]
+   ## retorna a log verossimilhança com um valor FIXO de ALPHA
+   sum(log(pgamma(1, alpha*y, alpha*eXb)-
+           pgamma(1, alpha*y+alpha, alpha*eXb)))
+ }
```

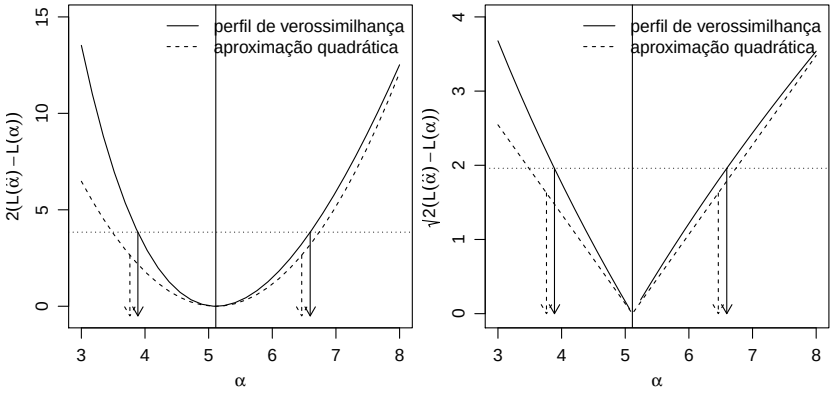

Figura 2.6: Perfil de log-verossimilhança representado pelo valor na diferença na deviance (esq.) e a raíz da diferença na *deviance* (dir.) para o parâmetro α. Setas indicam os limites do intervalo de confiança.

```
> alpha <- sort(c(seq(3,8,l=30), op$par[1])) # valores para alpha
> perfil <- sapply(alpha,
+                  function(a){
+                   op <- optim(coef(rp), ll.alpha, alpha=a, y=cap$nc, X=X,
+                               method="BFGS", control=list(fnscale=-1))
+                   c(op$value, op$par[1])
+                   })
> coef <- op$par; vcov <- -solve(op$hessian); llik <- op$value
> alp <- coef["alpha"]; sd.alp <- sqrt(vcov["alpha","alpha"])
> dev.perf <- 2*(llik-perfil[1,])  # deviance da log-ver perfilhada
> dev.quad <- (alp-alpha)^2/sd.alp # deviance da aprox quadrática
> require(rootSolve)
> qchi <- qchisq(0.95, df=1)
> fperf <- approxfun(alpha, dev.perf-qchi)
> lim <- uniroot.all(fperf, c(0, 10)) # limites do IC perf
> lim2 <- alp+c(-1,1)*1.96*sd.alp     # limites do IC assint
```

O perfil de log-verossimilhança para α está representado na Figura 2.6. A aproximação quadrática apresentou intervalos de mesma amplitude daqueles obtidos pelo perfil de verossimilhança, porém com um leve deslocamento para a esquerda. Nenhum dos intervalos contém o valor sob a hipótese nula e portanto os dados apresentam subdispersão. Assim, o número de capulhos por algodão apresenta um distribuição mais regular que aquela prevista pela distribuição de Poisson.

Os valores esperados para o número de capulhos foram obtidos pela equação 2.5, bem como os limites do intervalo de predição (Figura 2.7).

As estimativas dos parâmetros do modelo indicam que não existe efeito de desfolha no estágio presença de capulhos. No florescimento a desfolha

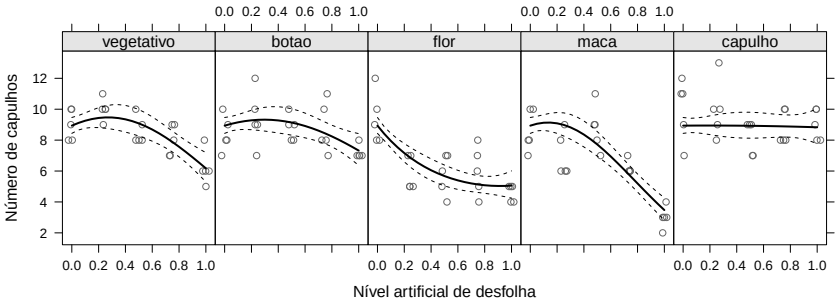

Figura 2.7: Valores observados e preditos acompanhados do intervalo de confiança de 95% para o número de capulhos do algodão em função do nível de desfolha e estágio fenológico.

reduz o número de capulhos à taxas decrescentes, o impacto é maior para leves desfolhas e a partir de 70% não há mais considerável redução. Nos demais estágios observa-se uma tolerância à até 30% de desfolha sem prejuízo aos capulhos, acima disso ocorre redução à taxas crescentes.

```
> tabcoef <- data.frame(Estimativas=coef, ErroPadrão=sqrt(diag(vcov)))
> tabcoef$zvalor <- with(tabcoef, Estimativas/ErroPadrão)
> tabcoef$pvalor <- with(tabcoef, pnorm(abs(zvalor), lower=FALSE)*2)
> tabcoef
```

	Estimativas	ErroPadrão	zvalor	pvalor
alpha	5.112297805	0.68872752	7.42281618	1.146559e-13
(Intercept)	2.234239342	0.02802741	79.71622029	0.000000e+00
estvegetativo:des	0.412024360	0.22796029	1.80743922	7.069382e-02
estbotao:des	0.274377741	0.22448099	1.22227609	2.216032e-01
estflor:des	-1.182180751	0.26654192	-4.43525263	9.196438e-06
estmaca:des	0.319589495	0.24988237	1.27895977	2.009112e-01
estcapulho:des	0.007104167	0.22267231	0.03190413	9.745485e-01
estvegetativo:I(des^2)	-0.762638914	0.25818749	-2.95381824	3.138688e-03
estbotao:I(des^2)	-0.464149443	0.25044536	-1.85329623	6.383991e-02
estflor:I(des^2)	0.645341332	0.30030943	2.14892127	3.164064e-02
estmaca:I(des^2)	-1.198887094	0.29689851	-4.03803680	5.390040e-05
estcapulho:I(des^2)	-0.018586566	0.24424267	-0.07609877	9.393405e-01

Capítulo 3

Modelos de regressão com efeitos aleatórios

A área de modelagem estatística teve um grande impulso com a criação dos modelos de regressão, dos quais os ilustrados no Capítulo 2 são alguns exemplos. As aplicações aparecem nas mais diversas áreas da ciência. Nesta diversidade de aplicações é muito fácil encontrar situações de relevância prática nas quais os modelos de regressão tradicionais deixam de ser adequados pela existência de características que violam suposições de modelos usuais. Alguns exemplos são:

- para covariáveis contínuas, a suposição de efeito estritamente linear no preditor pode não ser adequada,
- as observações podem ser correlacionadas no espaço,
- as observações podem ser correlacionadas no tempo,
- interações complexas podem ser necessárias para modelar o efeito conjunto de algumas covariáveis,
- heterogeneidade entre indivíduos ou unidades podem não ser suficientemente descrita por covariáveis.

Em tais situações a variabilidade das observações usualmente não segue a prescrita pelo modelo de probabilidades e a classe de modelos de regressão é estendida pela adição de efeitos aleatórios, incorporando variáveis não observadas (latentes). Esta abordagem é extensivamente utilizada por ser altamente flexível mas ainda preservando especificações de modelos entre distribuições conhecidas e tratáveis analítica e/ou computacionalmente. Esta classe de modelos pode facilmente ser descrita como

uma extensão dos modelos de regressão com efeitos fixos, pela inclusão de mais uma suposição. Considere que Y seja um vetor de dimensão n. A seguinte estrutura hierárquica descreve um modelo de regressão com efeitos aleatórios. Neste livro nos vamos nos limitar a inclusão de efeitos aleatórios gaussianos, como na seguinte especificação:

$$[Y|b, X] \sim f(\underline{\mu}, \phi)g(\underline{\mu}) \qquad = X\underline{\beta} + Z\underline{b}\underline{b} \sim NMV(\underline{0}, \Sigma).$$

O preditor linear é decomposto em duas componentes, a parte de efeitos fixos $X\underline{\beta}$ e a parte aleatória $Z\underline{b}$. As matrizes de delineamento X e Z são consideradas conhecidas representando os efeitos de covariáveis de interesse. O vetor $\underline{\beta}$ representa os efeitos fixos que deverão ser estimados. O vetor aleatório $\underline{b}$ são quantidades não observadas (latentes) para o qual vamos atribuir uma distribuição gaussiana multivariada de média 0 e matriz de covariância Σ. De acordo com a estrutura imposta a Σ podemos induzir diversos tipos de correlação entre as observações Y. É usual adotar a suposição de independência condicional em $[Y|b]$, ou seja, dado os efeitos aleatórios as observações são independentes, o que é usado na construção da verossimilhança e explorado por algoritmos numéricos.

A inferência baseada em verossimilhança para esta classe de modelos apresenta desafios computacionais, principalmente quando a distribuição atribuída a variável resposta é diferente da gaussiana. Na discussão a seguir vamos, sem perda de generalidade, excluir a matriz de efeitos fixos X. Como temos duas quantidades aleatórias, devemos obter a distribuição conjunta $[Y,b]$ que pode, seguindo a estrutura hierárquica do modelo, ser fatorada na forma $[Y,b] = [Y|b][b]$. Entretanto apenas Y é observável e portanto a verossimilhança é dada pela distribuição marginal $[Y]$ que é obtida fazendo uma média sobre os valores da variável não observadas $[Y] = \int [Y|b][b]db$. As estimativas são obtidas maximizando de $[Y]$ em relação aos parâmetros do modelo. Sob a suposição de que $[b]$ é gaussiana multivariada, temos que os parâmetros do modelos são $[\underline{\beta}, \Sigma, \phi]$, ou seja, o vetor de efeitos fixos mais os parâmetros que indexam a distribuição do efeito aleatório tipicamente com algum parâmetro de variabilidade ou precisão.

Quando a distribuição de $[Y]$ não é gaussiana, não é possível resolver analiticamente a a integral contida na função de verossimilhança e temos que recorrer a métodos numéricos. Isto implica que métodos de integração numérica são necessários para avaliar a verossimilhança marginal e obter as estimativas de máxima verossimilhança. Esta descrição é bastante genérica e será detalhada na sequência. Mas antes vamos apresentar um exemplo onde a distribuição de $[Y]$ é gaussiana porém as amostras não são independentes.

3.1 Modelo geoestatístico

O termo geoestatística, refere-se a modelos e métodos para dados seguindo as seguintes características: os valores $Y_i : i = 1, \ldots, n$ são observados em um conjunto discreto de localizações amostrais, x_i, em alguma região espacial A. Cada valor é uma versão ruidosa de um fenômeno espacial contínuo não observável, denotado por $S(x)$, nas correspondentes localizações amostrais. Em nossa notação X é a matriz com covariáveis e x denota a posição (coordenada). O objetivo mais comum neste tipo de análise é recuperar o processo $S(x)$. Para isto, o modelo geoestatístico, como apresentado em Diggle & Ribeiro Jr. (2007) é definido da seguinte forma para respostas gaussianas:

$$[Y|b, X] \sim N(\underline{\mu}, I\tau^2)\underline{\mu} \qquad = X\underline{\beta} + S(x)S(\underline{x}) \sim NMV(\underline{0}, \Sigma_S)$$

em que X é uma matriz de covariáveis conhecidas e $\underline{\beta}$ o vetor de parâmetros associados a elas. Para um conjunto de observações $S(\underline{x})$ são os valores do efeito aleatório em cada posição Portanto o modelo geoestatístico desta forma pode ser interpretado como um modelo com interceptos aleatórios, com diferentes intercepto em cada posição. A parte deste modelo que merece mais atenção é a matriz Σ_S, pois ela descreve a estrutura da dependência espacial entre as observações. Os elementos desta matriz são dados por uma função de correlação cujo argumento é a distância entre cada par de observações. Diversas opções para construir esta matriz são possíveis, neste exemplo vamos usar a função de correlação exponencial. A matriz considerando apenas duas localizações espaciais, toma a seguinte forma:

$$\Sigma_S = \begin{bmatrix} \sigma^2 & \sigma^2 \exp(-u_{12}/\phi) \\ \sigma^2 \exp(-u_{21}/\phi) & \sigma^2 \end{bmatrix}$$

em que u_{12} é a distância euclidiana entre as posições espaciais das variáveis $Y(x_1)$ e $Y(x_2)$. A matriz completa depende da matriz de distância entre todas as posições espaciais na amostra e dos parâmetros σ^2 e ϕ. O parâmetro ϕ controla a extensão da dependência espacial entre as observações, enquanto que o parâmetro σ^2 é a variância deste efeito. Além disso, o modelo conta com o parâmetro τ^2 que é a variância das observações Y, além dos parâmetros associados a média $\underline{\beta}$.

A inferência para os parâmetros envolvidos no modelo $\underline{\theta} = (\underline{\beta}, \sigma^2, \tau^2\phi)$ pode ser feita baseado na função de verossimilhança, que neste caso toma a forma de uma distribuição normal multivariada e portanto $Y \sim NMV(X\underline{\beta}, \Sigma)$ onde Σ é da forma:

$$\Sigma = \begin{bmatrix} \sigma^2 + \tau^2 & \sigma^2 \exp(-\frac{u_{12}}{\phi}) \\ \sigma^2 \exp(-\frac{u_{21}}{\phi}) & \sigma^2 + \tau^2 \end{bmatrix}.$$

Com isso, tem-se a função de verossimilhança,

$$L(\underline{\theta}) = (2\pi)^{-\frac{n}{2}}|\Sigma|^{-\frac{1}{2}}\exp\{-\frac{1}{2}(Y - X\underline{\beta})^\top\Sigma^{-1}(Y - X\underline{\beta})\}.$$

A função de log-verossimilhança,

$$l(\underline{\theta}) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(Y - X\underline{\beta})^\top\Sigma^{-1}(Y - X\underline{\beta}). \tag{3.1}$$

Para estimação dos parâmetros, o objetivo agora é maximizar a função de log-verossimilhança em relação a $\underline{\theta}$. Note que temos dois conjuntos de parâmetros, os associados a média $\underline{\beta}$ e os associados a estrutura de variância e covariância (σ^2, τ^2, ϕ). Note ainda que a log-verossimilhança pode facilmente ser derivada em função de $\underline{\beta}$. Já para o caso dos parâmetros que indexam a matriz Σ, exceto por um parâmetros de escala, a derivação não é tão trivial e vai depender do modelo da função de correlação.

Derivando a função 3.1 em relação aos $\underline{\beta}$ e igualando a zero, chegamos aos estimadores de máxima verossimilhança.

$$\underline{\hat{\beta}} = (X^\top\Sigma^{-1}X)^{-1}(X^\top\Sigma^{-1}Y) \tag{3.2}$$

Substituindo a expressão em 3.2 na equação em 3.1 chegamos a função de log-verossimilhança concentrada apenas nos parâmetros que definem a estrutura de variância e covariância do modelo, denote isto por $\underline{\theta}^* = (\sigma^2, \tau^2, \phi)$.

$$l^*(\underline{\theta}^*) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}\hat{e}^\top\Sigma^{-1}\hat{e}$$

onde $\hat{e} = (Y - X\underline{\hat{\beta}})$. Os três parâmetros restantes estão dentro da matriz Σ, logo é necessário derivá-los usando cálculo matricial. É possível mostrar que a função escore é dada por,

$$\frac{\partial l^*(\underline{\theta}^*; \underline{Y})}{\partial\theta_i^*} = -\frac{1}{2}Tr\left[\Sigma^{-1}\frac{\partial\Sigma}{\partial\theta_i^*}\right] - \frac{1}{2}\underline{\hat{e}}^\top\left[-\Sigma^{-1}\frac{\partial\Sigma}{\partial\theta_i^*}\Sigma^{-1}\right]\underline{\hat{e}}, \quad i = 1, \ldots, 3.$$

onde as matrizes $\dfrac{\partial\Sigma}{\partial\theta_i^*}$ representa a matriz obtida por derivar cada elemento da matriz Σ em relação ao respectivo parâmetro. Para exemplificar, considere o caso de apenas duas observações, a derivada de Σ em relação ao parâmetro σ é

$$\frac{\partial\Sigma}{\partial\sigma} = \begin{bmatrix} 2\sigma & 2\sigma\exp(-\frac{u_{12}}{\phi}) \\ 2\sigma\exp(-\frac{u_{21}}{\phi}) & 2\sigma \end{bmatrix}.$$

Derivando agora em relação ao parâmetro τ,

$$\frac{\partial \Sigma}{\partial \tau} = \begin{bmatrix} 2\tau & 0 \\ 0 & 2\tau \end{bmatrix},$$

finalmente em relação ao parâmetro ϕ

$$\frac{\partial \Sigma}{\partial \phi} = \begin{bmatrix} 0 & \sigma^2 u_{12} \exp(-\frac{u_{12}}{\phi})/\phi^2 \\ \sigma^2 u_{21} \exp(-\frac{u_{21}}{\phi})/\phi^2 & 0 \end{bmatrix}.$$

Dado estes resultados estamos aptos a começar a implementação do modelo geoestatístico. Quando trabalhamos com modelos espaciais a parte que requer mais cuidado é a matriz de variância/covariância, então começamos a implementação por ela. A função abaixo chamada `monta.sigma()` monta a estrutura da matriz Σ do modelo geoestatístico.

Código 3.33: Função para a matriz de variância-covariância para o modelo geoestatístico segundo uma função de correlação exponencial.

```
> monta.sigma <- function(s, t, phi, U){
+   Sigma <- as.matrix(s^2 * exp(-U/phi))
+   diag(Sigma) <- s^2 + t^2
+   return(Sigma)
+ }
```

Sempre que implementamos um modelo complexo é interessante simular dados dele para nos certificarmos que o processo de inferência esta correto. Para a simulação vamos supor que a estrutura de média é composta de apenas um parâmetro $\beta_0 = 50$, vamos fixar os parâmetros $\sigma^2 = 2$, $\tau^2 = 1$ e $\phi = 0.25$, os dados serão simulados dentro de um quadrado unitário em uma grade regular.

Código 3.34: Função para simular variáveis aleatórias segundo um modelo geoestatístico.

```
> simula.geo <- function(b, s, t, phi, n.simul){
+ coord.X <- seq(0,1,l=sqrt(n.simul))
+ coord.Y <- seq(0,1,l=sqrt(n.simul))
+ grid.simul <- expand.grid(coord.X, coord.Y)
+ U <- dist(grid.simul, diag=TRUE, upper=TRUE)
+ Sigma <- monta.sigma(s=s, t=t, phi=phi, U=U)
+ Z <- rnorm(dim(Sigma)[1])
+ Y = crossprod(chol(Sigma), Z) + b
+ dados <- data.frame(Y=Y, coord.X=grid.simul[,1],
+                     coord.Y=grid.simul[,2])
+ return(dados)
+ }
```

Na função `simula.geo()` começamos criando as coordenadas X e Y e fazendo todas as combinações entre essas, o que gera uma grade regular. Após criar a grade calculamos a matriz de distância entre todos os pontos, chamamos a matriz de U. Com a matriz U e os parâmetros criamos a matriz Σ, simulamos dados de uma Normal independente e aplicamos uma transformação multiplicando o vetor Z pela Cholesky da matriz Σ que induz a dependência entre as observações. Desta forma, temos dados do modelo geoestatístico usando a função de correlação exponencial.

O próximo passo para a inferência é a implementação da função de log-verossimilhança concentrada.

Código 3.35: Função de log-verossimilhança para o modelo geoestatístico.

```
> ll.geo <- function(s,t,phi,dados){
+  U <- dist(dados[,2:3],diag=TRUE, upper=TRUE)
+  Sigma <- monta.sigma(s=s, t=t, phi=phi, U=U)
+  X <- as.vector(rep(1,l=length(dados$Y)))
+  invSX <- solve(Sigma,X)
+  beta.hat <-solve(crossprod(invSX, X),crossprod(invSX,dados$Y))
+  ll = dmvnorm(dados$Y, mean=X%*%beta.hat,sigma=Sigma, log=TRUE)
+  return(-ll)
+ }
```

A partir deste ponto podemos maximizar a log-verossimilhança diretamente através da `optim()` ou qualquer outra forma de maximização numérica. Porém, como obtemos o gradiente analítico vamos utiliza-lo para ajudar no algoritmo de maximização numérica e depois vamos comparar o tempo computacional com e sem o uso do gradiente analítico.

Para implementar o gradiente precisamos primeiro implementar três funções que implementam cada uma das matrizes de derivadas em relação a cada um dos parâmetros que indexam a matriz Σ.

Código 3.36: Funções para a derivada dos componentes da matriz de variância-covariância do modelo geoestatístico.

```
> ## Derivada de sigma, tau e phi
> deriv.s <- function(s, t, phi, U){
+   Sigma.s <- 2*s*as.matrix(exp(-U/phi))
+   diag(Sigma.s) <- 2*s
+   return(Sigma.s)}
> deriv.t <- function(s, t, phi, U){
+   dimensao <- dim(as.matrix(U))
+   Sigma.t <- matrix(0, ncol=dimensao[1], nrow=dimensao[2])
+   diag(Sigma.t) <- 2*t
+   return(Sigma.t)
+ }
> deriv.phi <- function(s, t, phi, U){
+   Sigma.phi <-s^2* as.matrix(U) * as.matrix(exp(-U/phi))/phi^2
+   return(as.matrix(Sigma.phi))
+ }
```

E agora podemos escrever a função escore completa.

Código 3.37: Função escore para o modelo geoestatístico.

```
> escore <- function(s, t, phi, dados){
+  U <- dist(dados[,2:3], diag=TRUE, upper=TRUE)
+  Sigma <- monta.sigma(s=s, t=t, phi=phi, U=U)
+  X <- rep(1,l=length(dados$Y))
+  invSX <- solve(Sigma, X)
+  beta.hat <- solve(crossprod(invSX, X), crossprod(invSX,dados$Y))
+  e.hat <- dados$Y - X%*%beta.hat
+  Sigma1 <- solve(Sigma)
+ ## Escores de sigma, tau e phi
+  S1D <- Sigma1 %*% deriv.s(s=s,t=t,phi=phi,U=U)
+  U.s <- 0.5*(sum(diag(S1D))-t(e.hat)%*%(S1D%*%Sigma1)%*%e.hat)
+  T1D <- Sigma1 %*% deriv.t(s=s,t=t,phi=phi,U=U)
+  U.t <- 0.5*(sum(diag(T1D))-t(e.hat)%*%(T1D%*%Sigma1)%*%e.hat)
+  P1D <- Sigma1%*%deriv.phi(s=s,t=t,phi=phi,U=U)
+  U.phi <- 0.5*(sum(diag(P1D))-t(e.hat)%*%(P1D%*%Sigma1)%*%e.hat)
+  return(c(U.s, U.t, U.phi))
+ }
```

Um detalhe computacional tanto na função escore como na log-verossimilhança concentrada é que na saída da função pedimos para que seja retornado o negativo da função, isso se deve apenas para compatibilidade com a função `mle2()` que por *default* minimiza a função objetivo, no caso a log-verossimilhança.

Com tudo implementado podemos simular um conjunto de dados do modelo e ajustar usando a função `mle2()` do pacote **bbmle** por conveniên-

cia. Poderíamos usar diretamente a função `optim()` com qualquer um de seus algoritmos, ou mesmo outros maximizadores residentes no R.

A seguinte chamada simula 225 amostras do modelo geoestatístico.

```
> set.seed(12)
> dados <- simula.geo(b=50, s=2, t=1, phi=0.25, n.simul=225)
```

Estimando os parâmetros sem o uso do gradiente analítico pelo algoritmo `L-BFGS-B`.

```
> require(bbmle)
> require(mvtnorm)
> system.time(modelo <- mle2(ll.geo, start=list(s=1,t=0.1,phi=0.1),
+                            method="L-BFGS-B",
+                            lower=list(s=1e-32, t=1e-32,phi=1e-32),
+                            upper=list(s=Inf, t= Inf, phi = Inf),
+                            data=list(dados=dados)))
   user  system elapsed
 50.643   0.684  51.882
```

A mesma chamada anterior com o uso do gradiente analítico.

```
> system.time(modelo.escore <- mle2(ll.geo, gr= escore,
+                   start=list(s=1,t=0.1,phi=0.1), method="L-BFGS-B",
+                   lower=list(s=1e-32, t=1e-32,phi=1e-32),
+                   upper=list(s=Inf, t= Inf, phi = Inf),
+                   data=list(dados=dados)))
   user  system elapsed
 23.853   0.284  24.244
```

De acordo com o tempo computacional exigido por cada chamada podemos verificar que o uso do gradiente analítico diminui o tempo para a maximização da log-verossimilhança praticamente pela metade. Este é um ganho expressivo, principalmente quando trabalha-se com grandes bases de dados e/ou procedimentos computacionalmente intensivos. Informações do ajuste são resumidas a seguir.

```
> summary(modelo.escore)
Maximum likelihood estimation

Call:
mle2(minuslogl = ll.geo, start = list(s = 1, t = 0.1, phi = 0.1),
    method = "L-BFGS-B", data = list(dados = dados), gr = escore,
    lower = list(s = 1e-32, t = 1e-32, phi = 1e-32), upper = list(s = Inf,
        t = Inf, phi = Inf))

Coefficients:
    Estimate Std. Error z value     Pr(z)
s   1.581869   0.229672  6.8875 5.677e-12 ***
t   0.995908   0.158835  6.2701 3.609e-10 ***
phi 0.184622   0.082826  2.2290   0.02581 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

-2 log L: 819.527
```

Deixamos para o leitor encontrar as estimativas do vetor β, bem como seus respectivos intervalos de confiança. Para finalizar o procedimento

de inferência no modelo geoestatístico a Figura 3.1 apresenta os perfis de verossimilhança para os parâmetros que indexam a matriz de variância/covariância do modelo. Note a forte assimetria em todos os parâmetros. O intervalo perfilhado para o parâmetro τ bate na borda esquerda do espaço paramétrico com um nível de confiança de aproximadamente 80%, este tipo de problemas não é incomum em modelos com efeitos espaciais. Pode-se tentar contornar isso fazendo reparametrizações, por exemplo estimar o $\log(\tau)$ ou alguma outra função que seja adequada. O importante destacar é que não há uma receita geral que vai funcionar bem todas as vezes, em cada modelo diferentes reparametrizações podem ser tentadas até que se chegue a uma satisfatória.

```
> perfil <- profile(modelo)
> save(perfil,file="perfilGeo.RData")
```

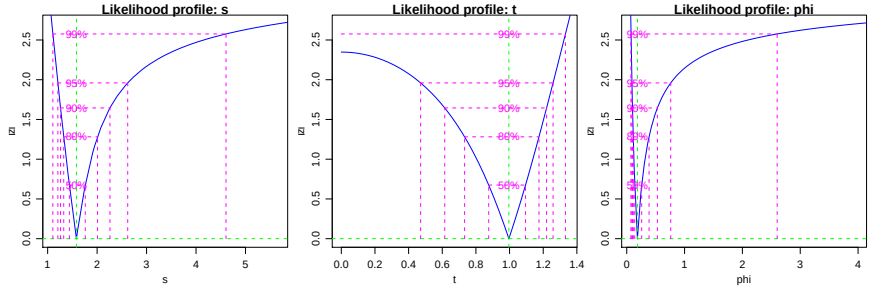

Figura 3.1: Perfis de verossimilhança modelo geoestatístico.

3.2 Verossimilhança Marginal

Os modelos mistos lineares generalizados são os modelos de regressão com efeitos aleatórios mais comumente usados, que estendem os mistos com resposta gaussiana para distribuições na família exponencial como nos modelos lineares generalizados. O objetivo desta Seção é descrever a formulação de um modelo de regressão com efeitos aleatórios de uma forma geral. Modelos para dados longitudinais, medidas repetidas, modelos com efeitos espaciais, temporais e espaço temporais possam ser descritos todos na mesma *estrutura* tratando de uma forma unificada uma ampla classe de modelos estatísticos.

A seguir, vamos descrever o modelo no contexto de dados agrupados e conforme necessário vamos indicando pequenas mudanças que geram modelos conhecidos, por exemplo, para dados longitudinais e espaciais. Seja Y_{ij} a j-ésima medida para a unidade amostral $i, i = 1, \ldots, N$, $j = 1, \ldots, n_i$ e $\underline{Y}_i$ o vetor n_i-dimensional de todas as medidas realizadas na unidade amostral i. Assumindo independência condicional no vetor q-dimensional de

efeitos aleatórios $\underline{b}_i$, para o qual atribuímos uma distribuição $NMV_q(\underline{0}, \Sigma)$, as respostas Y_{ij} são independentes com densidade da forma,

$$f_i(y_{ij}|\underline{b}_i, \underline{\beta}, \phi),$$

com $g(\mu_{ij}) = \mathbf{x}_{ij}^T\underline{\boldsymbol{\beta}} + \mathbf{z}_{ij}^T\underline{\mathbf{b_i}}$ para uma função de ligação $g(\cdot)$ conhecida, com $\mathbf{x}_{ij}$ e $\mathbf{z}_{ij}$ vetor de covariáveis conhecidas de dimensão p e q respectivamente, $\underline{\boldsymbol{\beta}}$ um vetor p-dimensional de coeficientes de regressão fixos desconhecidos, e ϕ algum parâmetro extra na verossimilhança, geralmente indicando precisão ou variância. Para completar a especificação do modelo, seja $f(\underline{\mathbf{b_i}}|\Sigma)$ a densidade da $NMV_q(\mathbf{0},\Sigma)$ distribuição atribuída para os efeitos aleatórios $\underline{\mathbf{b_i}}$.

Como já mencionado, a estimação dos parâmetros envolvidos no modelo pode ser feita maximizando a verossimilhança marginal, com a integração dos efeitos aleatórios. A contribuição para a verossimilhança da cada unidade amostral (grupo) é:

$$f_i(\underline{y}_i|\underline{\beta}, \Sigma, \phi) = \int \prod_{j=1}^{n_i} f_{ij}(y_{ij}|\underline{b}_i, \underline{\beta}, \phi) f(\underline{b}_i|\Sigma) d\underline{b}_i,$$

a verossimilhança completa para $\underline{\beta}$, Σ e ϕ é dada por

$$L(\underline{\beta}, \Sigma, \phi) = \prod_{i=1}^{N} f_i(\underline{y}_i|\underline{\beta}, \Sigma, \phi), \tag{3.3}$$

e sob a suposição de independência entre os grupos temos que

$$L(\underline{\beta}, \Sigma, \phi) = \prod_{i=1}^{N} \int \prod_{j=1}^{n_i} f_{ij}(y_{ij}|\underline{b}_i, \underline{\beta}, \phi) f(\underline{b}_i|\Sigma) d\underline{b}_i. \tag{3.4}$$

A principal dificuldade em maximizar 3.3 é a presença das N integrais sobre os efeitos aleatórios q-dimensionais. Em alguns casos especiais estas integrais podem ser resolvidas analiticamente, como no caso do modelo geoestatístico em 3.1 e, de forma geral, modelos com resposta gaussiano. Porém, na maioria das situações onde a resposta é não gaussiana as integrais envolvidas no cálculo da função de verossimilhança não tem solução analítica.

Além disso, um problema adicional é a dimensão do vetor de efeitos aleatórios. Quando q é pequeno, o que acontece em modelos de regressão com somente o intercepto aleatório ($q = 1$) ou inclinação aleatória ($q = 2$ para uma única covariável) as integrais são passíveis de ser resolvidas por métodos de integração numérica convencionais, como Gauss-Hermite, Laplace e Monte Carlo. Estes métodos serão abordados na sequência. Porém,

em alguns modelos, como por exemplo os modelos espaciais, a dimensão do vetor aleatório pode chegar a $q = N$, ou seja, o vetor tem a dimensão do tamanho da amostra, possivelmente grande, o que torna os métodos convencionais de integração numérica não aplicáveis. Métodos de integração numérica como Gauss-Hermite e Monte Carlo vão ser úteis quando a dimensão do vetor de efeitos aleatórios é pequena, digamos, menor que seis. Para efeitos aleatórios de maior dimensão uma implementação muito cuidadosa do método de Laplace pode ser adequada em algumas situações.

Em modelos onde o vetor de efeitos aleatórios é de grande dimensão o mais usual é lançar mão de métodos que avaliam a verossimilhança por algoritmos de amostragem. Neste contexto os métodos MCMC - Monte Carlo via Cadeias de Markov - são extremamente poderosos para ajustar modelos de alta complexidade podendo ser usados para inferência baseada apenas na função de verossimilhança, ou na sua forma mais usual, sob o paradigma bayesiano.

Na sequência vamos apresentar alguns métodos tradicionais de integração numérica, que podem ser usados quando ajustamos modelos de efeitos aleatórios de baixa complexidade na estrutura aleatória. Os métodos serão aplicadas na estimação de alguns modelos simples para medidas repetidas e dados longitudinais não gaussianos. Vamos iniciar com um modelo simple mais que serve de exemplo para apresentação dos métodos.

3.2.1 Simulação de modelo Poisson com intercepto aleatório

Em todos os métodos de integração numérica que serão apresentados vamos utilizar o modelo de Poisson com intercepto aleatório para exemplificar o cálculo numérico. Para isto, precisamos de amostras deste modelo para podermos avaliar a função de verossimilhança para uma dada configuração de parâmetros, que é o objetivo final do uso dos métodos de integração numérica em modelos de regressão com efeitos aleatórios. A função `simPois()` simula amostras deste modelo de acordo com a parametrização usada.

Código 3.38: Função para simular variáveis aleatórias de um modelo de Poisson com efeito aleatório de intercepto.

```
> simPois <- function(f.fixo, f.aleat, beta.fixo, prec.pars, data){
+   X <- model.matrix(f.fixo, data)
+   Z <- model.matrix(f.aleat, data)
+   n.bloco <- ncol(Z)
+   n.rep <- nrow(Z)/n.bloco
+   bi <- rnorm(n.bloco,0,sd=1/prec.pars)
+   XZ <- cbind(X,Z)
+   beta <- c(beta.fixo,bi)
+   preditor <- XZ%*%beta
+   lambda <- exp(preditor)
+   y <- rpois(length(lambda),lambda=lambda)
+   return(cbind(y=y, data))
+ }
```

Para simular do modelo precisamos das matrizes de delineamento X e Z e dos parâmetros β_0 e τ. De acordo com o tamanho das matrizes X e Z a função identifica quantas unidades amostrais e quantas repetições por unidade amostral deve ser simulada. Feita a função podemos usá-la.

```
> dt <- data.frame(ID=as.factor(rep(1:10,each=10)))
> set.seed(123)
> dados <- simPois(f.fixo=~1, f.aleat=~-1 + ID,
+                  beta.fixo = 2, prec.pars=3, data=dt)
```

De acordo com o código acima, foram simulados 10 unidades amostrais em cada uma destas unidades são retiradas 10 amostras totalizando 100 observações. O modelo tem uma média geral igual a 2 e atribui a cada grupo um desvio (efeito aleatório) deste valor. Neste exemplo, para cada avaliação da verossimilhança devemos resolver 10 integrais uma para cada unidade amostral. Nos exemplos, vamos usar apenas uma unidade amostral e fixar os parâmetros nos valores simulados para avaliar a integral. A função integrando escrita de forma vetorial, fica dada por:

Código 3.39: Integrando da função de verossimilhança do modelo de regressão de Poisson com efeito aleatório de intercepto.

```
> integrando <- function(b, f.fixo, beta.fixo, prec.pars,
+                        log=TRUE, dados){
+   mf <- model.frame(f.fixo, dados)
+   y <- model.response(mf)
+   X <- model.matrix(f.fixo, mf)
+   tau <- exp(prec.pars)
+   ll <- sapply(b,function(bi){
+      preditor <- X%*%beta.fixo + bi
+      lambda <- exp(preditor)
+      sum(dpois(y, lambda=lambda, log=TRUE)) +
+                dnorm(bi, 0, sd=1/tau,log=TRUE)})
+   if(log == FALSE) ll <- exp(ll)
+   return(ll)
+ }
```

Escrever a função em forma vetorial, significa simplesmente que podemos passar em vetor de valores que a função será avaliada em cada um destes valores. Outro fato importante na forma de escrever o integrando é fazer o máximo possível das operações em escala logarítmica. Isso evita problemas com representações numéricas. Porém devemos sempre ter em mente que estamos calculando a integral na escala original e nunca em logaritmo. Por exemplo,

```
> ## Escala original
> integrando(b=c(-1,0,1), f.fixo = y~1, dados = subset(dados, ID == 1),
+            beta.fixo = 2, prec.pars=4, log=FALSE)
[1] 0.000000e+00 4.011525e-09 0.000000e+00

> ## Escala log
> integrando(b=c(-1,0,1), f.fixo = y~1, dados = subset(dados, ID == 1),
+            beta.fixo = 2, prec.pars=4, log=TRUE)
[1] -1535.10535   -19.33409 -1564.77790
```

O formato vetorial da função facilita a construção de algoritmos para integração numérica. É conveniente fazer um gráfico da função integrando para termos uma ideia do formato da função que estamos integrando. A Figura 3.2 apresenta o gráfico do integrando avaliado para cada uma das 10 unidades amostrais simuladas, o eixo y foi padronizada para poder colocar todas as funções no mesmo gráfico.

3.3 Técnicas de integração numérica

A necessidade de resolver uma integral numericamente aparece com bastante frequência quando ajustamos modelos de regressão com efeitos

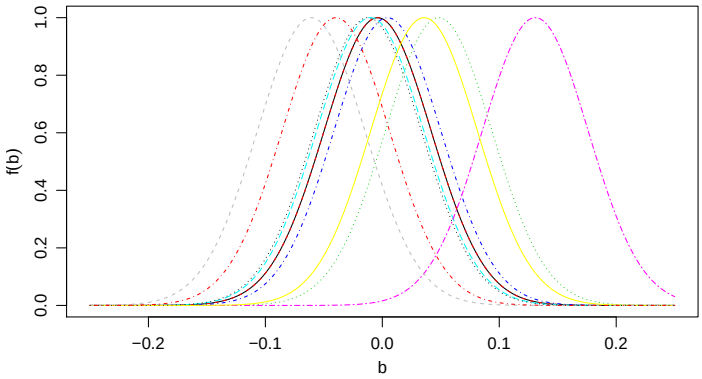

Figura 3.2: Integrando de acordo com unidade amostral - Modelo Poisson com intercepto aleatório.

aleatórios. Como exemplo ilustrativo, escolhemos o modelo de regressão Poisson com intercepto aleatório, por ser um modelo simples, contendo apenas dois parâmetros o que permite construir gráficos de contornos da verossimilhança e sua aproximação quadrática. Este modelo pode ser usado para dados que apresentem uma variância maior do que a prevista no modelo de Poisson, ou seja, dados sobredispersos. O modelo tem a seguinte forma: $Y_{ij} \sim P(\lambda_{ij})$ onde $ln(\lambda_{ij}) = \beta_0 + b_i$ e $b_i \sim N(0, 1/\tau^2)$, em que β_0 é o intercepto, b_i o efeito aleatório e τ^2 o parâmetro de precisão. Lembre-se que $i = 1, \ldots, N$ indica o número de unidades amostrais e $j = 1, \ldots, n_i$ indica o número de medidas feitas na unidade amostral i. Neste caso, a contribuição para a verossimilhança de cada unidade amostral é dada por:

$$\begin{aligned} f_i(y_{ij}|b_i, \beta_0) &= \int_{-\infty}^{\infty} \frac{\exp\{-\lambda_{ij}\}\lambda_{ij}^{y_{ij}}}{y_{ij}!} \left(\frac{\tau}{2\pi}\right)^{1/2} \exp\{-\frac{\tau^2}{2} b_i^2\} db_i \\ &= \int_{-\infty}^{\infty} \frac{\exp\{-\exp(\beta_0 + b_i)\} \exp\{(\beta_0 + b_i)^{y_{ij}}\}}{y_{ij}!} \\ &\qquad \left(\frac{\tau}{2\pi}\right)^{1/2} \exp\{-\frac{\tau^2}{2} b_i^2\} db_i. \end{aligned} \tag{3.5}$$

A integral em 3.5 tem apenas uma dimensão e precisa ser resolvida para cada uma das N unidades amostrais, e isto é repetido em cada passo de algum algoritmo de maximização numérica. Vamos isar esta integram para ilustrar diversos métodos de integração numérica e posteriormente utilizalos para estimar os parâmetros do modelo de Poisson com intercepto aleatório.

Diversos métodos de integração numérica podem ser encontrados em textos clássicos de cálculo numérico. O método do retângulo, dos trapézios, do ponto central e suas diversas variações, são métodos simples de serem implementados. Porém, na situação de modelos de regressão com efeitos aleatórios são de pouca valia, devido a restrição de que a integral a ser resolvida de deve ser própria com limites finitos e fixados. Este não é o caso na equação 3.5. No uso destes métodos não resolvamos integral na reta real, mas sim em um domínio finito adequado da função no integrando. Por exemplo, se é razoável assumir que que a quase toda a massa da distribuição está contida em $[-10, 10]$, avaliamos a integral neste intervalo.

Dentre os diversos métodos possíveis optamos por descrever o método de trapezoidal de Simpson, Quadratura Gaussiana usando os polinômios de Hermite, próprios para a integração na reta real. Métodos baseados em simulação, integração Monte Carlo e Quase Monte Carlo além da aproximação de Laplace. Combinando o método da Quadratura Gaussiana com a aproximação de Laplace, chegamos à Quadratura Adaptativa e o mesmo pode ser feito combinando Quase Monte Carlo com Laplace para obter um Quase Monte Carlo adaptativo.

3.3.1 Método Trapezoidal

O método trapezoidal consiste no uso de uma função linear para aproximar o integrando ao longo do intervalo de integração. O uso do polinômio de Newton entre os pontos $x = a$ e $x = b$ resulta em:

$$f(x) \approx f(a) + (x - a)\left[\frac{f(b) - f(a)}{b - a}\right].$$

Com a integração analítica, obtém-se:

$$\begin{aligned} I(f) &\approx \int_a^b f(a) + (x - a)\left[\frac{f(b) - f(a)}{b - a}\right] dx \\ &= f(a)(b - a) + \frac{1}{2}[f(b) - f(a)](b - a) \end{aligned}$$

Simplificando o resultado, obtém-se uma fórmula aproximada popularmente conhecida como regra ou método trapezoidal.

$$I(f) \approx \frac{[f(a) + f(b)]}{2}(b - a) \tag{3.6}$$

A expressão em 3.7 é extremamente simples de ser usada, requer apenas duas avaliações da função integrando. Sua versão R pode ser escrita como se segue.

Código 3.40: Função para integração numérica por meio do método dos trapézios.

```
> trapezio <- function(integrando, a, b, ...){
+   Int <- ((integrando(a, ...) + integrando(b, ...))/2)*(b-a)
+   return(Int)
+ }
```

Podemos agora usar o método trapezoidal para avaliar a integral do modelo Poisson com intercepto aleatório no intervalo $[-0{,}5; 0{,}5]$.

```
> log(trapezio(integrando = integrando, a = -0.5, b = 0.5, f.fixo = y~1,
+     dados= subset(dados, ID == 1), beta.fixo = 2, prec.pars=4, log=FALSE))
[1] -399.5667
```

Este método é extremamente simples e serve apenas para apresentar as ideias gerais de integração numérica. Na sequência veremos que o resultado apresentado por este método é muito ruim.

3.3.2 Método de Simpson $1/3$

Neste método, um polinômio de segunda ordem é usado para aproximar o integrando. Os coeficientes de um polinômio quadrático podem ser determinadas a partir de três pontos. Para uma integral ao longo do domínio $[a,b]$, são usados os dois pontos finais $x_1 = a$, $x_3 = b$, e o ponto central, $x_2 = (a+b)/2$. O polinômio pode ser escrito na forma:

$$p(x) = \alpha + \beta(x - x_1) + \lambda(x - x_1)(x - x_2) \tag{3.7}$$

onde α, β e λ são constantes desconhecidas avaliadas a partir da condição que diz que o polinômio deve passar por todos os pontos, $p(x_1) = f(x_1)$, $p(x_2) = f(x_2)$ e $p(x_3) = f(x_3)$. Isso resulta em:

$$\alpha = f(x_1), \quad \beta = [f(x_2) - f(x_1)]/(x_2 - x_1) \quad \text{e} \quad \lambda = \frac{f(x_3) - 2f(x_2) + f(x_1)}{2(h)^2}$$

onde $h = (b-a)/2$. Substituindo as constantes de volta em 3.8 e integrando $p(x)$ ao longo do intervalo $[a,b]$, obtém-se

$$I = \int_{x_1}^{x_3} f(x)dx \approx \int_{x_1}^{x_3} p(x)dx = \frac{h}{3}\left[f(a) + 4f(\frac{a+b}{2}) + f(b)\right].$$

Note que para o cálculo da integral é necessário apenas três avaliações da função, o que torna o método muito rápido. Podemos também facilmente implementar este método para integrar uma função qualquer, tal função terá como seus argumentos os limites $[a,b]$ e a função a ser integrada.

Código 3.41: Função para integração numérica por meio do método de Simpson.

```
> simpson <- function(integrando, a, b, ...){
+   h <- (b-a)/2
+   x2 <-(a+b)/2
+   integral <- (h/3)*(integrando(a,...) +
+                4*integrando(x2, ...) + integrando(b, ...))
+   return(integral)
+ }
```

Uma vez implementada a função podemos usá-la para integrar a nossa função de interesse. Lembre-se ainda que para o procedimento de maximização nos interessa o log do valor da integral e não a integral em log, por isso precisamos avaliar a função em sua escala original o que é computacionalmente incoveniente, mas necessário. Além disso, precisamos definir os limites de integração, neste caso fixamos -0.5 a 0.5 tendo em mente o gráfico do integrando. Apenas para comparação dos resultados usamos a função `integrate()` do R.

```
> ## Escala original
> simpson(integrando = integrando, a = -0.5, b = 0.5, f.fixo = y~1,
+     dados=subset(dados,ID==1), beta.fixo=2, prec.pars=4, log=FALSE)
[1] 2.67435e-09
> ## Em log
> log(simpson(integrando = integrando, a = -0.5, b = 0.5,
+       f.fixo = y~1, dados=subset(dados, ID == 1),
+       beta.fixo = 2, prec.pars=4, log=FALSE))
[1] -19.73956
> # Resultado com a integrate
> log(integrate(integrando, lower=-Inf, upper=Inf, f.fixo = y~1,
+     dados=subset(dados, ID == 1), beta.fixo = 2,
+       prec.pars=4, log=FALSE)$value)
[1] -22.42844
```

O resultado do método de Simpson é compatível com o obtido via `integrate()`, e bastante diferente do obtido pelo método do Trapézio. O mal desempenho do último é basicamente por este estar quase que totalmente voltado aos limites do intervalo de integração, que neste caso são definidos arbitrariamente. Se olharmos para a Figura 3.2 só a massa de probabilidade concentra-se em $[-0.1,0.1]$. Se integrarmos a função neste intervalo pelo método do Trapézio chegamos a um valor de -36.19794 mais próximo aos obtidos via `Simpson` e `integrate()`. O problema que enfrentamos aqui é como definir tais limites em situações práticas de forma geral. Esta é uma das grandes limitações destes métodos mesmo em uma única dimensão. Outra grande limitação é como expandir estes métodos para integrais dimensões maiores e como definir os limites em tais dimensões. O

número de avaliações da função cresce exponencialmente com o número de dimensões da integral. Estes problemas, não são de fácil solução e motivaram diversos outros métodos que tentam contornar o problema mantendo um número razoável da avaliações a função.

3.3.3 Quadratura de Gauss-Hermite

Nos dois métodos de integração apresentados até agora, a integral de $f(x)$ ao longo do intervalo $[a,b]$ foi avaliada representado $f(x)$ como um polinômio de fácil integração. A integral é avaliada como uma soma ponderada dos valores de $f(x)$ nos diferentes pontos. A localização dos pontos comuns é predeterminada em um dos métodos de integração. Até agora os dois métodos consideram pontos igualmente espaçados. Na quadratura de Gauss, a integral também é avaliada usando uma soma ponderadas dos valores de $f(x)$ em pontos distintos ao longo do intervalo $[a,b]$ (chamados pontos de Gauss). Estes pontos, contudo, não são igualmente espaçados e não incluem os pontos finais. O método de Gauss-Hermite é uma extensão do método de Quadratura Gaussiana para resolver integrais da forma:

$$\int_{-\infty}^{\infty} e^{-x^2} f(x) dx$$

Neste caso a integral é aproximada por uma soma ponderada, da função avaliada nos pontos de Gauss e pesos de integração.

$$\int_{-\infty}^{\infty} e^{-x^2} f(x) dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

onde n é o número de pontos usadas para a aproximação. Os x_i são as raízes do polinômio de Hermite $H_n(x)(i = 1 < 2, \ldots, n)$ e os pesos w_i associados são dados por

$$w_i = \frac{2^{n-1} n! \sqrt{\pi}}{n^2 [H_{n-1}(x_i)]^2}$$

Para a aproximação de integrais via o método de Gauss-Hermite precisamos dos pesos de integração w_i e dos pontos de Gauss x_i. A função `gauss.quad()` do pacote **statmod** calcula os pesos e os pontos de Gauss-Hermite. A função abaixo, implementa o método de integração de Gauss-Hermite para uma função qualquer unidimensional.

Código 3.42: Função para integração numérica por meio do método de Gauss-Hermite unidimensional.

```
> gauss.hermite <- function(integrando, n.pontos, ...){
+   pontos <- gauss.quad(n.pontos, kind="hermite")
+   integral <- sum(pontos$weights*integrando(pontos$nodes,...)
+                   /exp(-pontos$nodes^2))
+   return(integral)
+ }
```

Esta função tem apenas dois argumentos, o primeiro é a função a ser integrada e o segundo o número de pontos a ser utilizado na aproximação. A segunda linha da função faz apenas uma soma ponderada da função avaliada nos pontos de Gauss. O método de Gauss-Hermite apresenta duas grandes limitações. A primeira está relacionada a escolha dos pontos de Gauss, que são escolhidos baseados em $e\{-x^2\}$, independente da função $f(x)$ no integrando. Dependendo do suporte de $f(x)$, os pontos selecionados podem ou não estar dentro da área de interesse. Uma idéia natural é reescalonar os pontos de modo a colocá-los na área de maior densidade da função $f(x)$ o que gera o método chamada de Quadratura Adaptativa de Gauss-Hermite, que veremos adiante. A Figura 3.3 ilustra o problema da definição dos pontos de integração.

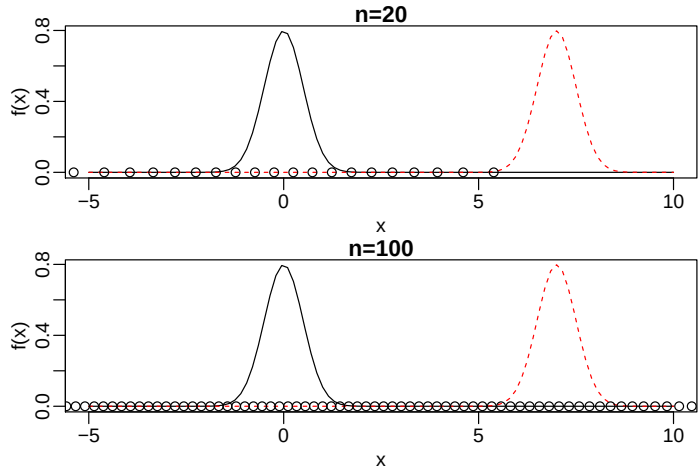

Figura 3.3: Espalhamento dos pontos de integração pelo método de Gauss-Hermite.

Pela Figura 3.3 fica claro que para integrar a função em preto os pontos $(n = 20)$ são satisfatórios, porém para a função em vermelho são claramente inadequados, já que, a área da função de maior massa não tem

nenhum ponto de integração. Desta forma, para conseguir um resultado satisfatório é necessário aumentar muito o número de pontos de integração, encarecendo o procedimento. Vamos usar esta função para avaliar o valor da integral, contida no modelo Poisson com intercepto aleatório.

```
> ## Em log
> log(gauss.hermite(integrando = integrando, n.pontos=21,
+                   f.fixo = y~1, dados=subset(dados, ID == 1),
+                   beta.fixo = 2, prec.pars=4, log=FALSE))
[1] -20.0701
```

O segundo problema do método de Gauss-Hermite está relacionado com a dimensão da integral a ser resolvida. Quando a função é unidimensional, basta espalhar os pontos sobre a reta real e avaliar a função neste pontos. Para funções em duas ou mais dimensões precisamos do produto cartesiano dos pontos de integração para espalhar na função multidimensional, ou seja, o número de pontos cresce exponencialmente de acordo com a dimensão da função a ser integrada. Por exemplo, se em uma dimensão usamos 20 pontos para a integração em duas dimensões precisamos de $20^2 = 400$, em três $20^3 = 8000$. Isso mostra que para integrar funções multidimensionais o método de Gauss-Hermite torna rapidamente proibitivo. O método de Quadratura Adaptativa de Gauss-Hermite ameniza um pouco este problema, por requerer menos pontos de integração. Porém, o problema persiste para dimensões maiores que cinco ou seis, em geral. A função abaixo implementa o método de Gauss-Hermite para dimensões maiores que um.

Código 3.43: Função para integração numérica por meio do método de Gauss-Hermite multidimensional.

```
> gauss.hermite.multi <- function(integrando,n.dim,n.pontos, ...){
+ normaliza <- function(x){exp(-t(as.numeric(x))%*%as.numeric(x))}
+ pontos <- gauss.quad(n.pontos,kind="hermite")
+ nodes <- matrix(rep(pontos$nodes,n.dim),ncol=n.dim)
+ pesos <- matrix(rep(pontos$weights,n.dim),ncol=n.dim)
+ lista.nodes <- lista.pesos <- list()
+ for(i in 1:ncol(nodes)){
+  lista.nodes[[i]] <- nodes[,i]
+  lista.pesos[[i]] <- pesos[,i]}
+ nodes = as.matrix(do.call(expand.grid,lista.nodes))
+ pesos = do.call(expand.grid,lista.pesos)
+ pesos.grid = apply(pesos,1,prod)
+ norma = apply(nodes,1,normaliza)
+ integral <- sum(pesos.grid*(integrando(nodes,...)/norma))
+ return(integral)
+ }
```

Vamos usar a função `gauss.hermite.multi()` em uma dimensão apenas

para exemplificar sua chamada.

```
> log(gauss.hermite.multi(integrando = integrando, n.pontos=21, n.dim=1,
+                   f.fixo = y~1, dados=subset(dados, ID == 1), beta.fixo =
+                   prec.pars=4, log=FALSE))
[1] -20.0701
```

3.3.4 Adaptativa Gauss-Hermite e Aproximação de Laplace

Com adaptativa Gauss-Hermite, os pontos de integração serão centrados e escalonados como se $f(x)e^{-x^2}$ fosse a distribuição gaussiana. A média desta distribuição coincide com a moda $\hat{x}$ de $ln[f(x)e^{-x^2}]$, e a variância será igual a

$$\left[-\frac{\partial^2}{\partial x^2} ln[f(x)e^{-x^2}]|_{z=\hat{z}}\right]^{-1}.$$

Assim, o novos pontos de integração adaptados serão dados por

$$x_i^+ = \hat{x} + \left[-\frac{\partial^2}{\partial x^2} ln[f(x)e^{-x^2}]|_{z=\hat{z}}\right]^{-1/2} x_i$$

com correspondentes pesos,

$$w_i^+ = \left[-\frac{\partial^2}{\partial x^2} ln[f(x)e^{-x^2}]|_{z=\hat{z}}\right]^{-1/2} \frac{e^{x_i^+}}{e^{-x_i}} w_i.$$

Como antes, a integral é agora aproximada por

$$\int f(x)e^{-x^2}dx \approx \sum_{i=1}^{n} w_i^+ f(x_i^+)$$

Quando integração de Gauss-Hermite ou adaptativa Gauss-Hermite é usada no ajuste de modelos de regressão com efeitos aleatórios, uma aproximação é aplicada para a contribuição na verossimilhança para cada uma das N unidades amostrais no conjunto de dados. Em geral, quanto maior a ordem de n pontos de integração melhor será a aproximação. Tipicamente, adaptativa Gauss-Hermite precisa de muito menos pontos que Gauss-Hermite. Por outro lado, adaptativa Gauss-Hermite requer o cálculo de $\hat{x}$ para cada unidade amostral no conjunto de dados, assim a maximização numérica do integrando encarece bastante o custo computacional desta abordagem. Além disso, como o integrando é função dos parâmetros desconhecidos β,Σ e ϕ, os pontos de quadratura, bem como os pesos usados na adaptativa Gauss-Hermite dependem destes parâmetros, e assim precisam ser atualizados a cada passo de um processo de estimação iterativo, através de algum maximizador numérico, como os encontrados na função `optim()`.

Um caso especial ocorre quando adaptativa Gauss-Hermite é aplicado com um ponto de integração. Denote $f(x)e^{-x^2}$ por $Q(x)$. Como $n = 1$, $x_1 = 0$ e $w_1 = 1$, nos obtemos $x_1^+ = \hat{x}$, que é o máximo de $Q(x)$. Além disso, os pesos de integração são iguais a

$$w_1^+ = |Q''(\hat{x})|^{-1/2}\frac{e^{-\hat{x}}}{e^{-0}} = (2\pi)^{n/2}|Q''(\hat{x})|^{-1/2}\frac{e^{Q(\hat{x})}}{f(\hat{x})}.$$

Assim, a aproximação fica dada por

$$\begin{aligned}\int f(x)e^{-x^2}dx &= \int e^{Q(x)}dx \\ &\approx w_1^+ f(x_1^+) = (2\pi)^{n/2}|Q''(\hat{x})|^{-1/2}e^{Q(\hat{x})},\end{aligned}$$

mostrando que a adaptativa Gauss-Hermite com um ponto de integração é equivalente a aproximar o integrando usando a Aproximação de Laplace. A função `laplace()` abaixo implementa a aproximação de Laplace para uma função qualquer.

Código 3.44: Função para integração numérica por meio do método de Laplace.

```
> laplace <- function(funcao, otimizador,n.dim, ...){
+   integral <- -999999
+   inicial <- rep(0,n.dim)
+   temp <- try(optim(inicial, funcao,..., method=otimizador,
+               hessian=TRUE, control=list(fnscale=-1)))
+   if(class(temp) != "try-error"){
+   integral <- exp(temp$value) * (exp(0.5*log(2*pi) -
+                   0.5*determinant(-temp$hessian)$modulus))}
+   return(integral)
+ }
```

Note a necessidade do uso da `optim()` para encontrar a moda da função e obter o Hessiano numérico. Importante, notar que na chamada para a integração via aproximação de Laplace a função integrando deve estar em escala logarítmica. A chamada abaixo, exemplifica esta aproximação.

```
> log(laplace(integrando, otimizador="BFGS", n.dim=1,
+          f.fixo = y~1, dados=subset(dados, ID == 1),
+          beta.fixo = 2, prec.pars=4, log=TRUE))
[1] -22.42681
attr(,"logarithm")
[1] TRUE
```

Para finalizar com o uso de integração por Quadratura, a função `adaptative.gauss.hermite()` implementa a integração adaptativa de Gauss-Hermite para uma função qualquer.

Código 3.45: Função para integração numérica por meio do método de Gauss-Hermite multidimensional adaptativo.

```
> adaptative.gauss.hermite <- function(funcao, n.dim, n.pontos,
+                                      otimizador, ... ){
+ normaliza <- function(x){exp(-t(as.numeric(x))%*%as.numeric(x))}
+ pontos <- gauss.quad(n.pontos,kind="hermite")
+ integral <- -999999
+ inicial <- rep(0,n.dim)
+ temp <- try(optim(inicial, funcao,..., method=otimizador,
+                   hessian=TRUE, control=list(fnscale=-1)))
+ z.chapeu <- temp$par
+ sd.chapeu <- sqrt(diag(solve(-temp$hessian)))
+ mat.nodes <- matrix(NA, ncol=n.dim,nrow=n.pontos)
+ mat.pesos <- matrix(NA,ncol=n.dim,nrow=n.pontos)
+ for(i in 1:length(z.chapeu)){
+   mat.nodes[,i] <- z.chapeu[i] + sd.chapeu[i]*pontos$nodes
+   mat.pesos[,i] <- sd.chapeu[i] *
+     (exp(-mat.nodes[,i]^2)/exp(-pontos$nodes^2))*pontos$weights
+ }
+ lista.nodes <- list()
+ lista.pesos <- list()
+ for(i in 1:ncol(mat.nodes)){
+  lista.nodes[[i]] <- mat.nodes[,i]
+  lista.pesos[[i]] <- mat.pesos[,i]}
+ nodes = as.matrix(do.call(expand.grid,lista.nodes))
+ pesos = do.call(expand.grid,lista.pesos)
+ pesos.grid = apply(pesos,1,prod)
+ norma = apply(nodes,1,normaliza)
+ integral <- sum(pesos.grid*(exp(funcao(nodes,...))/norma))
+ return(integral)
+ }
```

Para comparar os resultados utilizamos a função usando diferentes quantidade de pontos de integração.

```
> ## 1 ponto
> log(adaptative.gauss.hermite(integrando, otimizador="BFGS", n.dim=1,
+          n.pontos=1, f.fixo = y~1, dados=subset(dados, ID == 1),
+          beta.fixo = 2, prec.pars=4, log=TRUE))
[1] -22.77338
> ## 10 pontos
> log(adaptative.gauss.hermite(integrando, otimizador="BFGS", n.dim=1,
+          n.pontos=10, f.fixo = y~1, dados=subset(dados, ID == 1),
+          beta.fixo = 2, prec.pars=4, log=TRUE))
[1] -22.42682
> ## 21 pontos
> log(adaptative.gauss.hermite(integrando, otimizador="BFGS", n.dim=1,
+          n.pontos=21, f.fixo = y~1, dados=subset(dados, ID == 1),
+          beta.fixo = 2, prec.pars=4, log=TRUE))
```

```
[1] -22.42681
```

Com isso, terminamos nossa explanação dos métodos baseados na ideia de aproximar o integrando por algum tipo de polinômio que seja de fácil integração, e usar este como uma aproximação para a verdadeira integral. Na sequência, vamos apresentar um método diferente baseado em simulação, a ideia implícita é estimar o valor da integral. Este procedimento recebe o nome de integração Monte Carlo, além do método básico vamos apresentar algumas variações como o método de Quase Monte Carlo e Quase Monte Carlo adaptativo.

3.3.5 Integração Monte Carlo

Integração Monte Carlo é um método simples e geral para aproximar integrais. Assuma que desejamos estimar o valor da integral de uma função $f(x)$ em algum domínio D qualquer, ou seja,

$$I = \int_D f(x)dx \tag{3.8}$$

A função não precisa ser unidimensional. De fato, técnicas Monte Carlo são muito usadas para resolver integrais de alta dimensão, além de integrais que não tem solução analítica, como no nosso caso.

Seja uma função densidade de probabilidade $p(x)$ cujo domínio coincide com D. Então, a integral em 3.9 é equivalente a

$$I = \int_D \frac{f(x)}{p(x)} p(x)dx.$$

Essa integral cprresponde a $E\left(\frac{f(x)}{p(x)}\right)$, ou seja, o valor esperado de $\frac{f(x)}{p(x)}$ com respeito a variável aleatória distribuída como $p(x)$. Esta igualdade é verdadeira para qualquer função densidade de probabilidade em D, desde que $p(x) \neq 0$ sempre que $f(x) \neq 0$.

Pode-se estimar o valor de $E\left(\frac{f(x)}{p(x)}\right)$ gerando número aleatórios de acordo com $p(x)$, calcular $f(x)/p(x)$ para cada amostra, e calcular a média destes valores. Quanto mais amostras forem geradas, esta média converge para o verdadeiro valor da integral sendo este o princípio básico da integração Monte Carlo.

No caso específico de modelos de regressão com efeitos aleatórios, a grande maioria das integrais devem ser resolvidas nos reais, ou seja, precisamos de uma distribuição $p(x)$ com este suporte. Escolhas natural são as distribuições uniforme e gaussiana de dimensão igual a da integral a

ser resolvida. Além disso, precisamos decidir a parametrização desta distribuição, ou seja, qual será seu vetor de média e sua matriz de variância/covariância. Em algoritmos básicos, o mais comum é usar o vetor de média como 0 e variância unitária. Mas, podemos adaptar este método de forma muito similar ao Gauss-Hermite adaptativo, espalhando os pontos pelo integrando de forma a cobrir melhor a região relevante de integração.

Além do espalhamento dos pontos, a geração dos pontos aleatórios também é um fator importante para este método. Como números aleatórios serão gerados cada rodada do algoritmo, obtém-se diferentes valores para a integral o que é indesejável para maximização numéricos. Uma abordagem alternativa são métodos *Quase Monte Carlo*, nos quais os números são gerados de acordo com uma sequência de baixa discrepância. Duas opções para a geração destas sequência de baixa discrepância, estão disponíveis no pacote **fOptions**, são elas `Halton` e `Sobol`. Afora esta escolha de pontos de baixa discrepâncis em substituição a aleatórios, o procedimento é o mesmo da integral de Monte Carlo.

Para exemplificar a ideia de integração Monte Carlo e Quase Monte Carlo, a função `monte.carlo()` implementa o método para uma função qualquer, e permite ao usuário escolher a forma de espalhamento dos pontos.

Código 3.46: Função para integração numérica por meio do método de Monte Carlo e Quasi Monte Carlo.

```
> monte.carlo <- function(funcao, n.dim, n.pontos, tipo, ...){
+ if(tipo == "MC"){ pontos <- rmvnorm(n.pontos,mean=rep(0,n.dim))}
+ if(tipo == "Halton"){ pontos <- rnorm.halton(n.pontos,n.dim)}
+ if(tipo == "Sobol"){ pontos <- rnorm.sobol(n.pontos,n.dim)}
+ norma <- apply(pontos,1,dmvnorm)
+ integral <- mean(funcao(pontos,...)/norma)
+ return(integral)
+ }
```

Vamos resolver a integral contida no modelo Poisson com intercepto aleatório usando usar a função `monte.carlo()` com diferentes opções.

```
> log(monte.carlo(integrando, n.dim=1, tipo = "MC", n.pontos=20,
+          f.fixo = y~1, dados=subset(dados, ID == 1),
+          beta.fixo = 2, prec.pars=4, log=FALSE))
[1] -21.47252
> log(monte.carlo(integrando, n.dim=1, tipo = "Halton", n.pontos=20,
+          f.fixo = y~1, dados=subset(dados, ID == 1),
+          beta.fixo = 2, prec.pars=4, log=FALSE))
[1] -21.41082
> log(monte.carlo(integrando, n.dim=1, tipo = "Sobol", n.pontos=20,
+          f.fixo = y~1, dados=subset(dados, ID == 1),
+          beta.fixo = 2, prec.pars=4, log=FALSE))
```

```
[1] -21.41079
```

O mesmo problema na forma de espalhamento dos pontos encontrados no método de Quadratura de Gauss-Hermite, ocorre nos métodos de Monte Carlo e Quase Monte Carlo. Os pontos são sorteados de uma gaussiana de média 0 e variância 1, mas quando o integrando não for adequadamente coberto por estes pontos a integração será ruim. Podemos novamente adaptar os pontos de interação que agora são as amostras sorteadas, espalhando os pontos em volta de sua moda de acordo com o hessiano obtido no ponto modal, de modo a explorar melhor o integrando. O processo de adequação dos pontos é idêntico ao da adaptativa Gauss-Hermite, e não será detalhado novamente aqui. A função `adaptative.monte.carlo()` implementa este método para uma função qualquer.

Código 3.47: Função para integração numérica por meio do método de Monte Carlo Adaptativo e Quasi Monte Carlo Adaptativo.

```
> adaptative.monte.carlo <- function(funcao, n.pontos, n.dim,
+                                    tipo, otimizador, ... ){
+  pontos <- switch(tipo,
+                   "MC" = {rmvnorm(n.pontos,mean=rep(0,n.dim))},
+                         "Halton" = {rnorm.halton(n.pontos, n.dim)},
+                         "Sobol" = {rnorm.sobol(n.pontos, n.dim)})
+  integral <- -999999
+  inicial <- rep(0,n.dim)
+    temp <- try(optim(inicial, funcao, ... , method=otimizador,
+                hessian=TRUE,control=list(fnscale=-1)))
+    if(class(temp) != "try-error"){
+     z.chapeu <- temp$par
+     H <- solve(-temp$hessian)
+     sd.chapeu <- sqrt(diag(H))
+     mat.nodes <- matrix(NA, ncol=n.dim,nrow=n.pontos)
+     for(i in 1:length(z.chapeu)){
+       mat.nodes[,i] <- z.chapeu[i] + sd.chapeu[i]*pontos[,i]
+     }
+  norma <- dmvnorm(mat.nodes,mean=z.chapeu,sigma=H,log=TRUE)
+  integral = mean(exp(funcao(mat.nodes,...) - norma))
+    }
+    return(integral)
+ }
```

Novamente, vamos usar a função `adaptative.monte.carlo()` para resolver a integral contida no modelo Poisson com intercepto aleatório.

```
> log(adaptative.monte.carlo(integrando, n.dim=1, tipo="MC",
+          n.pontos=20, otimizador="BFGS",
+          f.fixo = y~1, dados=subset(dados, ID == 1),
+          beta.fixo = 2, prec.pars=4, log=TRUE))
[1] -22.42683
```

```
> log(adaptative.monte.carlo(integrando, n.dim=1, tipo="Halton",
+          n.pontos=20, otimizador="BFGS",
+          f.fixo = y~1, dados=subset(dados, ID == 1),
+          beta.fixo=2, prec.pars=4, log=TRUE))
```

```
[1] -22.42678
```

```
> log(adaptative.monte.carlo(integrando, n.dim=1, tipo="Sobol",
+          n.pontos=20, otimizador="BFGS",
+          f.fixo = y~1, dados=subset(dados, ID == 1),
+          beta.fixo=2, prec.pars=4, log=TRUE))
```

```
[1] -22.4268
```

Nesta Seção, revisamos diversos métodos de integração numérica e seu uso no R. Na sequência veremos alguns exemplos de modelos de regressão com efeitos aleatórios e como usar estes diversos métodos para a estimação por máxima verossimilhança.

3.4 Modelo Poisson com Intercepto aleatório

Seção 3.2.1 simulamos dados de um modelo Poisson com intercepto aleatório. Agora vamos ver como usar os diversos métodos de integração numérica dentro do processo de estimação dos parâmetros $\underline{\theta} = (\beta_0, \tau)$ deste modelo. O primeiro passo é escrever uma função com modelo completo.

Código 3.48: Integrando da função de verossimilhança para o modelo de regressão de Poisson com efeito aleatório de intercepto.

```
> Poisson.Int <- function(b,beta.fixo, prec.pars, X, Z, Y,log=TRUE){
+   tau <- exp(prec.pars)
+   ll = sapply(b,function(bi){
+     preditor <- as.matrix(X)%*%beta.fixo + as.matrix(Z)%*%bi
+     lambda <- exp(preditor)
+     sum(dpois(Y,lambda=lambda,log=TRUE)) +
+         dnorm(bi, 0, sd = 1/tau , log=TRUE)
+   })
+   if(log == FALSE){ll <- exp(ll)}
+   return(ll)}
```

No código 3.49 definimos uma função genérica que capta um conjunto de dados e monta a log-verossimilhança, já integrada de acordo com uma opções de integração apresentadas anteriormente. Essa função vai ser usada para diversos modelos com efeitos aleatórios apresentados neste texto.

Código 3.49: Função de verossimilhança genérica que admite diversas opções de métodos de integração numérica.

```
> veroM <- function(modelo, formu.X, formu.Z, beta.fixo, prec.pars,
+                   integral, pontos, otimizador, n.dim, dados){
+   dados.id <- split(dados, dados$ID)
+   ll <- c()
+   for(i in 1:length(dados.id)){
+     X <- model.matrix(as.formula(formu.X),data=dados.id[[i]])
+     Z <- model.matrix(as.formula(formu.Z),data=dados.id[[i]])
+     if(integral == "LAPLACE"){
+     ll[i] <- laplace(modelo,otimizador=otimizador,n.dim=n.dim,
+                X=X, Z=Z, Y=dados.id[[i]]$y, beta.fixo=beta.fixo,
+                prec.pars=prec.pars,log=TRUE)}
+     if(integral == "GH"){
+     ll[i] <- gauss.hermite.multi(modelo, n.pontos= pontos, n.dim=n.dim,
+                    X=X, Z=Z, Y=dados.id[[i]]$y, beta.fixo=beta.fixo,
+                    prec.pars=prec.pars,log=FALSE)}
+     if(integral == "MC"){
+     ll[i] <- monte.carlo(modelo,n.pontos=pontos, n.dim=n.dim,
+                    tipo= "MC", X=X, Z=Z, Y=dados.id[[i]]$y,
+                    beta.fixo=beta.fixo, prec.pars=prec.pars,
+                    log=FALSE)}
+     if(integral == "QMH"){
+     ll[i] <- monte.carlo(modelo,n.pontos=pontos, n.dim=n.dim,
+                    tipo= "Halton", X=X, Z=Z, Y=dados.id[[i]]$y,
+                    beta.fixo=beta.fixo,
+                    prec.pars=prec.pars,log=FALSE)}
+     if(integral == "QMS"){
+     ll[i] <- monte.carlo(modelo,n.pontos=pontos, n.dim=n.dim,
+                   tipo= "Sobol", X=X, Z=Z, Y=dados.id[[i]]$y,
+                   beta.fixo=beta.fixo, prec.pars=prec.pars,log=F)}
+     if(integral == "AGH"){
+     ll[i] <- adaptative.gauss.hermite(modelo,n.pontos=pontos,
+                   n.dim=n.dim, otimizador=otimizador,
+                   X=X, Z=Z, Y=dados.id[[i]]$y, beta.fixo=beta.fixo,
+                   prec.pars=prec.pars,log=TRUE)}
+     if(integral == "AMC"){
+     ll[i] <- adaptative.monte.carlo(modelo,n.pontos=pontos,
+                    n.dim=n.dim, otimizador=otimizador, tipo="MC",
+                    X=X, Z=Z, Y=dados.id[[i]]$y, beta.fixo=beta.fixo,
+                    prec.pars=prec.pars,log=TRUE)}
+     if(integral == "AQMH"){
+     ll[i] <- adaptative.monte.carlo(modelo,n.pontos=pontos, n.dim=n.dim,
+                   otimizador=otimizador, tipo="Halton", X=X, Z=Z,
+                   Y=dados.id[[i]]$y, beta.fixo=beta.fixo,
+                   prec.pars=prec.pars,log=TRUE)}
+     if(integral == "AQMS"){
+     ll[i] <- adaptative.monte.carlo(modelo,n.pontos=pontos, n.dim=n.dim,
+                   otimizador=otimizador, tipo="Sobol", X=X, Z=Z,
+                   Y=dados.id[[i]]$y, beta.fixo=beta.fixo,
+                   prec.pars=prec.pars,log=TRUE)}
+     }
+   return(sum(log(ll)))
+ }
```

A função `veroM()` foi definida de forma genérica, Para o modelo Poisson com intercepto aleatório definimos:

Código 3.50: Função de verossimilhança marginal para o modelo de regressão de Poisson com efeito aleatório de intercepto.

```
> mod.Poisson <- function(b0, tau, integral, pontos, otimizador,
+     n.dim, dados) {
+     ll = veroM(modelo = Poisson.Int, formu.X = "~1", formu.Z = "~1",
+         beta.fixo = b0, prec.pars = tau, integral = integral,
+         pontos = pontos, otim = otimizador, n.dim = n.dim,
+         dados = dados)
+     return(-ll)
+ }
```

Usando a função `mle2()` para estimar os parâmetros via o algoritmo `BFGS`, e aproximação de Laplace temos o seguinte.

```
> P.laplace = mle2(mod.Poisson,start=list(b0=0,tau=log(1/4)),
+   data=list(integral="LAPLACE",otimizador = "BFGS", n.dim=1,
+   dados=dados, pontos=NA))
> summary(P.laplace)
Maximum likelihood estimation

Call:
mle2(minuslogl = mod.Poisson, start = list(b0 = 0, tau = log(1/4)),
    data = list(integral = "LAPLACE", otimizador = "BFGS", n.dim = 1,
        dados = dados, pontos = NA))

Coefficients:
    Estimate Std. Error z value     Pr(z)
b0  2.006958   0.072709 27.6027 < 2.2e-16 ***
tau 1.620670   0.290304  5.5827 2.369e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

-2 log L: 491.6252
```

Para os demais métodos de integração obtemos os seguintes valores da log-verossimilhança:

```
> par <- coef(P.laplace)
> MET <- c("LAPLACE","GH","MC","QMH","QMS","AGH","AMC","AQMH","AQMS")
> sapply(MET, function(metodo){
+        mod.Poisson(b0=par[1],tau=par[2], integral=metodo,
+                    pontos=21, n.dim=1,otimizador="BFGS", dados=dados)})
 LAPLACE       GH       MC      QMH      QMS      AGH      AMC     AQMH
245.8126 243.9302 248.7211 245.2455 244.9611 245.8104 245.8128 245.7723
    AQMS
245.8448
```

Neste examploe todos os métodos apresentaram valores muito próximos do obtido pela aproximação de Laplace, mais isto não significa que

todos se comportam de forma igual dentro dos maximizadores numéricos. Por exemplo, o método Monte Carlo, precisa de muitos pontos para conseguir convergência, o que o torna muito lento. A cada iteração estamos resorteando de uma gaussiana multivariada. Alternativamente e de forma mais eficiente, podemos sortear apenas uma vez e usar os mesmos pontos em todas as interações do algoritmo numérico. O mesmo se aplicada a todos os outros métodos. Para implementações mais eficiente devemos abandonar ou alterar a função autocontida como apresentada, que foi usada aqui apenas como exemplo didático. Algoritmos mais eficiente podem já receber os pontos de integração como argumento da função Implementações análogas podem ser feitas para implementar a Quadratura de Gauss-Hermite de forma mais eficiente. Ressaltamos que neste momento não estamos interessados em eficiência computacional, apenas em apresentar os aspectos gerais do métodos de integração numérica.

O mesmo ajuste usando quadratura de Gauss-Hermite fonece o resultado a seguir.

```
> P.GH = mle2(mod.Poisson,start=list(b0=0,tau=log(1/4)),
+   data=list(integral="GH",pontos=100, n.dim=1, dados=dados))
> summary(P.GH)
Maximum likelihood estimation

Call:
mle2(minuslogl = mod.Poisson, start = list(b0 = 0, tau = log(1/4)),
    data = list(integral = "GH", pontos = 100, n.dim = 1, dados = dados))

Coefficients:
    Estimate Std. Error z value     Pr(z)
b0  2.016415   0.073391 27.4749 < 2.2e-16 ***
tau 1.635619   0.291663  5.6079 2.048e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

-2 log L: 491.7102

> P.AQMS = mle2(mod.Poisson,start=list(b0=0,tau=log(1/4)),
+   data=list(integral="AQMS",pontos=10,otimizador="BFGS",
+   n.dim=1, dados=dados))
> summary(P.AQMS)
Maximum likelihood estimation

Call:
mle2(minuslogl = mod.Poisson, start = list(b0 = 0, tau = log(1/4)),
    data = list(integral = "AQMS", pontos = 10, otimizador = "BFGS",
        n.dim = 1, dados = dados))

Coefficients:
    Estimate Std. Error z value     Pr(z)
b0  2.006742   0.072547 27.6613 < 2.2e-16 ***
tau 1.622157   0.287350  5.6452  1.65e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

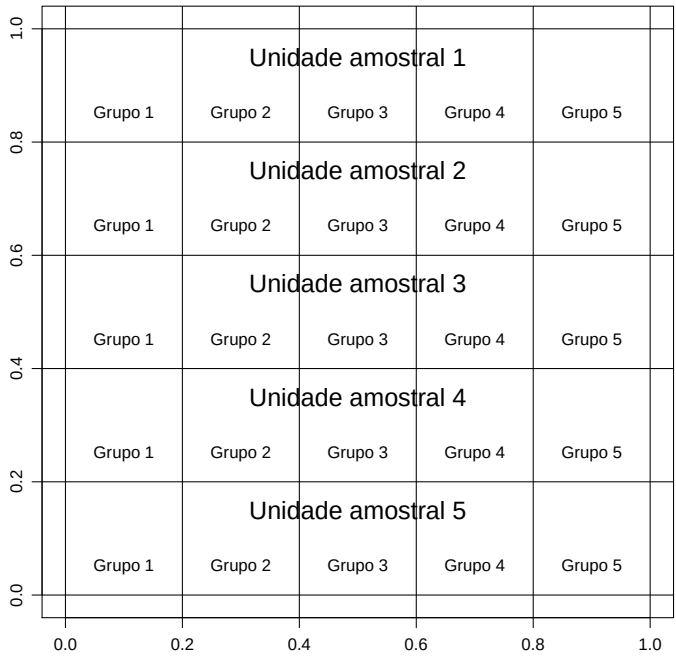

Figura 3.4: Estrutura de um delineamento com efeito aninhado.

```
-2 log L: 491.7037
```

Com isso, passamos por todas as etapas do processo de estimação de um modelo de regressão com efeitos aleatórios ilustrando princípios básicos e fundamentos dos algoritmos. Na sequência, vamos discutir alguns modelos com mais elementos, com por exemplo, com diferentes estruturas de efeitos aleatórios. O processo de especificação/implementação do modelo e os métodos de integração seguem os mesmos princípios vistos até aqui. Nos próximos exemplos mudamos a matriz Z de delineamento associada aos efeitos aleatórios.

3.5 Poisson com efeito aninhado

Considere o experimento, onde i unidades amostrais são divididas em j blocos e dentro de cada bloco são realizadas k repetições. A Figura 3.4 ilustra este delineamento amostral.

Suponha que a variável de interesse segue um modelo de Poisson. Então um modelo adequado para este tipo de experimento é:

$$
\begin{aligned}
Y_{ijk} &\sim P(\lambda_{ijk}) \\
g(\lambda_{ijk}) &= (\beta_0 + b_i) + b_{i:j} \\
b_i &\sim N(0, \sigma^2) \;\; ; \;\; b_{i:j} \sim N(0, \tau^2)
\end{aligned}
$$

onde $i = 1, \ldots, N$ é o número de unidades amostrais envolvidas no experimento que no caso da Figura 3.4 possui $N = 5$. O índice $j = 1, \ldots, n_i$ identifica os blocos dentro de cada unidade amostral e $k = 1, \ldots, n_{ij}$ é o número de repetições dentro de cada grupo em cada unidade amostral. Note que este modelo tem três parâmetros $\underline{\theta} = (\beta_0, \sigma^2, \tau^2)$.

Para este exemplo vamos simular um conjunto de dados seguindo esta estrutura. Vamos fixar o número de unidades amostrais em $N = 4$, vamos dividir cada unidade em $n_i = 5$ blocos e dentro de cada bloco realizar $n_{ij} = 4$ observações. A função `rpois.ani()` simula dados deste modelo. A Figure 3.5 ilustra a estrutura do experimento.

Código 3.51: Função para simular do modelo de regressão Poisson com efeitos aleatórios aninhados sobre o intercepto.

```
> rpois.ani <- function(N, ni, nij, beta.fixo, prec.pars){
+   ua <- as.factor(rep(1:nij,each=N*ni))
+   bloco <-rep(as.factor(rep(1:ni,each=nij)),N)
+   rep <- rep(as.factor(rep(1:nij,ni)),N)
+   dados <- data.frame(ua,bloco,rep)
+   dados$Bloco.A <- interaction(ua,bloco)
+   Z1 <- model.matrix(~ ua - 1,data=dados)
+   Z2 <- model.matrix(~Bloco.A -1, data=dados)
+   X <- model.matrix(~1, data=dados)
+   n.ua <- ncol(Z1)
+   n.bloco <- ncol(Z2)
+   b.ua <- rnorm(n.ua,0,sd=1/prec.pars[1])
+   b.bloco <- rnorm(n.bloco, sd=1/prec.pars[2])
+   Z <- cbind(Z1,Z2)
+   XZ <- cbind(X,Z)
+   beta <- c(beta.fixo,b.ua,b.bloco)
+   preditor <- XZ%*%beta
+   lambda <- exp(preditor)
+   y <- rpois(length(lambda),lambda=lambda)
+   dados$y <- y
+   names(dados) <- c("ID","bloco","rep","Bloco.A","y")
+   return(dados)
+ }
```

Para simular do modelo precisamos fixar o vetor de parâmetros, vamos usar $\beta_0 = 3$, $\sigma = 1$ e $\tau = 2$. A seguinte chamada da função `rpois.ani()`

realiza a simulação e retorna um conjunto de dados, no formato adequado para ser utilizado posteriormente no processo de inferência.

```
> set.seed(123)
> dados <- rpois.ani(N = 4, ni = 5, nij = 4, beta.fixo = 3,
+                    prec.pars=c(2,2))
> head(dados)
```

```
  ID bloco rep Bloco.A  y
1  1     1   1     1.1 13
2  1     1   2     1.1 15
3  1     1   3     1.1 11
4  1     1   4     1.1 11
5  1     2   1     1.2  9
6  1     2   2     1.2  6
```

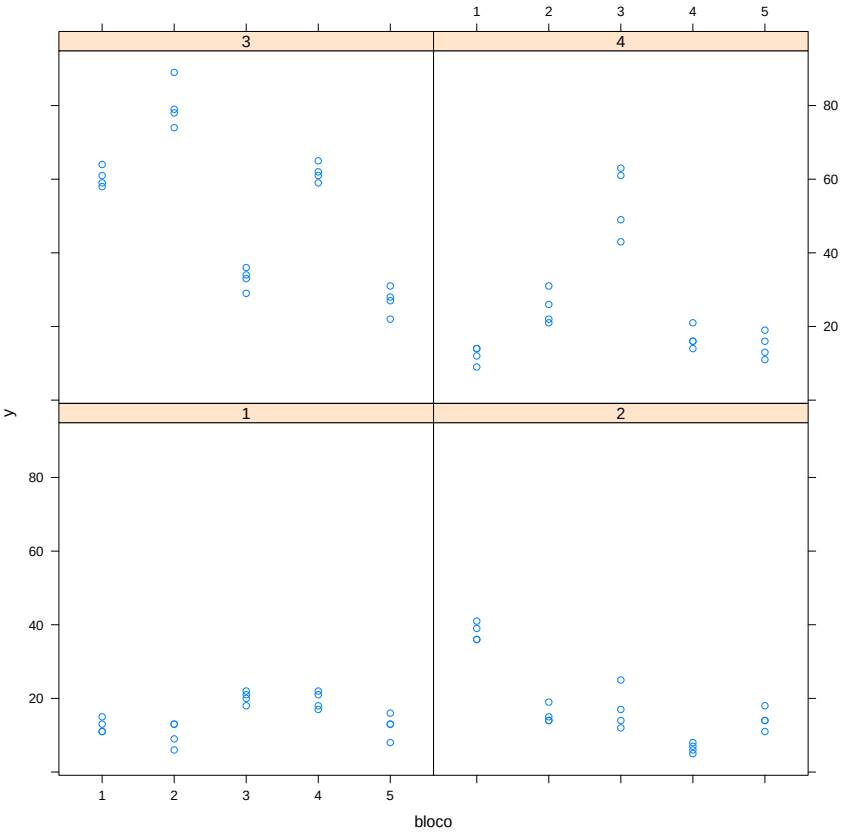

Figura 3.5: Análise descritiva modelo com efeito aninhado.

A seguir escrevemos uma função com a estrutura do modelo.

Código 3.52: Integrando da função de verossimilhança para o modelo de regressão de Poisson com efeito aleatório aninhado sobre o intercepto.

```
> Poisson.Ani <- function(b, beta.fixo, prec.pars,X, Z, Y,log=TRUE){
+   sigma <- exp(prec.pars[1])
+   tau <- exp(prec.pars[2])
+   preditor <- as.matrix(X)%*%beta.fixo + as.matrix(Z)%*%b
+   lambda <- exp(preditor)
+   ll =  sum(dpois(Y,lambda=lambda,log=TRUE)) +
+             dnorm(b[1], 0, sd = 1/sigma , log=TRUE) +
+             sum(dnorm(b[2:6], 0, sd = 1/tau , log=TRUE))
+   if(log == FALSE){ll <- exp(ll)}
+   return(ll)}
```

Note a reparametrização feita nos parâmetros de variância, onde vamos estimá-los em escala logarítmica. Esta transformação muda o espaço de busca do algoritmo numérico dos reais positivos para todo os reais, o que ajuda no processo de otimização. Além disso, observe a complexidade deste modelo, para cada unidade amostral temos 6 desvios aleatórios, um pela própria unidade amostral e mais 5 um para cada bloco dentro da unidade amostral. Isso impacta fortemente no método de integração a ser escolhido para resolver a integral contida na função de verossimilhança. Por exemplo, pelo método de Gauss-Hermite, suponha que escolhemos 21 pontos de integração em seis dimensões implica que a cada interação do algoritmo de maximização numérica precisamos avaliar a função $21^6 = 85766121$ o que é praticamente inviável. Nestas situações apenas a aproximação de Laplace ainda é aplicável, e será usada neste problema. Veja também que com apenas 4 unidades amostrais, devemos estimar $4 + 4 * 5 = 24$ efeitos aleatórios no total. A função `vero.Poisson.Ani()` apresenta uma versão simplificada da função `veroM()` do código 3.49 para o caso do Modelo Poisson com efeito aninhado usando apenas a integração por Laplace.

Código 3.53: Função de verossimilhança para o modelo de regressão de Poisson com efeito aleatório aninhado sobre o intercepto.

```
> vero.Poisson.Ani <- function(modelo, formu.X, formu.Z, beta.fixo,
+                              prec.pars,otimizador, dados){
+  dados.id <- split(dados, dados$ID)
+  ll <- c()
+  for(i in 1:length(dados.id)){
+     X <- model.matrix(as.formula(formu.X),data=dados.id[[i]])
+     Z <- model.matrix(as.formula(formu.Z),data=dados.id[[i]])
+     ll[i] <- laplace(modelo,otimizador=otimizador,n.dim=ncol(Z),
+                      X=X, Z=Z , Y=dados.id[[i]]$y,
+                      beta.fixo=beta.fixo,
+                      prec.pars=prec.pars,log=TRUE)
+   }
+   return(sum(log(ll)))
+ }
```

Escrevemos o modelo no formato adequado para ser usado dentro da função `mle2()`.

Código 3.54: Função de log-verossimilhança marginal para o modelo de regressão de Poisson com efeito aleatório aninhado sobre o intercepto.

```
> mod.Poisson.Ani <- function(b0,sigma,tau, otimizador,formu.X,
+                             formu.Z,  dados){
+ ll = vero.Poisson.Ani(modelo = Poisson.Ani, formu.X = formu.X,
+                       formu.Z = formu.Z, beta.fixo = b0,
+                       prec.pars=c(sigma,tau),
+                       otimizador=otimizador,dados=dados)
+   #print(round(c(b0,sigma,tau,ll),2))
+   return(-ll)}
```

O processo de otimização da função de log-verossimilhança marginalizada pela função `mle2()`.

```
> require(bbmle)
> dados$UM <- 1
> ini <- c(log(mean(dados$y)), log(sd(dados$y))/2)
> Poisson.Aninhado = mle2(mod.Poisson.Ani,
+                         start=list(b0= ini[1],sigma=ini[2],tau= ini[2]),
+                         method="BFGS",control=list(lmm=3, reltol=1e-5),
+                         data=list(formu.X="~1", formu.Z="~UM+bloco-1",
+                         otimizador = "BFGS",dados=dados))
> summary(Poisson.Aninhado)
Maximum likelihood estimation

Call:
```

```
mle2(minuslogl = mod.Poisson.Ani, start = list(b0 = ini[1], sigma = ini[2],
    tau = ini[2]), method = "BFGS", data = list(formu.X = "~1",
    formu.Z = "~UM+bloco-1", otimizador = "BFGS", dados = dados),
    control = list(lmm = 3, reltol = 1e-05))

Coefficients:
       Estimate Std. Error z value     Pr(z)
b0    3.0869605  0.0342782  90.056 < 2.2e-16 ***
sigma 0.8706941  0.0134722  64.629 < 2.2e-16 ***
tau   0.6972780  0.0063951 109.033 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

-2 log L: 554.7753
```

Dada a reparametrização dos parâmetros de variância, precisamos retorná-los para escala original. De acordo com a propriedade de invariância dos estimadores de máxima verossimilhança, para as estimativas pontuais basta aplicar a transformação inversa, ou seja,

```
> exp(coef(Poisson.Aninhado)[2:3])
   sigma      tau
2.388568 2.008279
```

verifica-se que os valores estimados estão bastante próximos dos verdadeiros valores dos parâmetros utilizados na simulação. Para a construção de intervalos de confiança, para o parâmetro β_0 nenhum problema basta usar os resultados assintóticos e construir o intervalo de confiança usando o erro padrão fornecido junto ao $summary()$ do modelo, ou então,

```
> confint(Poisson.Aninhado, method="quad")
          2.5 %    97.5 %
b0    3.0197764 3.1541446
sigma 0.8442890 0.8970991
tau   0.6847438 0.7098122
```

note que a saída acima apresenta os intervalos de confiança para os parâmetros de variância reparametrizado. Se desejarmos obter intervalos aproximados para os parâmetros de variância na escala original não podemos apenas aplicar a transformação inversa. Para isto, precisamos utilizar os resultados apresentados nos Teoremas 4 e 5. Aplicando estes resultados temos,

```
> Vcov <- vcov(Poisson.Aninhado)
> sd.sigma <- sqrt(exp(coef(Poisson.Aninhado)[2])^2*Vcov[2,2])
> sd.tau <- sqrt(exp(coef(Poisson.Aninhado)[3])^2*Vcov[3,3])
> ic.sigma = exp(coef(Poisson.Aninhado)[2]) + c(-1,1)*qnorm(0.975)*sd.sigma
> ic.tau = exp(coef(Poisson.Aninhado)[3]) + c(-1,1)*qnorm(0.975)*sd.tau
> ic.sigma
[1] 2.325498 2.451638
> ic.tau
[1] 1.983107 2.033451
```

os intervalos de confiança obtidos via aproximação quadrática parecem muito curtos. Isso pode ser devido a uma pobre aproximação do Hessiano numérico, utilizado para calcular os erros padrões assintóticos, ou a

aproximação quadrática é muito ruim nesta situação, apresentando erros padrões extremamente otimistas.

Para investigar a primeira possibilidade, podemos recalcular o Hessiano numérico pelo método de Richardson especifico para aproximar numericamente a derivada segunda de uma função qualquer. Este método está implementado na função `hessian()` do pacote **numDeriv**. Para usar a função `hessian()`, precisamos reescrever a função de verossimilhança, passando os parâmetros em forma de vetor.

```
> mod.Poisson.Ani.Hessian <- function(par, dados){
+   saida <- mod.Poisson.Ani(b0=par[1], sigma = par[2], tau = par[3],
+                            otimizador="BFGS", formu.X = "~1",
+                            formu.Z = "~UM + bloco - 1", dados=dados)
+   return(saida)
+ }
```

Reescrita a função queremos avaliar o Hessiano no ponto de máximo, ou seja, a matriz de informação observada.

```
> Io <- hessian(mod.Poisson.Ani.Hessian, x = coef(Poisson.Aninhado),
+               method="Richardson", dados=dados)
```

Podemos comparar o inverso da matriz de informação observada, pelo algoritmo `BFGS` e o obtido fora pelo método de Richardson.

```
> Vcov         ## Anterior
              b0        sigma          tau
b0    0.0011749970 4.501189e-04 2.200299e-04
sigma 0.0004501189 1.815005e-04 8.595639e-05
tau   0.0002200299 8.595639e-05 4.089763e-05

> solve(Io)    ## Richardson
           [,1]         [,2]         [,3]
[1,] 0.056716174  0.003183134  0.000626882
[2,] 0.003183134  0.233766754 -0.010478210
[3,] 0.000626882 -0.010478210  0.035470558
```

É possível ver claramente que pelo método de Richardson as variância são maiores, levando a maior confiança dos resultados. Podemos novamente construir os intervalos de confiança agora com os novos desvios padrões, inclusive para o parâmetro de média β_0.

```
> Vcov.H <- solve(Io)
> ic.b0 <- coef(Poisson.Aninhado)[1] + c(-1,1)*sqrt(Vcov.H[1,1])
> sd.sigma.H <- sqrt(exp(coef(Poisson.Aninhado)[2])^2*Vcov.H[2,2])
> sd.tau.H <- sqrt(exp(coef(Poisson.Aninhado)[3])^2*Vcov.H[3,3])
> ic.sigma.H = exp(coef(Poisson.Aninhado)[2]) +
+                  c(-1,1)*qnorm(0.975)*sd.sigma.H
> ic.tau.H = exp(coef(Poisson.Aninhado)[3]) +
+                c(-1,1)*qnorm(0.975)*sd.tau.H
> ic.b0
[1] 2.848809 3.325112
> ic.sigma.H
[1] 0.1250859 4.6520504
> ic.tau.H
```

```
[1] 1.266958 2.749599
```

Com os erros padrões corrigidos os intervalos de confiança ficaram mais largos, e mais coerentes com o que esperamos, nos dois casos o verdadeiro valor dos parâmetros estão contidos nos intervalos. Uma outra opção, muito mais cara computacionalmente é obter os intervalos baseados em perfil de verossimilhança.

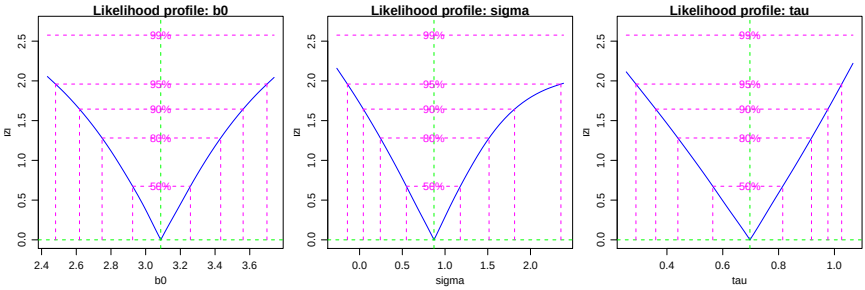

Figura 3.6: Perfil de verossimilhança - Modelo Poisson com efeito aninhado.

Podemos comparar os intervalos perfilhados com os obtidos pela aproximação quadrática.

```
> perfil.b0 <- confint(perfil)[1,]
> perfil.sigma <- exp(confint(perfil)[2,])
> perfil.tau <- exp(confint(perfil)[3,])
> ic = cbind(rbind(ic.b0,perfil.b0),
+            rbind(ic.sigma.H,perfil.sigma),
+            rbind(ic.tau.H,perfil.tau))
> ic
```

```
             2.5 %   97.5 %     2.5 %   97.5 %    2.5 %   97.5 %
ic.b0     2.848809 3.325112 0.1250859  4.65205 1.266958 2.749599
perfil.b0 2.480810 3.699131 0.8672940 10.54873 1.334975 2.790330
```

Os resultados mostram que a aproximação quadrática, tende a apresentar intervalos mais curtos que os de verossimilhança perfilhada. O parâmetro que parece sofrer mais com este efeito é o σ. Com isso, concluímos o processo de inferência do modelo Poisson com efeito aninhado.

3.6 Modelo Beta longitudinal

Considere a situação onde uma variável resposta Y_{it} restrita ao intervalo unitário, é observada em $i = 1, \ldots, N$ unidades amostrais, em $t = 1, \ldots, n_i$ tempos. A natureza da variável aleatória indica que a distribuição Beta é uma candidata natural para descrever os dados. Um possível modelo para

esta situação é o seguinte:

$$\begin{aligned} Y_{it} &\sim B(\mu_{it}, \phi) \\ g(\mu_{it}) &= (\beta_0 + b_i) + (\beta_1 + b_1)t \\ \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} &\sim NM_2 \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_I^2 & \rho \\ \rho & \sigma_S^2 \end{bmatrix} \right) \end{aligned}$$

A principal diferença deste modelo para os anteriores é que não supomos independência entre os efeitos aleatórios. Neste caso, temos um modelo com intercepto e inclinação aleatória e adicionando um parâmetro de correlação entre os efeitos. Para completar a especificação precisamos da função $g(\cdot)$ que liga o preditor a esperança da Beta que definimos pela função como a `logit`. A função `rbeta.model()` simula uma realização deste modelo.

Código 3.55: Função para simular variáveis aleatórias do modelo de regressão Beta longitudinal.

```
> inv.logit <- function(x){exp(x)/(1+exp(x))}
> rbeta.model <- function(ID, tempo, beta.fixo, prec.pars){
+   dados = data.frame("ID" = rep(1:ID,each=tempo),
+            "cov" = rep(seq(0, 1,l=tempo),ID))
+   dados.id <- split(dados,dados$ID)
+   cov.rho <- prec.pars[3]*sqrt(prec.pars[1])*sqrt(prec.pars[2])
+   Sigma<-matrix(c(prec.pars[1],cov.rho,cov.rho,prec.pars[2]),2,2)
+   y <- matrix(NA, ncol=ID, nrow=tempo)
+   for(i in 1:ID){
+     X <- model.matrix(~cov, data=dados.id[[i]])
+     Z <- model.matrix(~cov, data=dados.id[[i]])
+     b <- rmvnorm(n=1,mean=c(0,0),sigma=Sigma)
+     preditor <- X%*%as.numeric(beta.fixo) + Z%*%as.numeric(b)
+     mu <- inv.logit(preditor)
+     y[,i]<-rbeta(length(mu),mu*prec.pars[4],
+                  (1-mu)*prec.pars[4])
+   }
+   dados$y <- c(y)
+   return(dados)
+ }
```

O modelo Beta tem vetor de parâmetros $\underline{\theta} = (\beta_0, \beta_1, \sigma_I, \sigma_S, \rho, \phi)$. São dois parâmetros de média e quatro de variabilidade, sendo três deles associados à gaussiana bivariada atribuída aos efeitos aleatórios, além do parâmetro de dispersão da Beta. A Figura 3.7 apresenta um gráfico das trajetórias simuladas para cada unidade amostral ao longo do tempo. Para simulação usamos uma chamada da função `rbeta.model()` do código (3.55). «eval=F» dados <- rbeta.model(ID = 10, tempo = 15, beta.fixo = c(0.5, 0.8), prec.pars=c(1, 0.2, 0.3, 30))

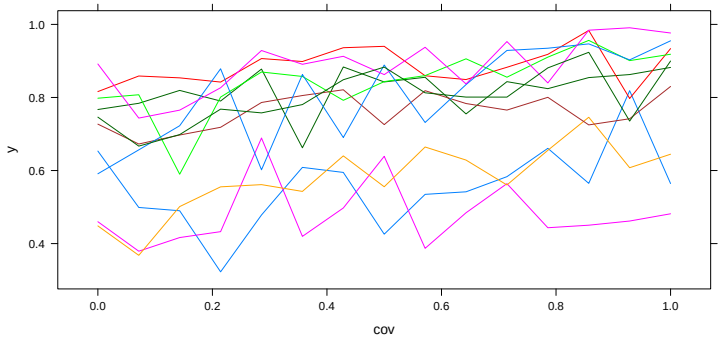

Figura 3.7: Trajetórias por unidade amostral - Modelo Beta longitudinal.

Escrever este modelo em R requer cuidado com os parâmetros de variância envolvidos. Todos os parâmetros de variância/dispersão foram reparametrizados, para serem estimados em escala logarítmica. Para o parâmetro de correlação ρ, que assume valores no intervalo $(-1,1)$, utilizamos a transformação logística.

Código 3.56: Integração da função de log-verossimilhança para o modelo de regressão Beta logitudinal.

```
> transf.rho <- function(rho){
+   -1+2*(exp(rho)/(exp(rho)+1))
+ }
> vero.slope <- function(uv,beta.fixo, prec.pars, X, Z, Y,log=TRUE){
+   sigmaI <- exp(prec.pars[1])^2
+   sigmaS <- exp(prec.pars[2])^2
+   rho <- transf.rho(prec.pars[3])
+   phi <- exp(prec.pars[4])
+   cov.rho <- rho*(sqrt(sigmaI)*sqrt(sigmaS))
+   if(class(dim(uv)) == "NULL"){uv <- matrix(uv,1,2)}
+   ll = apply(uv,1,function(uvi){
+     preditor <- X%*%beta.fixo + Z%*%as.numeric(uvi)
+     mu <- inv.logit(preditor)
+     sigma <- matrix(c(sigmaI,cov.rho,cov.rho,sigmaS),2,2)
+     sum(dbeta(Y, mu*phi, (1-mu)*phi, log=TRUE)) +
+         dmvnorm(uvi, c(0,0), sigma = sigma , log=TRUE)})
+   if(log == FALSE){ll <- exp(ll)}
+   return(ll)}
```

Usando a função `veroM()` definida no código 3.49, podemos colocar o modelo da forma apropriada para utilizar a função `mle2()`.

Código 3.57: Função de log-verossimilhança marginal para o modelo de regressão Beta longitudinal.

```
> model.Beta <- function(b0, b1, sigmaI, sigmaS, rho, phi,
+     otimizador, n.dim, dados) {
+     ll = veroM(modelo = vero.slope, formu.X = "~cov", formu.Z = "~cov",
+         beta.fixo = c(b0, b1), prec.pars = c(sigmaI, sigmaS,
+             rho, phi), integral = integral, pontos = pontos,
+         otimizador = otimizador, n.dim = n.dim, dados = dados)
+     return(-ll)
+ }
```

Com o modelo no formato adequado podemos proceder com a inferência realizando o ajuste com uma chamada ã função `mle2()`.

```
> ajuste = mle2(model.Beta, start=list(b0=0, b1=0,sigmaI=-0.5,
+              sigmaS=-0.5, rho = 0.3, phi = log(25)), method="BFGS",
+              data=list(integral="LAPLACE", pontos=1, otimizador="BFGS",
+              n.dim=2,dados=dados))
> summary(ajuste)
```

```
Maximum likelihood estimation

Call:
mle2(minuslogl = model.Beta, start = list(b0 = 0, b1 = 0, sigmaI = -0.5,
    sigmaS = -0.5, rho = 0.3, phi = log(25)), method = "BFGS",
    data = list(integral = "LAPLACE", pontos = 1, otimizador = "BFGS",
        n.dim = 2, dados = dados))

Coefficients:
        Estimate Std. Error z value     Pr(z)
b0      0.717119   0.212721  3.3712 0.0007485 ***
b1      0.991160   0.029069 34.0970 < 2.2e-16 ***
sigmaI -0.528103   0.150590 -3.5069 0.0004534 ***
sigmaS -0.742937         NA      NA        NA
rho     1.299390   0.446684  2.9090 0.0036262 **
phi     3.546104   0.081774 43.3644 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

-2 log L: -334.7962
```

Os intervalos de confiança podem ser obtidos na escala reparametrizada. Usamos os os resultados dos Teoremas 1.4 e 1.5 para obter intervalos aproximados na escala original dos parâmetros.

```
> confint(ajuste, method="quad")
```

```
            2.5 %     97.5 %
b0      0.3001937  1.1340446
b1      0.9341860  1.0481339
sigmaI -0.8232545 -0.2329508
sigmaS        NaN        NaN
rho     0.4239062  2.1748739
phi     3.3858288  3.7063789
```

Outra forma alternativa é obter intervalos baseados em perfil de verossimilhança, que em geral apresentam resultados melhores, porém são computacionalmente mais "caros"(demorados). Por outro lado, a transformação para escala original é simples aplicando-se a diretamente a função de reparametrização os limites do intervalo. Isto é justificado pela propriedade de invariância da verossimilhança, A Figura 3.8 apresenta os perfis de verossimilhança dos parâmetros do modelo Beta longitudinal.

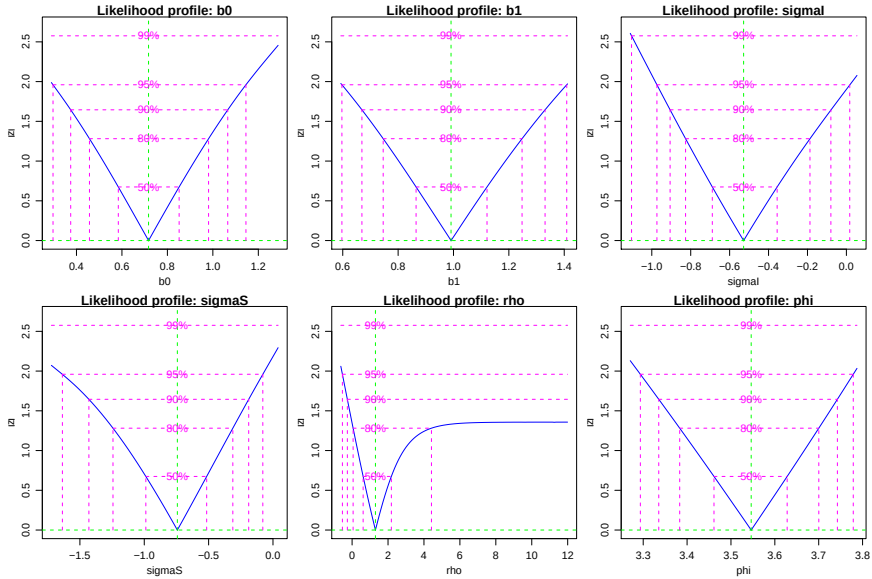

Figura 3.8: Perfil de verossimilhança - Modelo Beta longitudinal.

3.7 Modelo de Teoria de Resposta ao Item

A teoria de resposta ao item (TRI) tem ganhado muito destaque com a expansão e visibilidade de suas aplicações, principalmente em avaliações na área de educação. Os modelos básicos de TRI pode ser vistos como modelos de efeitos aleatórios, com a particularidade que, na sua forma básica, não há parâmetros de variância para estimar explicitamente. Para ilustrar isso, será implementado o modelo logístico de três parâmetros, destacando alguns casos particulares deste.

Considere o caso onde um teste é formado por i questões e j indivíduos são avaliados. O modelo logístico de três parâmetros postula que a probabilidade de um indivíduo qualquer responder corretamente a cada uma das questões envolve: (i) a habilidade latente do individuo θ_j, (ii) a dificuldade

β_i, (iii) a discriminância α_i e (iv) a probabilidade de acerto casual c_i, para cada uma das questões. A equação do logístico de três parâmetros é:

$$P(Y_{ij}|\theta_j) = c_i(1-c_i)\frac{1}{1+\exp\{-\alpha_i(\theta_j-\beta_i)\}}.$$

Pode-se identificar facilmente dois casos particulares. O primeiro quando a probabilidade de acerto casual é desprezível, ou seja, $c_i = 0$. O segundo quando a discriminância é igual para todas as questões, ou seja, $\alpha_i = \alpha$. No caso de $\alpha = 1$ tem-se o conhecido modelo de Rasch.

O modelo completo descrito de forma hierárquica fica da seguinte forma:

$$\begin{aligned} Y_{ij}|\theta_j &\sim B(n=1, p_{ij}) \\ \theta_j &\sim N(0,1). \end{aligned}$$

Para fazer inferência sobre os parâmetros deste modelo, é necessário a obtenção da verossimilhança marginal, obtida após a integração dos efeitos aleatórios, neste caso, as habilidades latentes θ_j. O integrando desta verossimilhança marginal é o produto de uma binomial por uma gaussiana padrão, e não tem solução analítica. Desta forma, pode-se usar métodos para integração numérica como os já apresentados.

A implementação deste modelo segue os mesmos princípios dos outros modelos de efeitos aleatórios já apresentados porém com duas diferenças básicas. A primeira é que o preditor neste caso é não linear dado pelo modelo logístico. A segunda é que não temos o parâmetro de variância do efeito aleatório, que é neste caso suposto igual a 1. Como primeiro passo, vamos implementar a função do modelo logístico com três parâmetros.

Código 3.58: Definição do modelo logístico.

```
> logistico <- function(beta, alpha, ce, theta){
+         return(ce + (1-ce)* (1/(1+ exp(-alpha*(theta-beta)))))
+ }
```

Vamos criar uma função que permite obter simulações deste modelo.

Código 3.59: Função para simular do modelo logístico de TRI.

```
> simula.tri <- function(n.ind, beta, alpha, ce){
+   theta <- rnorm(n.ind, 0, 1)
+   p <- matrix(NA, ncol = length(beta), nrow = n.ind)
+   y <- p
+   for(i in 1:length(beta)){
+   p[,i] <- logistico(beta = beta[i], alpha = alpha[i],
+                      ce = ce[i], theta=theta)}
+   for(i in 1:n.ind){
+   y[i,] <- rbinom(n=length(beta), size = 1, p = p[i,])}
+   dados <- data.frame(y = y, ID = 1:100)
+   return(dados)}
```

Para exemplificar o ajuste vamos simular um conjunto de dados com 100 indivíduos e 5 questões. Definimos os seguinte valores para os parâmetros: $\beta_i = (-2, -1,0,1,2)$ e $\alpha = 1$ e $ce = 0$, ou seja, um modelo de Rasch.

```
> set.seed(123)
> dados <- simula.tri(n.ind=100, beta=c(-2,-1, 0, 1, 2), alpha=c(1,1,1,1,1),
+                     ce=c(0,0,0,0,0))
```

A seguir definimos uma função para computar o integrando da função de verossimilhança para um individuo.

Código 3.60: Definição do integrando para modelo de Rasch.

```
> integrando <- function(theta, b1, b2,b3, b4, b5, y, log = FALSE){
+   beta <- c(b1,b2,b3,b4,b5)
+   n.beta <- length(beta)
+   p <- matrix(NA, ncol = n.beta, nrow = 1)
+   ll = sapply(theta,function(thetai){
+     for(i in 1:n.beta){
+       p[,i] <- logistico(beta = beta[i], alpha = 1, ce = 0, theta=thetai)}
+       sum(dbinom(y, size = 1, prob = p, log=TRUE)) +
+           dnorm(thetai,0,1, log=TRUE)})
+   if(log == FALSE){ll <- exp(ll)}
+   return(ll)
+ }
```

Para facilitar vamos usar para a integração numérica a função `integrate()`. Deixamos para o leitor implementar com as outras opções de integração numérica apresentadas.

Código 3.61: Verossimilhança marginal do modelo Rasch.

```
> marginal <- function(b1, b2, b3, b4, b5, dados){
+   y.id <- split(dados, dados$ID)
+   beta <- c(b1,b2,b3,b4,b5)
+   print(round(beta,4))
+   n.beta <- length(beta)
+   integral <- c()
+   for(i in 1:length(y.id)){
+   integral[i] <- integrate(integrando, lower= -Inf, upper = Inf,
+                     b1 = b1, b2 = b2, b3 = b3, b4 = b4, b5 = b5,
+                     y = as.numeric(y.id[[i]][1:n.beta]))$value}
+   ll <- sum(log(integral))
+   return(-ll)
+ }
```

Por fim, podemos maximizar a função de log-verossimilhança marginal, usando o pacote **bbmle**.

bbmle

```
> ajuste <- mle2(marginal, start=list(b1=0,b2=0,b3=0,b4=0,b5=0),
+                data=list(dados=dados))
> summary(ajuste)
Maximum likelihood estimation

Call:
mle2(minuslogl = marginal, start = list(b1 = 0, b2 = 0, b3 = 0,
    b4 = 0, b5 = 0), data = list(dados = dados))

Coefficients:
    Estimate Std. Error z value     Pr(z)
b1 -1.636976   0.286718 -5.7093 1.134e-08 ***
b2 -0.641374   0.247421 -2.5922  0.009535 **
b3  0.090727   0.241169  0.3762  0.706770
b4  1.063558   0.260293  4.0860 4.389e-05 ***
b5  1.864777   0.303266  6.1490 7.799e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

-2 log L: 575.0457
```

Esta é uma implementação simplesmente ilustrativa. Podemos torná-la muito mais rápida computacionalmente, escrevendo o integrando de uma forma mais eficiente e calculando apenas as integrais necessárias.

A seguir apresentamos os comandos para ajustar o mesmo modelo utilizando o pacote **ltm** (Rizopoulos, 2006). Os resultados coincidem com os anteriores porém a implementação é claramente mais eficiente.

```
> require(ltm)
This is package 'ltm' version '0.9-7'
> dados <- dados[,-6]
> rasch(dados, constraint=cbind(length(dados)+1, 1))
```

```
Call:
rasch(data = dados, constraint = cbind(length(dados) + 1, 1))

Coefficients:
Dffclt.y.1  Dffclt.y.2  Dffclt.y.3  Dffclt.y.4  Dffclt.y.5      Dscrmn
    -1.637      -0.641       0.091       1.064       1.865       1.000

Log.Lik: -287.523
```

3.8 Modelo linear dinâmico

Os modelos dinâmicos são uma classe de modelos de regressão usualmente aplicados a séries temporais. Nesses modelos, os coeficientes de regressão variam no tempo. A evolução dos coeficientes é parametrizada de forma a ser suave e dinâmica, originando esse nome. O modelo dinâmico mais simples é o modelo com apenas um intercepto,

$$\begin{aligned} y_t &= \theta_t + v_t, \quad v_t \sim N(0, V) \\ \theta_t - \mu &= \phi(\theta_{t-1} - \mu) + w_t, \quad w_t \sim N(0, W) \end{aligned} \tag{3.9}$$

em que V, W e ϕ são parâmetros de variância, μ é a média do intercepto θ_t.

O maior interesse é a modelagem de θ_t, considerando sua estrutura de evolução e de erro. Pode-se considerar que θ_t é um estado latente (não observável) subjacente ao que se observa. Desta forma, os modelos dinâmicos também são chamados de modelos de *espaço de estados*.

Nesse modelo V mede o erro das observações; μ está associada à média de y, ϕ e W controlam a suavidade da evolução de θ_t. Com $0 \leq \phi \leq 1$ temos um modelo autoregressivo de ordem 1 para θ. Com $\phi = 1$ temos um passeio aleatório para θ e y é um passeio aleatório mais um ruído. Com $\phi = 1$ e W pequeno em relação a V, y_t será parecido com y_{t-1}. West & Harrison (1997) e Petris et al. (2009) apresentam mais detalhes e estruturas mais gerais de modelos dinâmicos. Uma generalização da expressão acima é dada por

$$\begin{aligned} y_t &= F_t\theta_t + v_t \\ (\theta_t - \mu) &= G_t(\theta_t - \mu) + v_t. \end{aligned} \tag{3.10}$$

Aqui, vamos considerar essa especificação geral no contexto de regressão dinâmica. Então, nomeamos os elementos da seguinte forma:

- y_t vetor resposta de dimensão m,
- F_t pode ser uma matriz de covariáveis,
- θ_t vetor de coeficientes de regressão,

- v_t vetor de erros das observações, $v_t \sim N(0, V_t)$,
- μ média dos coeficientes de regressão,
- w_t vetor de erros dos estados, $w_t \sim N(0, W_t)$,
- G_t matriz de evolução dos estados.

São feitas as seguintes suposições de independência condicional:

$$[y_t|y_{0:t}, \theta_{0:t}] = [y_t|\theta_t]$$
$$[\theta_t|\theta_{0:t}, y_{0:t}] = [\theta_t|\theta_{t-1}].$$

Para o modelo linear Gaussiano, temos

$$[y_t|\theta_t] = N(F_t\theta_t, V_t)$$
$$[\theta_t|y] = N(G_t\theta_t, W_t).$$

É comum considerar que $V_t = V$, $W_t = W$ e $G_t = G$, isto é, constantes no tempo. Notemos que, além dessas condições, com $G = I$ e $w_t = 0$ para todo t, temos um modelo de regressão linear usual.

3.8.1 Filtro de Kalman e verossimilhança

Vamos considerar que $\psi_t = \{V_t, W_t, G_t\}$ é o conjunto de parâmetros desconhecidos. Também, vamos considerar que $\psi_t = \psi$, ou seja, os parâmetros são fixos no tempo. Podemos estimar ψ via máxima verossimilhança. A verossimilhança para os modelos lineares dinâmicos leva em conta que

$$[Y] = [Y_1][Y_2|Y_1]\dots[Y_n|Y_{n-1}, \dots, Y_1].$$

Considerando, v_t um erro Gaussiano, $[y_t|\dots]$ também é gaussiano e para ter a verossimilhança, basta conhecer as mas médias e variâncias condicionais de $[y_t|\dots]$ (Schweppe, 1965), que dependem de ψ. O filtro de Kalman, Kalman (1960) fornece um algoritmo para o cálculo dessas médias e variâncias condicionais. Portanto, o filtro de Kalman pode ser usado para calcular a verossimilhança, que é o produto dessas densidades condicionais, Akaike (1978), Harvey & Phillips (1979) Jones (1980) e Gardner et al. (1980) e Harvey (1981).

O filtro de Kalman, Kalman (1960), foi proposto originalmente para correção e filtragem de sinais eletrônicos. Nesse contexto, considera-se que o estado latente é o verdadeiro sinal e o que se observa é o sinal mais um ruído. Este algoritmo esta baseado na distribuição condicional de y_t e θ_t, obtidas, respectivamente, num passo de filtragem e num passo de suavização. Assim, o primeiro passo é usado para calcular a verossimilhança e o segundo para fazer inferência sobre θ_t.

Na filtragem, usam-se as equações para a predição de θ_t, isto é, $\theta_t|\theta_{t-1} = \theta_t^{t-1}$ e para obter θ_t filtrado: θ_t^t. Também, obtêm-se P_t^{t-1} e P_t^t, que é, respectivamente, P_t predito e filtrado. E na suavização obtêm-se θ_t^n e P_t^n, que são estimativas da média e variância de θ_t.

As equações de predição, para $t = 1,2,\ldots,n$, são:

$$\theta_t^{t-1} = G\theta_{t-1}^{t-1} \tag{3.11}$$

$$P_t^{t-1} = GP_{t-1}^{t-1}G^{'} + W \tag{3.12}$$

com $\theta_0^0 = \mu_0$ e $P_0^0 = C_0$. μ_0 e C_0 são, respectivamente, a média e a variância de θ_0. Para considerações sobre μ_0 e C_0 ver De Jong (1988).

As equações de filtragem, para $t = 1,2,\ldots,n$, são:

$$e_t = y_t - F_t\theta_t^{t-1} \tag{3.13}$$

$$Q_t = F_tP_t^{t-1}F_t^{'} + V \tag{3.14}$$

$$K_t = P_t^{t-1}F_t^{'}Q_t^{-1} \tag{3.15}$$

$$\theta_t^t = \theta_t^{t-1} + K_te_t \tag{3.16}$$

$$P_t^t = P_t^{t-1} - K_tF_tP_t^{t-1} \tag{3.17}$$

Na suavização, obtém-se os valores suavizados de θ_t, isto é, $\theta_t^n = \hat{\theta}$, a estimativa de θ_t. Também, obtêm-se P_t^n, o valor suavizado de P_t. P_t^n quantifica a incerteza de $\hat{\theta}_t$. Inicialmente, para $t = n$ têm-se: $\theta_n^n = \theta_t^t$, $P_n^n = P_t^t$, apenas para $t = n$ Para $t = n, n-1, \ldots, 1$, têm-se:

$$J_{t-1} = P_{t-1}^{t-1}G^{'}(P_t^{t-1})^{-1} \tag{3.18}$$

$$\theta_{t-1}^n = \theta_{t-1}^{t-1} + J_{t-1}(\theta_t^n - \theta_t^{t-1}) \tag{3.19}$$

$$P_{t-1}^n = P_{t-1}^{t-1} + J_{t-1}(P_t^n - P_t^{t-1})J_{t-1}^{'} \tag{3.20}$$

Com e_t e Q_t obtidos no passo de filtragem do filtro de Kalman e considerando que ambos dependem de ψ, temos

$$l(\psi) = -\frac{1}{2}\sum_{t=1}^{n}\log|Q_t| - \frac{1}{2}\sum_{t=1}^{n}e_t^{'}Q_t^{-1}e_t. \tag{3.21}$$

3.8.2 Exemplo simples

Vamos considerar o caso mais simples, descrito no início desta seção, considerando $\mu = 0$. Neste caso, temos um intercepto que evolui dinamicamente. O número de parâmetros neste modelo é três: variância do erro das observações, variância do erro dos estados e parâmetro de evolução dos estados.

Inicialmente, vamos simular um conjunto de dados.

```
> n <- 100
> V <- .2; W <- .3; rho <- 0.9
> set.seed(1)
> w <- rnorm(n, 0, sqrt(W))
> v <- rnorm(n, 0, sqrt(V))
> x <- w[1]
> for (i in 2:n)
+   x[i] <- rho*x[i-1] + w[i]
> y <- x + v
```

Vamos definir uma função que crie uma lista contendo contendo as componentes básicas n, m e k, representando as dimensões, as componentes y, F, x, V, W e G, para representar os componentes do modelo, respectivamente: y_t, F_t, θ_t, V, W e G. Além disso, essa lista também incluirá os elementos $x.p$, $x.f$, $x.s$, $P.p$, $P.f$ e $P.s$, para representar os estados e variâncias preditos, filtrados e suavizados, respectivamente; e os elementos e e Q, para representar os elementos usados no cálculo da verossimilhança. Também, colocamos os elementos `m0` e `c0`, para representar μ_0 e C_0.

Esta função é particular para este exemplo, com V, G e W escalares:

Código 3.62: Estrutura para o modelo com intercepto dinâmico intercepto.

```
> ddlm1 <- function(y, G, V, W, m0=0, c0=1e5) {
+   ## y: vetor de dados
+   ## G: coeficiente autoregressivo
+   ## V: variância dos erros de observação
+   ## W: variância dos erros do estado
+   ## m0, c0: media e variância de theta.0
+   dlm <- mget(ls(), environment()) ## lista com argumentos
+   n <- length(y)
+   dlm[c("n", "F", "e", "S")] <- list(n, rep(1,n), rep(0, n), rep(V, n))
+   dlm[c("x.s", "x.f", "x.p")] <- rep(list(rep(m0, n+1)), 3)
+   dlm[c("P.s", "P.f", "P.p")] <- rep(list(rep(W,  n+1)), 3)
+   ## retorna uma lista (nomeada) com argumentos e os elementos:
+   ## n: tamanho da serie
+   ## F: vetor de 1's
+   ### elementos predefinidos para armazenar resultados:
+   ## x.p, x.f e x.s; P.p, P.f, P.s e P.t; e, S
+   return(dlm)
+ }
```

Usando essa função para criar a estrutura de modelo dinâmico para os dados simulados

```
> mod1 <- ddlm1(y, 0, 100, 100)
```

Agora, definimos uma função para a filtragem específica para este exemplo, com V, G e W escalares:

Código 3.63: Filtragem para o modelo dinâmico simples: intercepto dinâmico.

```
> kfilter1 <- function(dlm) {
+   ## dlm: saída de funcao ddlm1() com componentes do modelo
+   for (i in 1:dlm$n+1) {
+     dlm <- within(dlm, {
+       x.p[i] <- G*x.f[i-1]
+       P.p[i] <- G*P.f[i-1]*t(G) + W
+       e[i-1] <- y[i-1] - F[i-1]*x.p[i]
+       S[i-1] <- F[i-1]*P.p[i]*F[i-1] + V
+       K <- (P.p[i]*F[i-1])/S[i-1]
+       x.f[i] <- x.p[i] + K*e[i-1]
+       P.f[i] <- P.p[i] - K*F[i-1]*P.p[i]
+     })
+   }
+   ## retorna objeto lista com valores atualizados dos
+   ## componentes x.p, P.p e, S, x.f e P.f definidos em dlm1()
+   return(dlm[1:16])
+ }
```

Considerando que temos y_t univariado, a função para calcular o negativo da log-verossimilhança pode ser simplesmente:

```
> nlldlm1 <- function(dlm)
+   -sum(dnorm(dlm$e, 0.0, sqrt(dlm$S), log=TRUE))
```

A busca do máximo da função de verossimilhança pode ser feita usando a função `optim()`. Vamos definir, então, uma função que recebe os parâmetros de interesse e retorna o negativo da log-verossimilhança do modelo.

```
> opfun1 <- function(pars, dlm) {
+   ## pars: vetor valores dos três parâmetros do modelo simples
+   ## dlm: objeto com componentes do modelo simples
+   dlm[c("V", "W", "G")] <- pars
+   dlm <- kfilter1(dlm)
+   ## retorna o negativo do log-verossimilhanca
+   return(nlldlm1(dlm))
+ }
```

Agora, vamos considerar valores iniciais e usar a função `optim()` para encontrar as estimativas. Vamos usar o método `L-BFGS-B` que considera restrição no espaço paramétrico, estrição passada pelos argumentos `lower` e `upper`.

```
> op1 <- optim(c(1,1,.5), opfun1, dlm=mod1,
+              hessian=TRUE, method="L-BFGS-B",
+              lower=rep(1e-7,3), upper=c(Inf, Inf, 1))
> op1$par                       # estimativas
[1] 0.1713409 0.3146996 0.8235336

> sqrt(diag(solve(op1$hess))) # erros padrão
[1] 0.08644305 0.12450556 0.07914220
```

Com a estimativa desses parâmetros, podemos obter a estimativa de θ_t. Para isso, precisamos implementar também o passo de suavização. Vamos definir uma função para isso.

Código 3.64: Suavização para o modelo com intercepto dinâmico.

```
> ksmooth1 <- function(dlm) {
+   ## dlm: lista com componentes do modelo dinamico simples,
+   ##      suavizada com a funcao kfilter1().
+   dlm <- within(dlm, {
+     x.s[n+1] <- x.f[n+1]
+     P.s[n+1] <- P.f[n+1]
+   })
+   for (i in dlm$n:2) {
+     J <- with(dlm, P.f[i]*G/P.p[i+1])
+     dlm <- within(dlm, {
+       x.s[i] <- x.f[i] + J*(x.s[i+1]-x.p[i+1])
+       P.s[i] <- P.f[i] + J*(P.s[i+1]-P.p[i+1])*J
+     })
+   }
+   ## retorna objeto lista com valores atualizados dos
+   ## componentes x.s e P.s definidos em dlm1()
+   return(dlm)
+ }
```

Definindo a série com os valores estimados, aplicamos o filtro e a suavização à série filtrada.

```
> fit1 <- ddlm1(y, op1$par[3], op1$par[1], op1$par[2])
> fit1f <- kfilter1(fit1)
> fit1s <- ksmooth1(fit1f)
```

Em R a estimação via máxima verossimilhança pode ser feita utilizando o pacote **dlm** (Petris et al., 2009). O pacote é voltado para estimação bayesiana. É possível obter estimativas de máxima verossimilhança, considerando-se que a matriz G é conhecida. Para ilustração fazemos uma comparação dos resultados obtidos e com resultados obtidos com o pacote **dlm**. Neste caso, considerando vamos considerar $G = 1$.

```
> require(dlm)
> m.build <- function(pars) {
+     m <- dlmModPoly(1, dV = pars[1], dW = pars[2])
+     m["GG"] <- pars[3]
+     m
+ }
> y.mle <- dlmMLE(y, rep(1, 3), m.build, hessian = TRUE, lower = rep(1e-07,
+     3), upper = c(Inf, Inf, 1))
> rbind(op1$par, y.mle$par)
          [,1]      [,2]      [,3]
[1,] 0.1713409 0.3146996 0.8235336
[2,] 0.1712436 0.3148899 0.8233935
```

```
> round(rbind(sqrt(diag(solve(op1$hess))), sqrt(diag(solve(y.mle$hess)))),
+     4)
       [,1]   [,2]   [,3]
[1,] 0.0864 0.1245 0.0791
[2,] 0.0865 0.1246 0.0792
> y.mod <- m.build(y.mle$par)
> y.filt <- dlmFilter(y, y.mod)
> y.smoo <- dlmSmooth(y.filt)
> xm.se <- sqrt(unlist(dlmSvd2var(y.smoo$U.S, y.smoo$D.S)))
```

Figura 3.9 podemos visualizar os valores de θ_t simulados e estimados por ambas, pelas funções que definimos e as do pacote **dlm** .

```
> par(mar=c(4,4,.5,.1), mgp=c(2.5, 1, 0), las=1)
> ps.options(horizontal=FALSE)
> ylm <- range(y.smoo$s[-1] -2*xm.se[-1],
+              y.smoo$s[-2] +2*xm.se[-1])
> plot.ts(x, ylim=ylm, lwd=3,
+         ylab=expression(theta), xlab="")
> lines(fit1s$x.s[-1], lty=2, lwd=2, col="gray")
> er1 <- sqrt(fit1s$P.s[-1])
> lines(fit1s$x.s[-1] - 1.96*er1, lty=2, lwd=2, col="gray")
> lines(fit1s$x.s[-1] + 1.96*er1, lty=2, lwd=2, col="gray")
> lines(y.smoo$s[-1], lty=3, lwd=3)
> lines(y.smoo$s[-1] - 1.96*xm.se[-1], lty=3, lwd=3)
> lines(y.smoo$s[-1] + 1.96*xm.se[-1], lty=3, lwd=3)
> legend("topleft", c("Real", "Implementado", "Pacote 'dlm'"),
+        lty=1:3, lwd=1:3, bty="n", col=c(1,"gray",1))
```

3.8.3 Exemplo de regressão dinâmica

Vamos agora considerar um modelo dinâmico de regressão com uma covariável. Neste exemplo, as equações de filtragem são bastante simplificadas. Nesse modelo, temos como parâmetros: a variância das observações V, a variância W de w_t e a matriz G. Vamos considerar W e G diagonais. Portanto, temos cinco parâmetros a serem estimados.

Inicialmente, vamos simular um conjunto de dados que contém agora uma covariável.

```
> n <- 100    # define tamanho da série
> set.seed(1) # simula covariável
> x <- rbind(1, 3*runif(n), rexp(n,1))
> W <- diag(c(0.3, 0.2, 0.1))
> ### simula erros dos estados
> w <- rbind(rnorm(n, 0, sqrt(W[1,1])),
+            rnorm(n, 0, sqrt(W[2,2])),
+            rnorm(n, 0, sqrt(W[3,3])))
> V <- 0.1 # variância do erro nas observações
> v <- rnorm(n, 0, sqrt(V)) # erro nas observações
> G <- diag(c(0.7, 0.8, 0.9)) # matriz de evolução dos estados
```

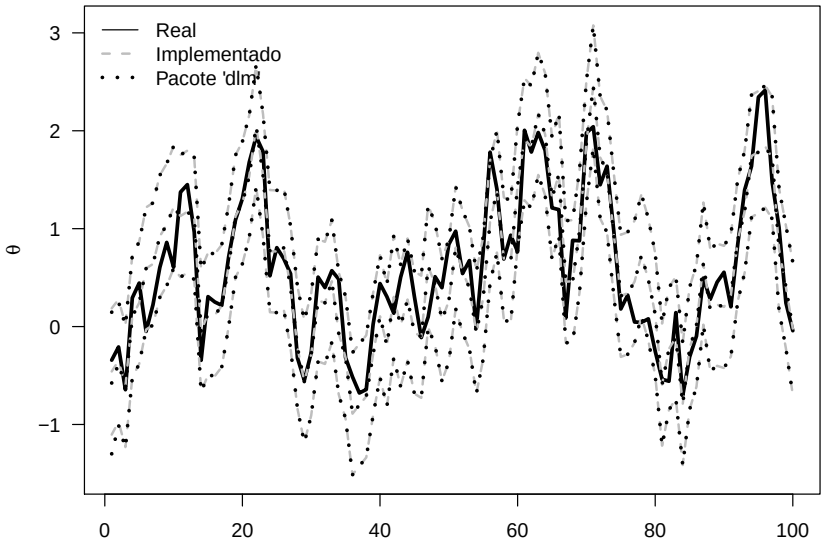

Figura 3.9: Estado latente simulado e estimado e respectivos intervalos de confiança obtidos pelas funções implementadas e as do pacote 'dlm'.

```
> ### simula estados e observações
> theta <- matrix(NA, nrow(W), n)
> theta[,1] <- w[,1]
> y <- rep(NA, n)
> y[1] <- x[,1]%*%theta[,1] + v[1]
> for (i in 2:n) {
+   theta[,i] <- G%*%theta[,i-1] + w[,i]
+   y[i] <- x[,i]%*%theta[,i] + v[i]
+ }
```

Vamos definir uma função que crie uma lista contendo contendo as componentes do modelo. Esta lista é semelhante à criada no exemplo anterior, porém, para o contexto de regressão dinâmica.

Código 3.65: Estrutura para o modelo de regressão dinâmica.

```
> ddlm1r <- function(y, F, G, V, W,
+             m0=rep(0,ncol(W)), V0=diag(ncol(W))*1000) {
+   ## y: vetor de dados , F: matriz de covariaveis
+   ## G: matriz diagonal com coeficientes autoregressivos
+   ## V: variância dos erros de observação
+   ## W: matriz diagonal de variância dos erros dos estados
+   ## m0 e c0: vetor media e matrix variancia de theta.0
+   dlm <- mget(ls(), environment())
+   n=length(y)  ;  k=nrow(W)
+   dlm[c("n", "k")] <- c(n, k)
+   dlm[c("x.s","x.f","x.p")] <- rep(list(matrix(m0, k, n+1)), 3)
+   dlm[c("P.s","P.f","P.p")] <- rep(list(array(V0,c(dim(W),n+1))),3)
+   dlm[c("e", "S")] <- list(rep(0, n), rep(V, n))
+   ## retorna uma lista com os adicionais elementos:
+   ## n: tamanho da serie ;  k: dimensão dos estados
+   ## e elementos predefinidos armazenar resultados
+   return(dlm)
+ }
```

Agora, definimos uma função para a filtragem:

Código 3.66: Filtragem para o modelo de regressão dinâmica.

```
> kfilter1r <- function(dlm) {
+   ## dlm: lista criada com funcao ddlm1r()
+   for (i in 1:dlm$n+1) {
+     dlm <- within(dlm, {
+       x.p[,i] <- G%*%x.f[,i-1]
+       aux <- tcrossprod(P.f[,,i-1], G)
+       P.p[,,i] <- G%*%aux + W
+       e[i-1] <- y[i-1] - F[,i-1]%*%x.p[,i]
+       aux <- crossprod(P.p[,,i], F[,i-1])
+       S[i-1] <- F[,i-1]%*%aux + V
+       K <- (P.p[,,i]%*%F[,i-1])/S[i-1]
+       x.f[,i] <- x.p[,i] + K*e[i-1]
+       aux <- K%*%F[,i-1]
+       P.f[,,i] <- P.p[,,i] - aux%*%P.p[,,i]
+     })
+   }
+   ## retorna objeto lista com valores atualizados dos
+   ## componentes x.p, P.p e, S, x.f e P.f definidos em dlm1r()
+   return(dlm[1:17])
+ }
```

Considerando que temos y_t univariado, a função para calcular o negativo da log-verossimilhança é a mesma do exemplo anterior. Precisamos, apenas, definir outra função que recebe os parâmetros de interesse e retorna

o negativo da log-verossimilhança do modelo.

Código 3.67: Função dos parâmetros do modelo de regressão dinâmica para minimizar.

```
> opfun1r <- function(pars, dlm) {
+   ## pars: vetor com 2*k + 1 parâmetros
+   ## dlm: lista com componentes do modelo de regressao
+   ##     dinamica (k=numero de covariaveis/estados)
+   k <- (length(pars)-1)/2
+   dlm[c("V", "W", "G")] <-
+       list(pars[1], diag(pars[1+1:k]), diag(pars[1+k+1:k]))
+   dlm <- kfilter1r(dlm)
+   return(nlldlm1(dlm))
+ }
```

Agora, vamos considerar valores iniciais e usar a função `optim()` para encontrar as estimativas.

```
> mod1r <- ddlm1r(y=y, F=x, G=0.5*diag(3), V=1, W=diag(3))
> op1r <- optim(rep(.5, 7), opfun1r, dlm=mod1r, method="L-BFGS-B",
+               lower=rep(1e-5,7), upper=rep(c(Inf, 1), c(4,3)),
+               hessian=TRUE)
> ### estimativas obtidas
> round(op1r$par, 4)

[1] 0.1631 0.1598 0.1969 0.1099 0.8031 0.6440 0.8027

> ### calculando os erros padroes das estimativas
> round(sqrt(diag(solve(op1r$hess))), 4)

[1] 0.1481 0.1873 0.0633 0.0727 0.1787 0.1200 0.1105
```

Agora, precisamos implementar a suavização. Neste caso, observando as equações 3.19, observamos que, quando θ_t é vetor (W é matriz), temos que fazer a conta $P_{t-1}^{t-1} G' (P_t^{t-1})^{-1}$ em cada passo. Aqui é bom considerar que essa matriz é simétrica e ter um ganho computacional. Uma opção para isso, é fazer a decomposição de Choleski, que é uma matriz triangular, e fazer a inversão dessa matriz, considerando sua estrutura. Em R usamos `chol2inv(chol(M))`, para inverter uma matriz quadrada simétrica e positiva definida `M`.

Outra consideração sobre contas matriciais é sobre o cálculo de `A'B`. Em R é mais rápido fazer `crossprod(A, B)` que fazer simplesmente `t(A) %*% B` para esse cálculo. Porém, se precisamos calcular `AB` (sem transpor qualquer das duas matrizes), é melhor usar `A %*% B` que usar `crossprod(t(A),B)`, se as matrizes são de dimensões pequenas.

Desta forma, definimos a seguinte função

Código 3.68: Suavização para o modelo de regressão dinâmica.

```
> ksmooth1r <- function(dlm) {
+   ## dlm: lista com componentes do modelo de regressão dinâmica.
+   ##      suavizada com a funcao kfilter1r().
+   dlm <- within(dlm, {
+     x.s[,n+1] <- x.f[,n+1]
+     P.s[,,n+1] <- P.f[,,n+1]
+   })
+   for (i in dlm$n:1) {
+     dlm <- within(dlm, {
+       aux <- crossprod(G,chol2inv(chol(P.p[,,i+1])))
+       J <- P.f[,,i]%*%aux
+       x.s[,i] <- x.f[,i] + J%*%(x.s[,i+1]-x.p[,i+1])
+       aux <- tcrossprod(P.s[,,i+1]-P.p[,,i+1], J)
+       P.s[,,i] <- P.f[,,i] + J%*%aux
+     })
+   }
+   ## retorna objeto lista com valores atualizados dos
+   ## componentes x.s e P.s definidos em dlm1r()
+   return(dlm[1:17])
+ }
```

Definindo a série com os valores estimados

```
> fit1r <- ddlm1r(y, x, diag(op1r$par[5:7]), op1r$par[1],
+                 diag(op1r$par[2:4]))
```

Aplicando o filtro e a suavização à série filtrada.

```
> kf1r <- kfilter1r(fit1r)
> ks1r <- ksmooth1r(kf1r)
```

Extraindo a variância dos estados (diagonal de P_t^n), para facilitar o cálculo e visualização dos intervalos de confiança para θ_t:

```
> th.se <- sqrt(apply(ks1r$P.s[,,-1], 3, diag))
```

Vamos implementar o mesmo modelo usando funções do pacote **dlm**.

```
> k <- nrow(x)
> mr.bf <- function(pars) {
+   ## pars: vetor de 2*k+1 parametros do modelo regressao
+   ##     dinamica (k=numero de covariaveis/estados)
+   m <- dlmModReg(t(x[-1,]), dV=pars[1], dW=pars[1+1:k])
+   m$GG <- diag(pars[1+k+1:k])
+   ## retorna objeto dlm com dV, dW e GG atualizados
+   m
+ }
> opd1r <- dlmMLE(y, rep(1, 1+2*k), mr.bf,
+                 lower=rep(1e-7, 1+2*k), hessian=TRUE)
> ### estimativas:
> round(rbind(op1r$par, opd1r$par), 4)
      [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]
[1,] 0.1631 0.1598 0.1969 0.1099 0.8031 0.6440 0.8027
[2,] 0.1608 0.1635 0.1972 0.1099 0.7996 0.6413 0.8021
```

```
> ### erros padrões
> round(rbind(sqrt(diag(solve(op1r$hess))),
+             sqrt(diag(solve(opd1r$hess)))), 4)

       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]
[1,] 0.1481 0.1873 0.0633 0.0727 0.1787 0.1200 0.1105
[2,] 0.1507 0.1926 0.0635 0.0730 0.1818 0.1227 0.1114

> ### monta modelo com estimativas obtidas
> modd1r <- mr.bf(opd1r$par)
> ### filtragem e  suavização
> kfd1r <- dlmFilter(y, modd1r)
> ksd1r <- dlmSmooth(kfd1r)
> ### extrai erros dos estados
> xm.se1r <- sqrt(sapply(dlmSvd2var(ksd1r$U.S, ksd1r$D.S), diag))
```

Notamos algumas diferenças nos valores das estimativas e respectivos erros padrões. As funções do pacote **dlm** usam a decomposição em valores singulares em vez da decomposição de Choleski. Na Figura (3.10), podemos visualizar θ_t simulados e estimados:

```
> par(mfrow=c(3,1), mar=c(2,2,1,.1), mgp=c(1, .3, 0), las=1)
> for (i in 1:3) {
+   er1 <- 1.96*th.se[i,]
+   er2 <- 1.96*xm.se1r[i,-1]
+   ylm <- range(ks1r$x.s[i,-1]-er1, ks1r$x.s[i,-1]+er1)
+   plot.ts(theta[i,], ylab="", ylim=ylm)
+   lines(ks1r$x.s[i,-1], lwd=2, lty=2, col="gray")
+   lines(ks1r$x.s[i,-1] - er1, lwd=2, lty=2, col="gray")
+   lines(ks1r$x.s[i,-1] + er1, lwd=2, lty=2, col="gray")
+   lines(ksd1r$s[-1,i], lwd=3, lty=3)
+   lines(ksd1r$s[-1,i] - er2, lwd=3, lty=3)
+   lines(ksd1r$s[-1,i] + er2, lwd=3, lty=3)
+ }
> legend("bottomright", c("Real", "Implementado", "Pacote 'dlm'"),
+        lty=1:3, col=c(1,"gray", 1), lwd=1:3, bty="n")
```

Na estimação de θ_t, e sua variância, em ambos os exemplos de modelos dinâmicos, foi considerado a estimativa de máxima verossimilhança de ψ obtida. Ou seja, não está sendo considerada a incerteza dessa estimativa na inferência sobre θ_t. Uma opção para isso é considerar um método baseado no algoritmo EM, no qual ψ e θ são estimados conjuntamente. Nos complementos online deste livro estarão exemplos com essa abordagem.

Uma solução natural para esse tipo de problema é a abordagem de inferência Bayesiana. Essa abordagem é bastante comum na literatura de modelos dinâmicos. Isto será ilustrado em versões futuras deste material.

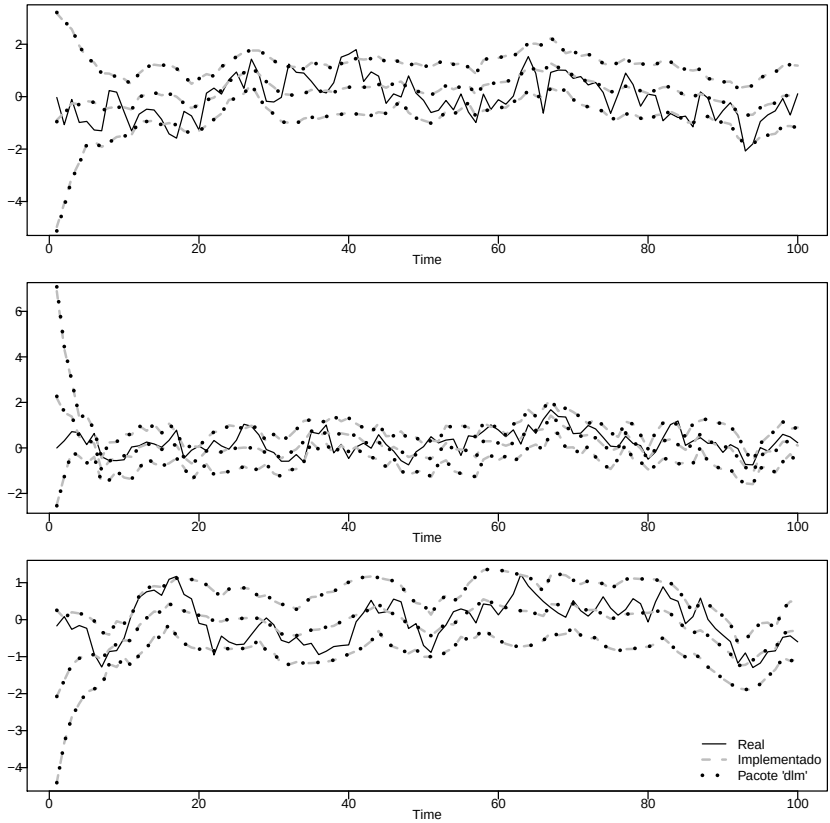

Figura 3.10: Coeficientes de regressão simulados, e estimados (e respectivos intervalos de confiança) via funções implementadas e via funções do pacote 'dlm'.

3.9 Referências

A presente versão do texto trata apenas de inferência baseada na função de verossimilhança. No Capítulo 1 são resumidos os principais conceitos utilizados nos exemplos. Diversas referências estão disponíveis na literatura algumas indicações são os livros de Cox & Hinkley (1979), Azzalini (1996), Royall (1997) e Pawitan (2001). Davison (2001) apresenta os passos no desenvolvimento da metodologia estatística incluindo os conceitos utilizados aqui sob uma perspectiva histórica. Uma breve e informativa revisão foi escrita por Reid (2010).

O Capítulo 2 trata de modelos de regressão lineares e lineares generalizados e uma a referência básica é o livro de McCullagh & Nelder (1989).

Referências adicionais são Cordeiro & Demétrio (2011), Paula (2010) e Dobson & Barnett (2008). Uma suscinta e elegante revisão de modelos lineares, lineares generalizados e aditivos generalizados é apresentada em VENABLES & DICHMONT (2004). Uma apresentação com ênfase no uso em R é feita por Fox & Weisberg (2011). Dentro deste capítulo utilizamos o modelo de regressão Simplex, e nos baseamos na dissertação de mestrado de Miyashiro (2008).

O Capítulo 3 apresenta os modelos de regressão com efeitos aleatórios, nele apresentamos como exemplo inicial o modelo geoestatístico, cuja principal referência é Diggle & Ribeiro Jr (2007). Na sequência apresentamos diversas técnicas de integração numérica. Para os métodos tradicionais, retângulo, trapézio livros tradicionais de cálculo numérico são ótimas referência um exemplo é o livro de Gilat & Subramaniam (2010). Os métodos de Gauss-Hermite, Laplace e adaptative Gauss-Hermite são descritos em Molenberghs & Verbeke (2005). Alguns métodos computacionais e sua implementação em R são apresentados em Rizzo (2008). Para os métodos de integração Monte Carlo, a referência que indicamos é Robert & Casella (2004) e os mesmos autores possuem um texto muito didático voltado para uso em R (Robert & Casella, 2010).

O programa R (R Development Core Team, 2012) é usado ao longo do livro. Na página o programa (www.r-project.org) há uma extensa lista de referências. Uma breve e excelente introdução é o texto de Dalgaard (2008). O livro MASS (Venables & Ripley, 2002) é mais amplo em conteúdos e é ainda fortemente recomendado para uma familiarização com recursos do programa. A site *r-bloggers* reúne diversos blogs sobre o programa. A lista brasileira do R - R-br (www.leg.ufpr.br/rbr) tem sido ativa e excelente fórum em língua portuguesa. Qualquer lista de referências será incompleta devido a extensão de materiais hoje disponíveis.

Diversos pacotes adicionais do Rforam utilizados. Entre eles usamos com mais frequencia o pacote **rootSolve** Soetaert & Herman (2009), Soetaert (2009) e o pacote *bbmle* Bolker & Team (2012).

Referências Bibliográficas

AKAIKE, H. Covariance matrix computations of the state variable of a stationary Gaussian process. **Annals of the Institute of Statistical Mathematics**, v.30, n.B, p.499–504, 1978.

AZZALINI, A. **Statistical Inference Based on the likelihood**. Chapman and Hall/CRC, 1996. 352p.

BOLKER, B.; TEAM, R. D. C. **bbmle: Tools for general maximum likelihood estimation**, 2012. R package version 1.0.4.1.

CORDEIRO, G. M.; DEMÉTRIO, C. G. B. Modelos Lineares Generalizados e Extensões, 2011.

COX, D.; HINKLEY, D. **Theoretical Statistics**. Chapman and Hall/CRC, 1979. 528p.

DALGAARD, P. **Introductory Statistics with R**. 2. ed. Springer, 2008. 380p.

DAVISON, A. C. Biometrika Centenary: Theory and general methodology. Reprinted in Biometrika: One Hundred Years, edited by D. M. Titterington and D. R. Cox., Oxford University Press. **Biometrika**, v.88, p.13–52, 2001.

DE JONG, P. The likelihood for a state space model. **Biometrika**, v.75, n.1, p.165–169, 1988.

DIGGLE, P.; RIBEIRO JR, P. J. **Model-based Geostatistics**. Springer, 2007. 243p.

DIGGLE, P. J.; RIBEIRO JR., P. J. **Model Based Geostatistics**. New York: Springer, 2007. 230p.

DOBSON, A. J.; BARNETT, A. **An Introduction to Generalized Linear Models**. Chapman and Hall/CRC, 2008. 320p.

FOX, J.; WEISBERG, S. **An R Companion to Applied Regression**. 2. ed. Thousand Oaks, CA, USA: Sage Publications, 2011.

GARDNER, G.; HARVEY, A. C.; PHILLIPS, G. D. A. An algorithm for exact maximum likelihood estimation by means of Kalman filtering. **Applied Statistics**, v.29, n.3, p.311–322, 1980.

GILAT, A.; SUBRAMANIAM, V. **Numerical Methods with MATLAB**. Wiley, 2010. 512p.

HARVEY, A. C. **Time Series Models**. Wiley, 1981.

HARVEY, A. C.; PHILLIPS, G. D. A. Maximum likelihood estimation of regression models with autoregressive-moving average disturbances. **Biometrika**, v.66, n.1, p.49–58, 1979.

JONES, R. H. Maximum likelihood fitting of ARMA models to time series with missing observations. **Technometrics**, v.22, n.3, p.389–395, 1980.

KALMAN, R. E. A new approach to linear filtering and prediction problems. **Journal of Basic Engineering**, v.82, n.1, p.35–45, 1960.

MCCULLAGH, P.; NELDER, J. A. **Generalized Linear Models**. Second ed ed. Chapman and Hall/CRC, 1989. 532p.

MIYASHIRO, E. S. Modelos de regressão beta e simplex para a análise de proporções. Universidade de São Paulo, São Paulo, 2008. Rel. téc.

MOLENBERGHS, G.; VERBEKE, G. **Models for Discrete Longitudinal Data (Springer Series in Statistics)**. Springer, 2005. 709p.

NELDER, J. A.; WEDDERBURN, R. M. Generalized linear models. **Journal of the Royal Statistical Society, Series A**, v.135, p.370–84, 1972.

PAULA, G. A. Modelos de Regressão com apoio computacional, 2010.

PAWITAN, Y. **In All Likelihood: Statistical Modelling and Inference Using Likelihood**. Oxford University Press, USA, 2001. 544p.

PETRIS, G.; PETRONE, S.; CAMPAGNOLI, P. **Dynamic Linear Models with R**. Springer-Verlag, New York, 2009. useR!

R DEVELOPMENT CORE TEAM. **R: A Language and Environment for Statistical Computing**. R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.

REID, N. Likelihood Inference. **WIREs Comp Stat**, v.2, p.517–525, 2010.

RIZOPOULOS, D. ltm: An R package for Latent Variable Modelling and Item Response Theory Analyses. **Journal of Statistical Software**, v.17, n.5, p.1–25, 2006.

RIZZO, M. L. **Statistical Computing with R**. Boca Raton, FL: Chapman & Hall/CRC, 2008.

ROBERT, C.; CASELLA, G. **Monte Carlo Statistical Methods (Springer Texts in Statistics)**. Springer, 2004. 675p.

ROBERT, C.; CASELLA, G. **Introducing Monte Carlo Methods with R**. Springer, 2010. Use R.

ROYALL, R. **Statistical evidence**. Chapman and Hall, 1997.

SCHWEPPE, F. C. Evaluation of likelihood functions for Gaussian signals. **IEEE Trans. Info. Theory**, v.11, p.61–70, 1965.

SOETAERT, K. **rootSolve: Nonlinear root finding, equilibrium and steady-state analysis of ordinary differential equations**, 2009. R package 1.6.

SOETAERT, K.; HERMAN, P. M. **A Practical Guide to Ecological Modelling. Using R as a Simulation Platform**. Springer, 2009. 372p. ISBN 978-1-4020-8623-6.

VENABLES, W. N.; DICHMONT, C. M. GLMs, GAMs and GLMMs: an overview of theory for applications in fisheries research. **Fisheries Research**, v.70, p.319–337, 2004.

VENABLES, W. N.; RIPLEY, B. D. **Modern Applied Statistics with S. Fourth Edition**. New York: Springer, 2002.

WEST, M.; HARRISON, P. J. **Bayesian Forecasting & Dynamic Models**. 2. ed. Springer Verlag, 1997.

WINKELMANN, R. Duration Dependence and Dispersion in Count-Data Models. **Journal of Business & Economic Statistics**, v.13, n.4, p.467–474, 1995.