20ª SINAPE

Simpósio Nacional de Probabilidade e Estatística

MINICURSO

Métodos Computacionais em Inferência Estatística

Wagner Hugo Bonat

Paulo Justiniano Ribeiro Jr

Elias Teixeira Krainski

Walmes Marques Zeviani

Métodos Computacionais em Inferência Estatística

Wagner Hugo Bonat
Paulo Justiniano Ribeiro Jr
Elias Teixeira Krainski
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: {paulojus,walmes,wbonat,eliaskr}@ leg.ufpr.br
Última revisão do texto: 16 de fevereiro de 2018

20ª SINAPE
Simpósio Nacional de Probabilidade
e Estatística
João Pessoa - PB - Brasil
30 de julho a 3 de agosto 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. Nossa expectativa é a de que o texto possa servir como material introdutório ao leitor e facilitar seu caminho para construir suas próprias implementações em modelagens de seu interesse.

O material foi motivado por nossa experiência em grupos de estudos e disciplinas conduzidas no âmbito do LEG/UFPR (Laboratório de Estatística e Geoinformação da Universidade Federal do Paraná) 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 e códigos apresentados são predominantemente ilustrativos, privilegiando a facilidade de leitura e entendimento. Os códigos não devem ser vistos como implementações definitivas nem tampouco tentam explorar usos mais eficientes 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 final 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* disponibilizada 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 tivemos oportunidade de expandir o material acrescentando novos tópicos. 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

Capítulo 1

Introdução

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 para 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 está 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 um modelo gaussiano, com média μ e variância σ^2, o espaço paramétrico é $\Re \times \Re_+$ ou seja, todo o conjunto dos reais para média μ enquanto que para σ^2 o espaço paramétrico restringe-se aos reais não negativos. Outro caso comum são modelos em que o parâmetro representa alguma proporção p 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 do espaço original ao final. Por exemplo pode-se adotar $\psi = \log\sigma$ para variância do modelo normal e $\psi = \log p/(1-p)$ para a proporção.

Partindo destes conceitos, um fenômeno aleatório ou estocástico é descrito minimamente por uma distribuição de probabilidade, que por sua vez é indexada 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 plebiscito. Suponha que n eleitores selecionados aleatoriamente são questionados sobre a sua intenção em votar a favor (1) ou contra (0) uma determinada mudança na legislação. Deseja-se estimar a proporção θ de eleitores favoráveis à mudança. Assume-se que o modelo Bernoulli seja adequado para a intenção de voto de cada eleitor e portanto o número de favoráveis em uma amostra aleatória de n eleitores tem distribuição binomial $\mathrm{B}(n,\theta)$. Este modelo tem como possíveis respostas os valores 0 e 1 e como parâmetro indexador θ que representa a proporção de

favoráveis e tem o intervalo unitário como seu **espaço paramétrico**. Com este conjunto de suposições e conhecimentos a respeito do modelo probabilístico, tem-se total condições de fazer **inferência** sobre o parâmetro θ a partir dos dados de uma amostra. No gráfico da esquerda da Figura 1.1 representamos a região definida pelo modelo para uma amostra aleatória de tamanho 100. A superfície é obtida calculando-se os valores das probabilidades de se observar y favoráveis em uma amostra para cada um dos possíveis valores do parâmetro. Para visualização omitimos os valores próximos às bordas $[0,1]$. Um corte da superfície de probabilidades em um particular valor do parâmetro fornece uma *distribuição de probabilidades* para as possíveis respostas como ilustrado no gráfico do centro para $\theta = 0{,}65$. Um corte para um valor de $y = 60$, que poderia ser o obtido em uma determinada amostra, fornece a *função de verossimilhança* $L(\theta|y)$ do gráfico à direita que fornece a medida de proximidade da cada possível valor do parâmetro à amostra.

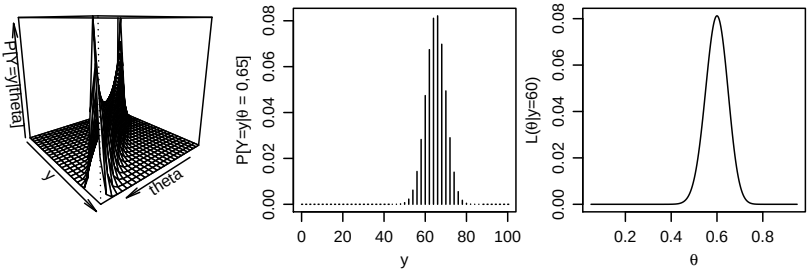

Figura 1.1: Superfície de probabilidades (esquerda), distribuição de probabilidades (centro) e função de verossimilhança (direita) para um modelo $B(n = 100, \theta)$.

Em outras palavras, no gráfico da função de verossimilhança $L(\theta)$, a ordenada de cada ponto da curva é dada pela probabilidade do valor y observado na amostra, ter sido gerado com cada um dos possíveis valores de θ. Desta forma é intuitivo pensar que a melhor estimativa para o parâmetro, baseada na amostra, é o valor do parâmetro que tem maior probabilidade de gerar o resultado visto na amostra, portanto o valor que maximiza a função de verossimilhança. Isto define o **estimador de máxima verossimilhança** $\hat{\theta}$. Também é intuitivo pensar que podemos definir uma "faixa" de valores que possuem uma probabilidade "não muito distante e aceitável" da máxima probabilidade de gerar o resultado visto na amostra. Tal faixa define um **estimador por intervalo** com valores inferior e superior $(\hat{\theta}_I, \hat{\theta}_S)$ que delimitam uma região no espaço paramétrico que possui valores de verossimilhança que não estejam abaixo de um percentual pré-dfinido do máximo possível valor da verossimilhança. Finalmente, pode-se verificar se

um determinado valor de interesse, tal como $\theta_0 = 0{,}5$ no exemplo considerado, é *compatível* com a amostra comparando-se sua verossimilhança com a máxima possivel. Este último caso permite definir um **teste de hipótese** de interesse para guiar uma tomada de decisão. Na Figura 1.2 redesenhamos o gráfico da função de verossimilhança agora com a escala vertical com valores relativos ao máximo valor e os elementos no gráfico ilustram os três objetivos centrais de inferência.

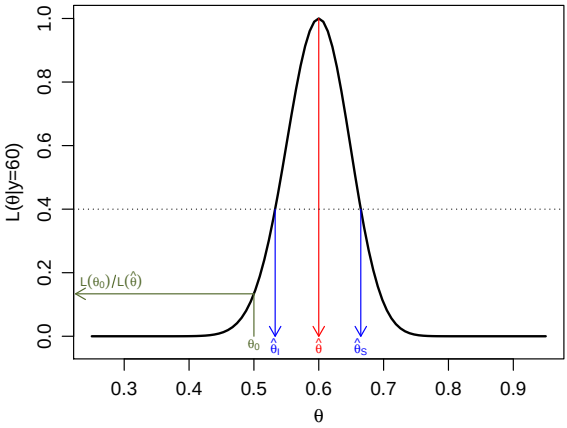

Figura 1.2: Visualização da estimativa de máxima verossimilhança $\hat{\theta}$, estimativa intervalar $(\hat{\theta}_I, \hat{\theta}_S)$ e valor correspondente a uma hipótese $\theta_0 = 0{,}5$ na função de verossimilhança relativa para um modelo $B(n = 100, \theta)$.

Neste texto será dada ênfase na inferência baseada na verossimilhança, que fornece **estimadores** e procedimentos com propriedades desejáveis para os parâmetros desconhecidos de um modelo probabilístico. A **função de verossimilhança** fornece todos os elementos necessários 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. A abordagem de inferência pela verossimilhança não é, entretanto, a solução de todo e qualquer problema, podendo tornar-se intratável analítica e/ou computacionalmente em certos modelos. Entretanto, os princípios permanecem válidos e diversos procedimentos estendem, aproximam ou substituem a verossimilhança quando necessário. Tais extensões estão fora do escopo deste texto que concentra-se na obtenção da verossimilhança com ênfase em procedimento numéricos. Nossa intenção é reforçar a intuição para aspectos fundamentais e básicos

de inferência.

Após analisar exemplos nos Capítulos iniciais exclusivamente sob o enfoque de verossimilhança, fazemos uma pequena incursão em inferência bayesiana, revisitando alguns exemplos e mostrando como implementar os modelos mais importantes em ferramentas de softwares conhecidos como o JAGS e o pacote **MCMCpack** do R. Esta pequena introdução a inferência bayesiana embasa a discussão posterior de dois recentes tópicos que expandem os paradigmas de inferência, o algoritmo *data clonning* para verossimilhança e e o INLA *Integrated Nested Laplace Approximation* para inferência bayesiana utilizando aproximações numéricas em substituição a procedimentos de inferência por simulação. O primeiro, faz uso de métodos de Monte Carlo via Cadeias de Markov (MCMC), tradicionais em inferência bayesiana para fazer inferência baseada em verossimilhança em modelos complexos. Deste algoritmo derivam-se alguns testes para diagnóstico de estimabilidade dos parâmetros envolvidos nos modelos, auxiliando a formulação de modelos cada vez mais complexos. O segundo, faz uso de aproximações determinísticas para distribuições a posteriori no contexto de inferência bayesiana, com consideráveis reduções em tempo computacional. Nestes tópicos adicionais, apresentamos as técnicas brevemente e apresentamos alguns exemplos usando pacotes do R, como o **dclone** e o **INLA**.

Capítulo 2

Verossimilhança

Neste Capítulo revisamos conceitos fundamentais de inferência estatística com ênfase aos relacionados à função de verossimilhança explorando computacionalmente os conceitos em uma série de exemplos.

Definição 2.1 (Função de Verossimilhança)**.** *Sejam dados $\underline{y}$ 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 $L(\underline{\theta},\underline{y}) \equiv f(\underline{Y},\underline{\theta})$.*

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) do modelo com a amostra observada medida pela probabilidade ou densidade conjunta para os valores observados..

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,

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

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:

$$L[\theta] = P_\theta[y_{1I} \leq y_1 \leq y_{1S}, y_{2I} \leq y_2 \leq y_{2S}, \ldots, y_{nI} \leq y_n \leq y_{nS}]. \qquad (2.1)$$

Esta definição é geral e requer a especificação da distribuição conjunta de $\underline{Y}$. Fazendo a suposição de observações independentes tem-se que:

$$L[\theta] = P_\theta[y_{1I} \leq y_1 \leq y_{1S}] \cdot P_\theta[y_{2I} \leq y_2 \leq y_{2S}], \ldots, P_\theta[y_{nI} \leq y_n \leq y_{nS}]. \tag{2.2}$$

Até este ponto a definição pode ser utilizada tanto para dados considerados pontuais quanto para 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, cada dado é medido em um intervalo $(y_i - \delta/2 \leq Y_i \leq y_i + \delta/2)$ e a verossimilhança 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 dos parâmetros temos a verossimilhança como produto das densidades individuais,

$$L[\theta] \approx \prod_{i=1}^{n} f(y_i, \underline{\theta}), \tag{2.3}$$

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

$$L[\theta] \approx f(\underline{y}, \underline{\theta}). \tag{2.4}$$

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 $\underline{Y}$.

Esta parte do texto concentra-se exclusivamente no uso da função de verossimilhança como base para inferência estatística, sejam na obtenção de estimativas pontuais, intervalares ou testes de hipótese. Começamos revisando conceitos de estimação e relações com a função de verossimilhança.

2.1 Estimação pontual

Seja $Y_1, Y_2, \ldots, Y_n$ variáveis aleatórias com forma conhecida da função probabilidade no caso de variáveis aleatórias discretas ou da função densidade de probabilidade para contínuas, denotadas em ambos os casos por $f(\underline{Y}, \underline{\theta})$. O vetor $\underline{\theta}$ denota os parâmetros desconhecidos (um único elemento de $\underline{\theta}$ será denotado por θ), a estimar através de uma amostra $y_1, y_2, \ldots, y_n$, 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 2.2 (Estatística)**.** *Uma **estatística** é uma variável aleatória $T = t(\underline{Y})$, onde a função $t(\cdot)$ não depende de θ.*

Definição 2.3 (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 2.4 (Distribuição amostral)**.** *A distribuição de probabilidade de T é chamada de **distribuição amostral** do estimador $t(\underline{Y})$.*

Definição 2.5 (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 2.6 (Eficiência relativa)**.** *A **eficiência relativa** entre dois estimadores T_1 e T_2 é a razão $er = \frac{V(T_1)}{V(T_2)}$ em que $V(\cdot)$ denota a variância.*

Definição 2.7 (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 2.8 (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 2.1, 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, $\hat{\underline{\theta}}$, 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 2.9. *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 2.10. *Se $\hat{\underline{\theta}}(\underline{y})$ é a estimativa de máxima verossimilhança, então $\hat{\underline{\theta}}(\underline{Y})$ é o estimador de máxima verossimilhança (EMV).*

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 2.11 (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 2.1 (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.

2.2 Intervalos de confiança

Definição 2.12 (Intervalo de confiança)**.** *Um intervalo de verossimilhança para θ é um intervalo da forma $\theta : L(\theta) \geq rL(\hat{\theta})$ ou equivalentemente, $\theta : D(\theta) \leq c^*$, com $D(\theta) = -2[l(\theta) - l(\hat{\theta})]$ 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 necessário para a construção dos intervalos ainda não temos elementos suficientes para 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 intervalo pelos valores que satisfazem $D(\theta) = -2[l(\theta) - l(\hat{\theta})] \leq -2\log(r)$. Esta é uma outra forma de considerar 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 $c^* = -2\log(r)$, é necessário encontrar as raízes da função de verossimilhança relativa ou da *deviance* que fornecem os limites do intervalo de confiança 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.

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

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

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

que define então intervalos de confiança da forma,

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

Isto corresponde a fazer uma aproximação quadrática da função *deviance*, que torna o intervalo fácil de ser obtido. Estendendo para o caso de múltiplos parâmetros, tem-se que uma região de confiança para $\underline{\theta}$ é dada pelo conjunto $\underline{\theta} \in \Theta : D(\underline{\theta}) \leq c^*$. 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.

2.3 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 2.13 (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 2.14 (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 2.15 (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 para um vetor de parâmetros da seguinte forma:

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

Assim $D(\underline{\theta})$ é não negativa uma vez que $I_O(\hat{\underline{\theta}})$ é uma matriz positiva definida. Uma vez definidas estas quantidades envolvidas, estamos aptos a enunciar Teorema a seguir.

Teorema 2.2 (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*

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

ou seja, a distribuição assintótica de $\hat{\underline{\theta}}$ 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 2.2. Assim,

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

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

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

Teorema 2.3 (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*

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

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 verossimilhança $\hat{\underline{\theta}}$ de $\underline{\theta}$ é assintoticamente não-viciado, isto é, $E(\hat{\underline{\theta}}) \to \underline{\theta}$.
- Assintoticamente $V(\hat{\underline{\theta}}) \to I_E^{-1}(\underline{\theta})$, 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 $J = I_E^{-1}(\underline{\theta})$, 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 $\hat{\theta}_i \pm z_{\frac{\alpha}{2}} J_{i,i}^{\frac{1}{2}}$. 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 utilizando diretamente a função *deviance* obtida com os dados. A segunda opção é em geral mais trabalhosa computacionalmente, uma vez que usualmente gera uma equação não linear que precisa ser resolvida numericamente. 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 é muito simples mas apresenta limitações. Restrições naturais do espaço paramétrico como, por exemplo, para parâmetros de variância e correlação não são respeitadas e podem resultar em limites absurdos, com limite(s) do intervalo fora do espaço paramétrico. Os intervalos serão sempre simétricos ao aproximar a verossimilhança por uma forma quadrática, o que normalmente não produz resultados adequados para parâmetros de variância e correlação. Em modelos com efeitos aleatórios há um interesse natural nos parâmetros de variância, precisão e correlação. Testar a significância de tais efeitos utilizando as variâncias associadas às estimativas que indexam o modelo pode produzir resultados imprecisos. Logo, esta abordagem é restrita em classes mais gerais de modelos estatísticos.

A segunda opção resulta em uma região conjunta para o caso de dois ou mais parâmetros, enquanto que pela aproximação é possível obter um intervalo marginal para cada parâmetro, porém baseado em uma aproximação quadrática da superfície de log-verossimilhança. Este tipo de representação é a mais desejável para inferência, porém não pode ser obtida diretamente apenas com o Teorema 2.3. Por exemplo, suponha que tem-se interesse em

um determinado componente do vetor de parâmetros, digamos θ_i. A partir da 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 2.3 para o caso em que a dimensão de $\underline{\theta}$ é maior que um, não temos um intervalo desta forma mas sim uma região o que apesar de mais informativa tem menor 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.

Temos um método simples via aproximação quadrática, porém que não funciona bem quando a superfície de log-verossimilhança é assimétrica. Por outro lado, o método da *deviance* não apresenta esta restrição mas fornece regiões de confiança conjuntas, e não *diretamente* limites $\hat{\theta}_L$ 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 e, 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, como $\hat{\phi} = g(\hat{\underline{\theta}})$, 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 2.4. *Considere obter um intervalo de confiança para $\phi = g(\underline{\theta})$ por invariância temos que $\hat{\phi} = g(\hat{\underline{\theta}})$ e a variância de $\hat{\phi}$ é dada por*

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

com

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

A partir do Teorema 2.4 é imediato o seguinte resultado.

Teorema 2.5. *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*

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

Pelo Teorema 2.5, 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 2.16 (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 2.16 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

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

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.

2.4 Testes de hipóteses

Definição 2.17. *Chamamos de hipótese estatística qualquer afirmação acerca da distribuição de probabilidade de uma ou mais variáveis aleatórias.*

Definição 2.18. *Chamamos de teste de uma hipótese estatística a função de decisão* $\chi \rightarrow \{a_0, a_1\}$, *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 não rejeição de H_0, vamos chamar de A_0 de região de não rejeição 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 não rejeitamos 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 2.19. *O poder do teste com região crítica* 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.

2.4.1 Teste da razão de verossimilhança

Definição 2.20. *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 2.3 à medida que $n \rightarrow \infty$

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

2.4.2 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.

2.4.3 Teste escore

Definição 2.21. *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,

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

ou seja, 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.

2.5 Exemplo - 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. A função de verossimilhança 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 constante 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:

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

Neste caso o estimador de máxima verossimilhança para λ pode ser en-

contrado analiticamente maximizando a função de log-verossimilhança.

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

Vamos obter o gráfico das funções associadas à verossimilhança para uma amostra simulada da distribuição de Poisson com parâmetro $\lambda = 10$.

```
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
```

A Figura 2.1, apresenta os gráficos dessas quatro formas de visualização da função de verossimilhança para os dados simulados. Utilizamos a função definida no código 2.1 que permite escolher a representação desejada da verossimilhança. As verossimilhanças relativa e *deviance* requerem que o valor da verossimilhança maximizada seja informado no argumento `maxlogL`, que é constante para uma determinada amostra. Deixamos este cálculo fora da função para evitar que esta quantidade constante seja recalculado nas sucessivas avaliações de função. Para facilitar o obtenção dos gráficos definimos a função na forma vetorizada utilizando `sapply()` para que a função possa receber um vetor de valores do parâmetro.

Código 2.1: Função com diferentes representações da verossimilhança para distribuição de Poisson.

```
veroPois <- function(par, dados, tipo, maxlogL){
   tipo = match.arg(tipo, choices=c("L","LR","logL","dev"))
   ll <- sapply(par, function(p) sum(dpois(dados, lambda=p,
                                           log=TRUE)))
   return(switch(tipo, "L" = exp(ll),
                       "LR" = exp(ll-maxlogL),
                       "logL" = ll,
                       "dev" = 2*(maxlogL-ll)))}
```

Os comandos a seguir mostram a obtenção da log-verossimilhança maximizada $l(\hat{\lambda})$ e a chamada para obter o gráfico da função *deviance* $D(\lambda)$. Para os demais gráficos basta alterar os valores do argumento `tipo`.

```
mll <- sum(dpois(y, lambda=mean(y), log=TRUE))
curve(veroPois(x, dados=y, tipo="dev", maxlogL=mll), 8, 11,
      ylab=expression(D(lambda)), xlab=expression(lambda))
```

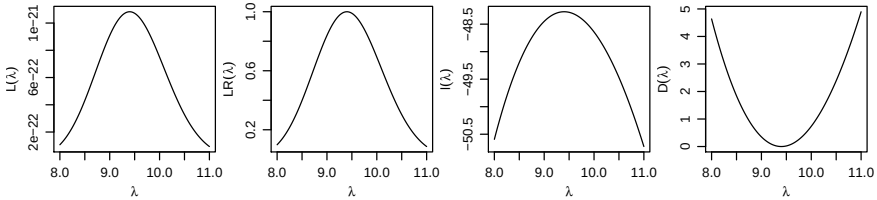

Figura 2.1: Diferentes formas de visualizar a função de verossimilhança - Distribuição Poisson.

Apesar das quatro formas serem equivalentes a forma usual para encontrar o estimador de máxima verossimilhança é a log-verossimilhança. De forma geral, cálculos analíticos com a função de verossimilhança $L(\lambda)$ podem ser mais trabalhosos enquanto que sua computação mais sensível a valores que podem gerar problemas numéricos, por exemplo excedendo a capacidade de representação de números. A verosimilhança relativa e *deviance* requerem o valor da função de verosimilhança avaliado na estimativa. Para a *deviance*, pela definição, a estimativa corresponde à raiz de função, ou seja, onde a função toca o eixo com valores do parâmetro.

Embora neste exemplo o EMV pode ser encontrado analiticamente, vamos ilustrar métodos numéricos comumente utilizados para encontrar EMV. Mas antes disto vamos redefinir a função de verossimilhança escrita agora como função da estatística suficiente calculada com os valores da amostra. Definimos $l(\lambda)$ como opção *default*. O argumento `amostra` deve receber uma lista com o tamanho e soma dos termos da amostra. Omitimos em $L(\lambda)$ e $l(\lambda)$ o termo que não depende do parâmetro. $LR(\lambda)$ e $D(\lambda)$ não se alteram pois termos se cancelam em seu cálculo.

Código 2.2: Refefinição função com diferentes representações da verossimilhança para distribuição de Poisson.

```
veroPois <- function(par, amostra, tipo="logL", maxlogL){
   tipo = match.arg(tipo, choices=c("L","LR","logL","dev"))
   ll <- with(amostra, -n*par + soma * log(par))
   return(switch(tipo, "L" = exp(ll),
                       "LR" = exp(ll-maxlogL),
                       "logL" = ll,
                       "dev" = 2*(maxlogL-ll)))}
```

Comandos equivalentes aos anteriores para obtenção do gráfico seriam como a seguir.

```
am <- list(n=length(y), soma=sum(y))
(emv <- mean(y))
```

```
[1] 9.4
```

```
mll <- veroPois(emv, amostra=am, tipo="logL")
curve(veroPois(x, amostra=am, tipo="dev", maxlogL=mll), 8, 11,
      ylab=expression(D(lambda)), xlab=expression(lambda))
```

Para ilustrar obtenção da estimativa do parâmetro por métodos numéricos vamos considerar as seguintes opções: i) solução de equação de estimação $U(\lambda = 0)$ por um método sem uso de gradientes (Brent) e por um método com uso de gradientes (Newton-Raphson); ii) maximização de função de verossimilhança.

Código 2.3: Função escore para Poisson.

```
UPois <- function(lambda, amostra){
  return(with(amostra, n - soma/lambda))
}
```

Para obter a estimativa utilizamos inicialmente a função `uniroot()` implementa um algoritmo para encontrar a raiz de uma equação.

```
uniroot(UPois, interval=range(y), amostra=am)$root
```

```
[1] 9.400002
```

O algoritmo talvez mais comumente utilizado é o de Newton-Raphson que, utilizando uma expansão em séries de Taylor de $U(\lambda)$, resolve a equação a seguir até que algum critério de convergência seja atingido.

$$\lambda^{r+1} = \lambda^r - \frac{U(\lambda)}{H(\lambda)}$$

Para implementar o algorítmo precisamos definir primeiro a função $H(\lambda) = U'(\lambda)$.

Código 2.4: Função $H(\lambda) = I_O(\lambda)$ para Poisson.

```
HPois <- function(lambda, amostra){
  return(amostra$soma/lambda^2)
}
```

Uma variante do método é utilizar $H(\lambda) = I_E(\lambda)$, conhecido como *Fisher scoring*. A estimativa é obtida por este algoritmo a partir de um valor inicial.

```
maxit <- 100; lambdaNR <- 5; iter <- 0; d <- 1
while(d > 1e-12 & iter <= maxit){
    lambdaNR.new <-
      lambdaNR - UPois(lambdaNR, am)/HPois(lambdaNR, am)
    d <- abs(lambdaNR - lambdaNR.new)
    lambdaNR <- lambdaNR.new ; iter <- iter + 1
}
c(lambdaNR, iter)
```

```
[1] 9.4 7.0
```

No exemplo a estimativa 9.4 foi obtida em 7 iterações. Os comandos acima podem ser encapsulados em uma função para facilitar o uso. Existem ainda funções no R que implementam esta algoritmo. Uma possível generalização é utilizar funções $U(\lambda)$ e $H(\lambda)$ obtidas numericamente para modelos em que não há expressões fechadas para estas funções. Isto nos remete a métodos numéricos para maximização de $l(\lambda)$. Para o caso de um único paramêtro utilizamos a função `optimize()` que utiliza o algoritmo de Brent e diversas outras funções são disponíveis no R e pacotes, sendo mais comum o uso de `optim()`.

```
unlist(optimize(veroPois, int=range(y), maximum=TRUE, amostra=am)[1:2])
```

```
  maximum  objective
9.399997 233.253422
```

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 2.2 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 2.2 sobrepomos o histograma das estimativas obtidas nas simulações com a gráfico da distribuição assintótica (normal).

Como é possível visualizar na Figura 2.2 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 comple-

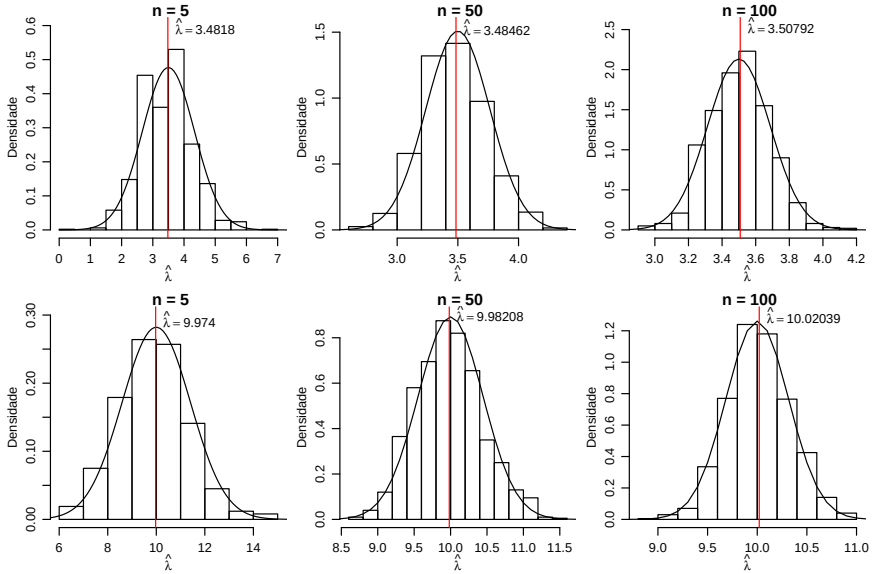

Figura 2.2: Distribuição assintótica estimador de máxima verossimilhança- Distribuição Poisson.

xos, como nos modelos lineares generalizados (MLG) estes resultados permanecem igualmente bons.

2.6 Exemplo - Intervalos de confiança

Como vimos na Seção 2.2 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 e dependem do valor do parâmetro e na prática é necessário usar a informação estimada pela amostra.

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

Para encontrar o intervalo de confiança vamos começar pela função *deviance*, lembrando que de acordo com o Teorema 2.3, 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:

$$\begin{aligned} D(\theta) &= 2[l(\hat{\theta}) - l(\theta)] \\ &= 2[n\log\hat{\theta} - \hat{\theta}n\overline{y} - (n\log\theta - \theta n\overline{y})] \\ &= 2n[\log\left(\hat{\theta}/\theta\right) + \overline{y}(\theta - \hat{\theta})] \leq c^* \end{aligned} \tag{2.5}$$

Os limites $(\hat{\theta}_L, \hat{\theta}_U)$ são encontrados resolvendo a equação em 2.5, 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 2.2.

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

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

Aqui 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 $(1-\alpha)$ 100% 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:

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

Tabela 2.1: 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

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

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

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:

$$\begin{aligned}
\frac{L(\theta)}{L(\hat{\theta})} &\geq r \\
\log\left[\frac{L(\theta)}{L(\hat{\theta})}\right] &\geq \log r \\
-2[l(\theta) - l(\hat{\theta})] &\geq 2 \cdot \log r \\
-3{,}84 &\geq 2 \cdot \log r \\
r &\approx 0{,}146
\end{aligned}$$

A Tabela 2.1 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 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)
```

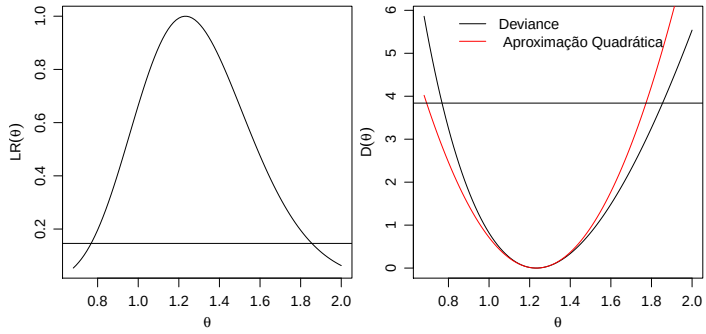

Figura 2.3: Verossimilhança relativa (esquerda) e função deviance exata e aproximada (direita) – Distribuição exponencial.

Como é possível ver na Figura 2.3 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 2.5: 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 2.3 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 2.4 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.

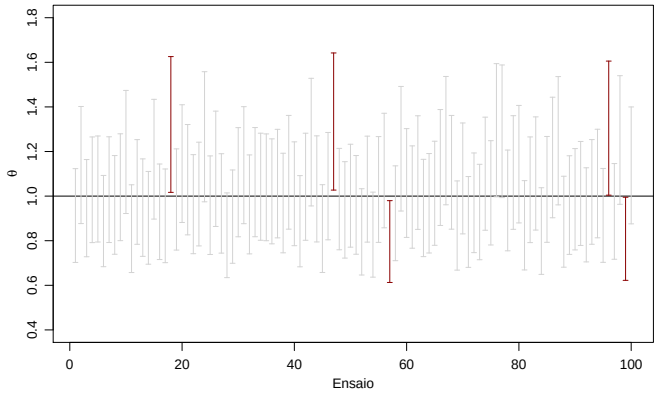

Figura 2.4: Interpretação frequentista de intervalos de confiança.

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 2.6: 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
```

2.7 Exemplo - 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 2.4, 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 2.4 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 $Y_i \sim P(\lambda)$. 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,

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

A função de log-verossimilhança,

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

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

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

que sendo resolvida fornece a estimativa de máxima verossimilhança

$$\hat{\lambda} = \frac{\sum_{i=1}^{n} y_i}{n} = \overline{y}.$$

Além disso, temos que a informação observada é dada por

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

Para obter informação esperada avaliamos a esperança da informação observada

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

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

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

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

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

A hipótese nula $H_0 : \lambda = \lambda_0$ será rejeitada se, e somente se, $-2\log\lambda(\underline{y}) \geq \chi^2_{1,\alpha}$. 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

$$T_w = \frac{\hat{\lambda} - \lambda_0}{\sqrt{V(\hat{\lambda})}} \sim N(0,1).$$

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

$$T_w = \frac{\overline{y} - \lambda_0}{\sqrt{\overline{y}/n}} \sim N(0,1).$$

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.

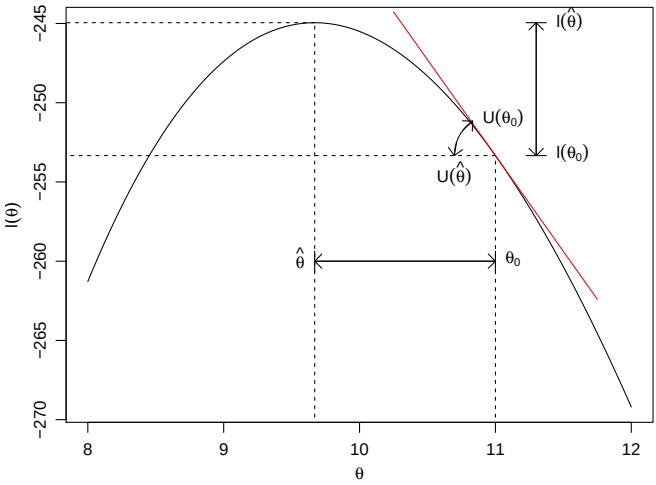

Figura 2.5: Diferentes formas de construir teste de hipótese baseado em verossimilhança.

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,

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

A Figura 2.5 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 2.7: 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 2.8: 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 2.9: 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)
dados <- rpois(100, lambda=10)
## Estatistica do TRV caso Poisson
Est <- function(H0, y){
  n <- length(y)
  EMV <- mean(y)
  lv <- 2*n*((H0 - EMV) + EMV*log(EMV/H0))
return(lv)}
## Procedendo com o TRV
trv(Est = Est, H0=8, alpha = 0.05, y=dados)
```

```
[1] "Rejeita H0"
[1] 32.660809  3.841459
```

```
## Teste de Wald
wald(H0=8, EMV = mean(dados), V.EMV = mean(dados)/length(dados),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 2.10: Função escore para a distribuição de Poisson.

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

Código 2.11: Informação esperada para a distribuição Poisson.

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

```
escore(H0 = 8, U = fc.escore, Ie = Ie, alpha=0.05, y=dados)
```

```
[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.

2.8 Exemplo - 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 $Y_i : i = 1,\ldots,n$ variáveis aleatórias independentes com função densidade de probabilidade dada por:

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

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,

$$L(\theta) = \prod_{i=1}^{n} 2\theta y_i \exp\{-\theta y_i^2\} = (2\theta)^n \prod_{i=1}^{n} y_i \exp\left[-\theta \sum_{i=1}^{n} y_i^2\right],$$

logo a função de log-verossimilhança,

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

Derivando em relação a θ chegamos a função escore,

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

Igualando a zero encontramos a estimativa de máxima verossimilhança,

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

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,

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

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

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

Neste caso em particular $I_O(\theta) = I_E(\theta)$ pois não dependem de $\underline{y}$, 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,

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

Isto pode ser resolvido usando algum método numérico conforme será explicado na sequência. Por agora, podemos resolver conforme no Exemplo 2.6, 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.

$$L(\mu) = \prod_{i=1}^{n} 2\mu^{-1} y_i \exp\left[-\frac{y_i^2}{\mu}\right] = (2\mu^{-1})^n \exp\left[-\frac{1}{\mu}\sum_{i=1}^{n} y_i^2\right] \prod_{i=1}^{n} y_i.$$

A log-verossimilhança é dada por,

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

Derivando em μ obtemos a função escore,

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

Igualando a zero chegamos a estimativa de máxima verossimilhança,

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

Para a informação observada tem-se

$$I_O(\mu) = -\frac{\partial^2 l(\mu)}{\partial \mu^2} = -\frac{\partial}{\partial \mu}\left[-\frac{n}{\mu} + \mu^{-2}\sum_{i=1}^{n} y_i^2\right]$$
$$= -n\mu^{-2} + 2\mu^{-3}\sum_{i=1}^{n} y_i^2.$$

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

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

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

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

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 que 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,

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

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

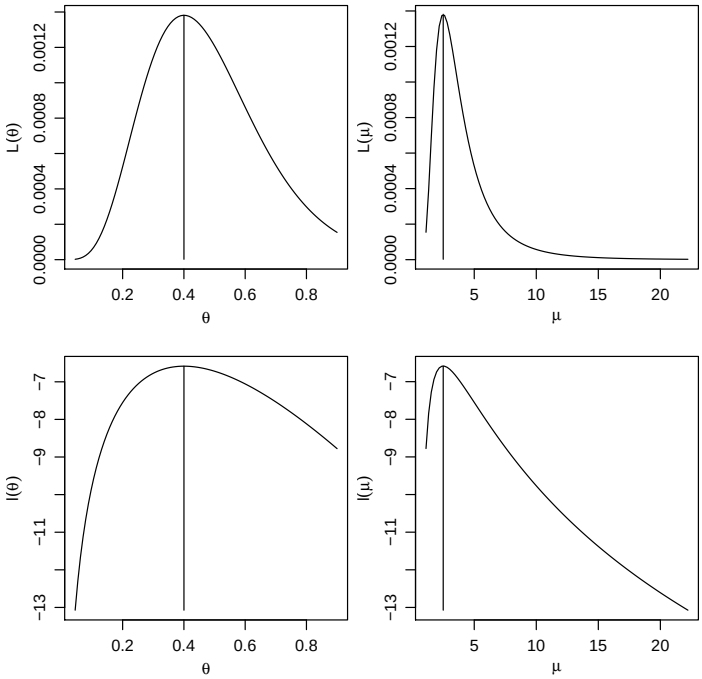

Figura 2.6: Verossimilhança e log-verossimilhança para θ e μ.

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

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

```
dados <- c(0.19,1.68,2.81,0.59,1.18)
theta.est <- length(dados)/sum(dados^2)
mu.est <- sum(dados^2)/length(dados)
c(theta.est, mu.est)
```

```
[1] 0.4001569 2.4990200
```

Entretanto, na prática não é conveniente nem necessário reescrever a função de verossimilhança para cada reparametrização de interesse. Pela propriedade de invariância pode-se obter a função de verossimilhança de μ partindo de θ e vice-versa. Os gráficos da função de verossimilhança e log-verossimilhança nas duas parametrizações são apresentados na Figura 2.6.

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 2.7, apresenta as funções *deviance* exatas e aproximadas em cada parametrização. Para isto, precisamos criar cada uma das funções, para depois poder avalia-las.

Código 2.12: Funções *deviance* para o parâmetro θ e μ.

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

Como é possível ver na Figura 2.7 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 assintóticas do estimador de máxima verossimilhança, sabemos que $\hat{\theta} \sim N(\theta, I_E^{-1}(\theta))$ e que podemos substituir $I_E^{-1}(\theta)$ por $I_E(\hat{\theta})$, $I_O(\theta)$ ou $I_O(\hat{\theta})$. Como visto anteriormente neste exemplo 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 $I_E(\hat{\mu}) = \frac{n}{\hat{\mu}^2}$ enquanto que a informação observada é dada por $I_O(\hat{\mu}) = -\frac{n}{\hat{\mu}^2} + \frac{2\sum_{i=1}^{N} y_i^2}{\hat{\mu}^3}$, 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 2.13: 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(dados)
#theta.est <- n/sum(dados^2)
ic.theta <- ic.assintotico(EMV = theta.est,
            Io = (n / theta.est^2), alpha=0.05)
#mu.est <- sum(dados^2)/n
ic.mu.obs <- ic.assintotico(EMV = mu.est,
             Io = -n/mu.est^2 + (2*sum(dados^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 2.14: 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, y=dados, desloca=qchisq(0.95, df=1))
```

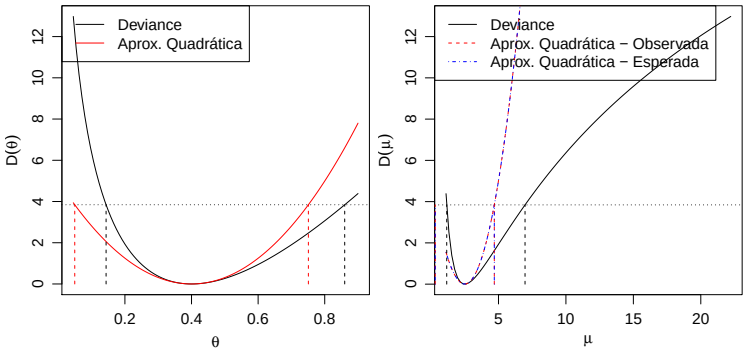

Figura 2.7: Deviances exatas e aproximadas para cada parametrização. Linhas verticais indicam os respectivos intervalos de confiança.

```
ic.dev.mu <- ic.deviance(dev=dev.mu, intervalo=c(0,100),
            mu.hat=mu.est, y=dados, 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 2.7. 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 é especialmente relevante se intervalos aproximados (assintóticos) vão ser utilizados.

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

```
                [,1]      [,2]
ic.theta     0.04941035 0.7509034
ic.dev.theta 0.14349722 0.8600242
```

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

```
              [,1]     [,2]
ic.dev.mu 1.1627258 6.968803
ic.mu.obs 0.3085726 4.689467
ic.mu.esp 0.3085726 4.689467
```

Em situações práticas 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 2.4. No exemplo, obtemos o intervalo aproximado para θ usando por exemplo a informação esperada e desejamos o intervalo para μ. Usando o Teorema 2.4 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 2.4 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

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

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,]   0.3085726 0.3085726
[2,]   4.6894674 4.6894674
```

Os intervalos são idênticos porém com um esforço de obtenção muito menor do que redefinir a verossimilhança. O Teorema 2.4 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.

2.9 Exemplo - Modelo AR1

Nos exemplos anteriores consideramos observações independentes e a verossimilhança é portanto dada pelo produto das densidades. Vamos considerar agora um modelo simples, porém com observações correlacionadas. Tomamos o caso de um modelo AR1 (autoregressivo de ordem 1) para dados gaussianos que é um modelo básico em séries temporais. Um texto de referência na área é Toloi (2004). Vamos considerar uma versão simplificada deste modelo assumindo que o processo possui média zero e variância unitária. Desta forma, o modelo é definido por:

$$\begin{aligned} y_{y+1} &= \rho y_t + e_{t+1} \\ e_t &\sim N(0, \sigma^2 = 1) \\ \text{com } &|\rho| < 1 \end{aligned} \tag{2.6}$$

Decorre deste modelo que

$$\begin{aligned} [Y_i|Y_{i-1}] &\sim N(\rho\, y[i-1], 1) \;\; ; \;\; i = 2, \ldots, n\,, \\ [Y_1] &\sim N(0, 1/(1-\rho^2)) \;\; ; \;\; i = 2, \ldots, n \end{aligned}$$

Uma simulação da série fixando os valores necessários é feita nos comandos a seguir. Definimoms o valor do parâmetro $\rho = 0{,}7$ e o número de observações $n = 100$. A série simulada é mostrada na Figura 2.8.

```
set.seed(1242)
rho <- 0.7 ; n <- 100
y <- numeric(n) ## cria vetor com elementos nulos
y[1] <- rnorm(1, m=0, sd=sqrt(1/(1-rho^2)))
for(i in 2:n)
  y[i] <- rho * y[i-1] + rnorm(1)
```

A expressão da função de verossimilhança dada por 2.4 é equivalente à distribuição conjunta para as n observações, uma distribuição gaussiana multivariada neste caso. Vamos explorar aqui algumas formas alternativas de escrever esta função.

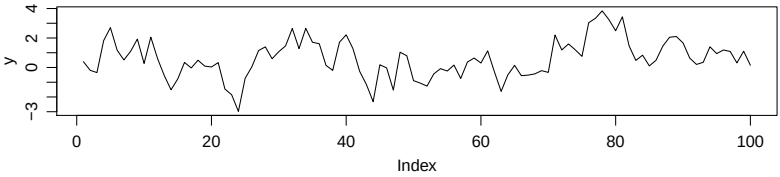

Figura 2.8: Valores da série simulada

Podemos escrever a distribuição conjunta com a expressão da distribuição multivariada gaussiana de $[Y_1, Y_2, \ldots, Y_n]$ induzida pelo modelo ou pelo produto de distribuições condicionais univariadas. No caso do modelo AR1 as distribuições univariadas dependem apenas da observação anterior e temos então que a expressão da verossimilhança é:

$$L[\rho] \equiv [Y_1, Y_2, \ldots, Y_n] = [Y_1][Y_2|Y_1] \ldots [Y_n|Y_{n-1}] = [Y_1] \prod_{i=2}^{n} [Y_i|Y_{i-1}]$$

Uma verosimilhança aproximada é obtida ignorando-se a contribuição da primeira observação, ou seja pela distribuição condicional à primeira observação.

$$L_A[\rho] = \prod_{i=2}^{n} [Y_i|Y_{i-1}].$$

É possível encontrar o estimador do parâmetro em forma fechada com $L_A[\rho]$ mas métodos numéricos são necessários para maximizar $L[\rho]$. Entretanto, no que se segue vamos sempre utilizar métodos numéricos uma vez que o foco aqui não é discutir este modelo em particular mas sim ilustrar implementações que possam ser adaptadas para modelos que possuam estrutura similar.

Começamos definindo no código 2.15 a função de verossimilhança aproximada. Por conveniência definimos também uma versão vetorizada da função que é útil para processar diversos valores do parâmetro de uma só vez como, por exemplo, quando fazemos gráficos da função.

Código 2.15: Função de log-verossimilhança (aproximada) para modelo AR1 com $\mu = 0$ e $\sigma = 1$.

```
llAR1.a <- function(par, sigma=1, dados){
    n <- length(dados)
    sum(dnorm(dados[2:n], mean=par*dados[1:(n-1)], sd=sigma,
              log=TRUE))
}
llAR1.a.V <- Vectorize(llAR1.a, "par")
```

Com o comando a seguir obtém-se, por algorítimos numéricos, a estimativa do parâmetro ρ que maximiza a função de verossimilhança aproximada.

```
unlist(rho.est.a <- optimize(llAR1.a, int=c(0, 1), dados=y, maximum=TRUE))
    maximum   objective
  0.7270684 -137.6080152
```

```
(rho.a <- rho.est.a$maximum)
```

```
[1] 0.7270684
```

No código 2.16 a seguir definimos a verossimilhança completa que inclui a distribuição da primeira observação, e a respectiva forma vetorizada da função.

$$[Y_1] \sim N(0, 1/(1-\rho^2)).$$

Código 2.16: Função de verossimilhança (completa) para modelo AR1 com $\mu = 0$ e $\sigma = 1$.

```
llAR1 <- function(par, sigma = 1, dados){
  n <- length(dados)
  dnorm(dados[1], mean=0, sd=sigma*sqrt(1/(1-par^2)), log=TRUE) +
    sum(dnorm(dados[2:n],mean=par*dados[1:(n-1)],sd=sigma,log=TRUE))
}
llAR1.V <- Vectorize(llAR1, "par")
```

Com esta função obtemos uma estimativa que, neste caso, é bem próxima à obtida com a verossimilhança aproximada.

```
unlist(rho.est <- optimize(llAR1, c(0,1), dados=y, maximum=TRUE))
   maximum   objective
  0.720074 -138.933886
```

```
(rho.emv <- rho.est$maximum)
```

```
[1] 0.720074
```

A seguir vamos traçar gráficos das funções de verossimilhança. Inicialmente vamos definir uma função deviance que é genérica no sentido de que pode ser calculada uma dada verossimilhança.

Código 2.17: Função deviance.

```
devfun <- function(par, llfun, est, ...)
     2*(llfun(est, ...) - llfun(par, ...))
```

Os gráficos das funções de verossimilhança e deviance aproximadas e completas para os dados simulados são mostrados na Figura 2.9. Para a primeira consideramos valores em todo o espaço paramétrico enquanto que para segunda tomamos apenas valores ao redor da estimativa de máxima

verossimilhança (completa). O intervalo de confiança foi definido aqui pela a faixa de valores para ρ cuja verossimilhança seja de ao menos 10% da verosimilhança maximizada ($r = 0{,}10$). Para isto encontramos o valor de corte correspondente na função deviance $cD = -2\log(r) = 4.61$ e usamos a função `uniroot.all()` do pacote `rootSolve` para encontrar os limites do intervalo.

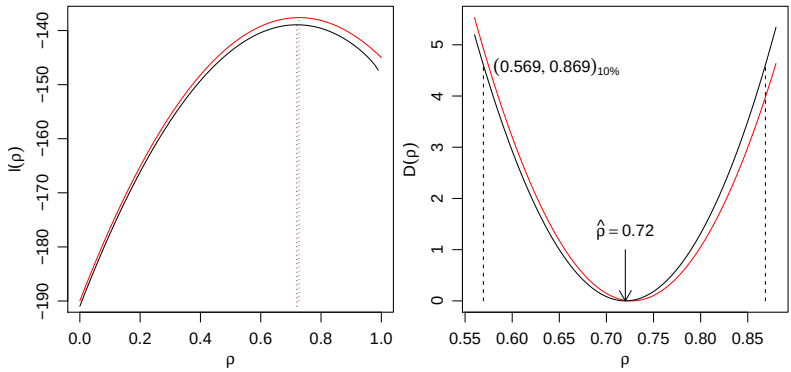

Figura 2.9: Função de verossimilhança (esquerda) e *deviance* (direita) aproximada (vermelha) e completa (preta) com estimativas pontual e intervalar do parâmetro ρ para os dados simulados do modelo AR1.

```
require(rootSolve)
ICdev <- function(par, devfun, cD, ...) devfun(par, ...) - cD
(IC.rel10 <- uniroot.all(ICdev, c(0,1), devfun=devfun, cD=-2*log(0.1),
                         llfun=llAR1.V, est=rho.emv, dados=y))
```

```
[1] 0.5694386 0.8687540
```

Em implementações que visam eficiência a função `llAR1()` pode ser reescrita evitando o uso de `dnorm()` e utilizando a expressões das densidades escritas em função de estatísticas suficientes. Isto reduz o tempo computacional em procedimentos interativos e/ou que avaliam a função muitas vezes.

Passamos agora a outra forma de escrever a verossimilhança utilizando a expressão da densidade conjunta (multivariada).

$$[Y_1, \ldots, Y_N] \sim N(0, \Sigma), \tag{2.7}$$

em que os elementos de Σ são $\Sigma_{ij} = \rho^{|i-j|}(1/(1-\rho^2))$. Os elementos da matriz são portanto função da distância $|i-j|$ entre as observações. No comando a seguir ilustramos como montar esta matriz calculando as distâncias entre pares de pontos e depois calculando os valores para $\rho = 0{,}70$.

```
n <- length(y)
{S <- diag(n) ; S <- abs(row(S)-col(S))}
S <- 0.7^S * (1/(1-0.7^2))
```

A expressão da (log)verossimilhança é obtida pela a densidade da gaussiana multivariada, sendo então:

$$l(\underline{\theta}) = l(\sigma, \rho) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma|) - \exp\{\frac{1}{2}y\prime\Sigma^{-1}y\}. \qquad (2.8)$$

Os três comandos a seguir mostram diferentes formas de avaliar esta densidade que produzem os mesmos resultados porém com tempos de execução distintos. Para ilustrar comparamos os tempos de execução de 100 avaliações destas funções. Os valores dos tempos não são relevantes e podem variar de um computador para outro. O relevante aqui é a comparação dos tempos, por exemplo tomando suas razões.

```
system.time(replicate(100, mvtnorm::dmvnorm(y,rep(0,n),S,log=T)))
```

```
   user  system elapsed
  0.136   0.116   0.092
```

```
system.time(replicate(100, (-n/2) * log(2*pi) -
    determinant(S,log=T)$mod/2 - 0.5*mahalanobis(y,center=0,cov=S)))
```

```
   user  system elapsed
  3.200   1.308   1.579
```

```
system.time(replicate(100, {Schol <- chol(S);
             (-n/2) * log(2*pi) - sum(log(diag(Schol))) -
             0.5*crossprod(backsolve(Schol, y, transpose=T))}))
```

```
   user  system elapsed
  0.044   0.080   0.041
```

O custo computacional é determinado pelas operações que envolvem a matriz de covariância. A primeiro utiliza a implementação do pacote **mvtnorm**. A segunda forma é a mais lenta pois acaba fazendo contas redundantes na cálculo do determinante e a forma quadrática $y'\Sigma^{-1}y$. A terceira forma é a mais eficiente pois tem o custo associado ao cálculo da decomposição de Choleski de Σ que, uma vez calculado, é usado para calcular de forma computacionalmente barata o tanto determinante quanto a forma quadrática. Estas diferenças podem ser muito relevantes em modelos que possuem alguma estrutura de covariância quando avalia-se a função várias vezes, como algoritmos de maximização ou de inferência por simulação. As diferenças serão maiores com o aumento do número de dados.

Mas, para o caso considerado aqui, ganhos adicionais de tempo computacional ainda podem ser obtidos. Em certos casos, resultados analíticos podem ser usados ao escrever as funções. No modelo AR1 considerado aqui a matriz Σ tem inversa de forma conhecida. e o código pode ser escrito de forma mais eficiente evitando inversão de matriz (ou solução de

sistema). Os elementos de Σ^{-1} são:

$$\begin{cases} \Sigma^{-1}_{i,i} = 1 & \text{para } i = 1 \text{ e } i = n \\ \Sigma^{-1}_{i,i} = 1 + \rho^2 & \text{para } 1 < i < n \\ \Sigma^{-1}_{i,j} = -\rho & \text{para } |i - j| = 1 \\ \Sigma^{-1}_{i,j} = 0 & \text{para } |i - j| > 1 \end{cases}$$

A matriz para os dados simulados poderia ser montada da forma como mostrado a seguir onde exibimos as cinco primeiras linhas e colunas.

```
iS <- diag(1+0.7^2, n)
diag(iS)[1] <- diag(iS)[n] <- 1
iS[abs(row(iS)-col(iS))==1] <- -0.7
iS[1:5, 1:5]
```

```
     [,1]  [,2]  [,3]  [,4]  [,5]
[1,]  1.0 -0.70  0.00  0.00  0.00
[2,] -0.7  1.49 -0.70  0.00  0.00
[3,]  0.0 -0.70  1.49 -0.70  0.00
[4,]  0.0  0.00 -0.70  1.49 -0.70
[5,]  0.0  0.00  0.00 -0.70  1.49
```

Desta forma o código pode ser escrito de forma mais eficiente evitando inversão de matriz (ou solução de sistema) no cálculo da forma quadrática. Além disto, o determinante de Σ possui expressão conhecida $\det(\Sigma) = 1/(1-\rho^2)$. Com estes resultados o cálculo da verossimilhança pode ser ainda substancialmente acelerados em comparação com os códigos anteriores.

Usando tais resultados pode-se obter tempos cumputacionais ainda mais rápidos que os anteriores.

```
system.time(replicate(100, {iSchol <- chol(iS)
(-n/2) * log(2*pi) + sum(log(diag(iSchol))) -
        0.5*drop(crossprod(iSchol %*% y))}))
```

```
   user  system elapsed
  0.032   0.020   0.018
```

```
system.time(replicate(100, 0.5*(-n*log(2*pi) + log(1-0.7^2) -
     mahalanobis(y, center=0, cov=iS, inverted=TRUE))))
```

```
   user  system elapsed
  0.028   0.008   0.013
```

```
     system.time(replicate(100, 0.5*(-n*log(2*pi) + log(1-0.7^2) -
        drop(crossprod(y, iS %*% y)))))
```

```
   user  system elapsed
  0.008   0.012   0.007
```

Finalmente vamos comparar com o tempo para o cálculo utilizando a forma fatorada da verossimilhança, com produto de distribuições univariadas.

```
system.time(replicate(100, llAR1(0.7, dados=y)))
```

```
   user  system elapsed
  0.004   0.000   0.002
```

E ainda há outras melhorias possíveis! A matriz inversa Σ^{-1} é esparsa (muitos elementos iguais a zero) e algorítmos e funções específicas como os do pacote **Matrix** podem ser utilizados não só para eficiência mas também para reduzir o uso de memória para armazenar tais matrizes. Deixamos tal implementação como sugentão ao leitor.

No código 2.18 implementamos a função de verossimilhança que é depois maximizada para obter a estimativa do parâmetro ρ.

Código 2.18: Função de verossimilhança escrita como densidade multivariada para modelo AR1 com $\mu = 0$ e $\sigma = 1$.

```
llAR1mv <- function(par, sigma = 1, dados){
    n <- length(dados)
    iS <- diag(1+par^2, n)
    diag(iS)[1] <- diag(iS)[n] <- 1
    iS[abs(row(iS)-col(iS))==1] <- -par
    return(0.5*(-n*log(2*pi) - 2*n*log(sigma) + log(1-par^2) -
                drop(crossprod(dados, iS %*% dados))/sigma^2))
}
```

```
unlist(optimize(llAR1mv, c(0,1), dados=y, maximum=TRUE))
```

```
   maximum  objective
0.720074 -138.933886
```

Vamos agora não mais fixar o valor para a variância σ^2 e considerá-la um parâmetro também a ser estimado. Neste caso os elementos da matrix de covariância em 2.7 são $\Sigma_{ij} = \rho^{|i-j|}(\sigma^2/(1-\rho^2))$. A função de log-verossimilhança $l(\underline{\theta}) = l(\sigma, \rho)$ fica como a seguir. A implementação supõe que o primeiro argumento da função é um vetor de comprimento dois com os valores para σ e ρ , nesta ordem.

Nesta parametrização dos parâmetros do modelo AR1 temos que o espaço paramétrico é restrito $\sigma > 0$ e $|\rho| < 1$. Uma possível reparametrização para qual o espaço paramétrico é o $\Re^2$ é adotar $\tau = \log(\sigma)$ e a transformação de Fisher $\varphi = \frac{1}{2}\log\left(\frac{1+\rho}{1-\rho}\right)$. Esta opção é implementada pelo argumento `repar` na função de verossimilhança definida em 2.19. Desta forma as três chamadas a seguir produzem o mesmo valor.

```
c(llAR1mv(0.7, dados=y), llAR1mv2(c(1, 0.7), dados=y),
  llAR1mv2(c(log(1), 0.5*log((1+0.7)/(1-0.7))), dados=y, repar=TRUE))
```

```
[1] -138.975 -138.975 -138.975
```

O código para obter as estimativas dos parâmetros por maximização numérica para os casos sem e com reparametrização é dado a seguir. No primeiro caso é necessário utilizar o método `L-BFGS-B` para poder delimitar o espaço paramétrico com os argumentos `lower` e `upper`. No segundo isto

Código 2.19: Função de log-verossimilhança $l(\sigma, \rho)$ para modelo AR1 com $\mu = 0$.

```
llAR1mv2 <- function(par, dados, repar=FALSE){
  ## par: vetor com valores de (sigma, rho), nesta ordem
  n <- length(dados)
  if(repar){
    sigma <- exp(par[1])
    rho <- (exp(2*par[2])-1)/(exp(2*par[2])+1)
  }
  else{ sigma <- par[1]; rho <- par[2]}
  iS <- diag(1+rho^2, n)
  diag(iS)[1] <- diag(iS)[n] <- 1
  iS[abs(row(iS)-col(iS))==1] <- -rho
  return(0.5*(-n*log(2*pi) - 2*n*log(sigma) + log(1-rho^2) -
              drop(crossprod(dados, iS %*% dados))/sigma^2))
}
```

não é necessário uma ver que o espaço paramétrico para o modelo reparametrizado é irrestrito. Pelo princípio da invariância tem-se que os valores maximizados das verossimilhanças são iguais. Eventuais diferenças, se houverem devem ser pequenas e devidas a "erros" numéricos. Mostramos ainda como os valores das estimativas no segundo caso, quando transformados de volta, produzem os mesmo valores primeiro.

```
opar <- optim(c(1, 0.7), fn=llAR1mv2, dados=y, method="L-BFGS-B",
              lower=c(0, -1), upper = c(Inf, 1),
              control=list(fnscale=-1), hessian=TRUE)
unlist(opar[1:2])
```

```
      par1       par2      value
 0.9661656  0.7205617 -138.8181162
```

```
rpar <- optim(c(log(1), 0.5* log((1+0.7)/(1-0.7))), fn=llAR1mv2,
         dados=y, repar=TRUE, control=list(fnscale=-1), hessian=TRUE)
unlist(rpar[1:2])
```

```
       par1        par2       value
-0.03448743  0.90889025 -138.81811683
```

```
c(exp(rpar[[1]][1]), (exp(2*rpar[[1]][2])-1)/(exp(2*rpar[[1]][2])+1))
```

```
[1] 0.9661005 0.7205992
```

A Figura 2.10 mostra as superfícies de verossimilhança para ambas parametrizações. Neste caso, a parametrização original produziu contornos mais próximos de um comportamento quadrático. Outro fato relevante revelado na figura é a quase ortogonalidade entre os parâmetros que pode ser observada em ambos os casos. Isto pode ser verificado numericamente na matriz de covariância dos estimadores obtida pela inversa da matriz de observação. Temos aqui que os elementos fora da diagonal possuem valores pequenos em comparação com os da diagonal, ou seja a matriz é proxima de uma matriz diagonal.

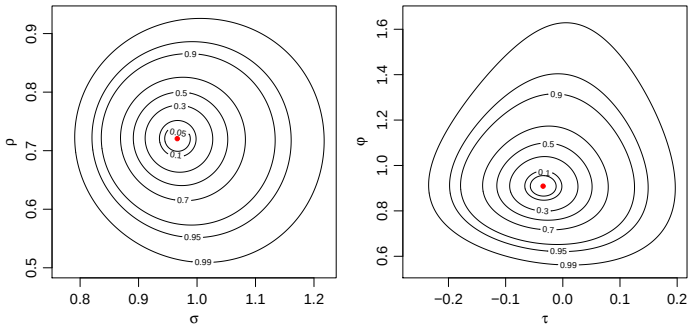

Figura 2.10: Superfícies de deviance para o modelo AR1 para parâmetros originais (esquerda) e transformados (direita).

```
 -solve(opar$hessian)
             [,1]          [,2]
[1,]  4.668280e-03 -6.622913e-05
[2,] -6.622913e-05  4.574118e-03
 -solve(rpar$hessian)
             [,1]          [,2]
[1,]  0.0050003310 -0.0001418373
[2,] -0.0001418373  0.0197914827
```

Vamos focar agora na inferência sobre o parâmetro ρ neste modelo de dois parâmetros, ou seja, o parâmetro σ^2 é considerado *nuisance*. No modelo definido em 2.7 e matriz de covariâncias pode ser reescrito como $\Sigma = \sigma^2 R_\rho$ destacando que a matriz de correlação R_ρ tem seus termos dependendo apenas do parâmetro ρ. A função de verrosimilhança 2.8 fica equivalente a

$$l(\underline{\theta}) = l(\sigma, \rho) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \frac{1}{2}|R_\rho| - \exp\{\frac{1}{2\sigma^2} y' R_\rho^{-1} y\}. \tag{2.9}$$

Tomando-se a derivada em relação à σ^2 e igualando à zero obtém-se o estimador deste parâmetro em relação à ρ,

$$\hat{\sigma}^2_\rho = \frac{y' R_\rho^{-1} y}{n}.$$

A verossimilhança concentrada ou *perfilhada* de ρ é então obtida substituindo-se σ^2 por $\hat{\sigma}^2_\rho$ em 2.9 e temos então:

$$pl(\rho) = -\frac{n}{2}\Big(\log(2\pi) + \log(\sigma^2) + 1\Big) - \frac{1}{2}\log(|R_\rho|).$$

Esta função é implementada em 2.20 e retorna dois valores: o da verossilhança $l(\rho, \sigma^2_\rho)$ e de σ^2_ρ, a estimativa de σ^2_ρ ao valor de ρ.

A implementação da função de verossimilhança perfilhada é simples. Tomamos a verossilhança que tem apenas ρ como parâmetro e adicionamos a opção para que se σ não for fornecido seja calculado para o valor corrente de ρ Se o valor de σ for fonecido é tomado como constante e a verossimilhança condicional à este valor é calculada.

Código 2.20: Função para cálculo da verossimilhança perfilhada $l(\rho|\sigma_\rho)$ e condicional $l(\rho|\sigma)$ para o modelo AR1 com $\mu = 0$.

```
llAR1.rho <- function(rho, sigma, dados){
    n <- length(dados)
    iS <- diag(1+rho^2, n)
    diag(iS)[1] <- diag(iS)[n] <- 1
    iS[abs(row(iS)-col(iS))==1] <- -rho
    if(missing(sigma))
        sigma2 <- drop(crossprod(dados, iS %*% dados)/n)
    else sigma2 <- sigma^2
    return(0.5*(-n*log(2*pi) - n*log(sigma2) + log(1-rho^2) -
                drop(crossprod(dados, iS %*% dados))/sigma2))
}
```

O gráfico da deviance perfilhada é visualizado à direita na Figura 2.11. Também é traçada a verossimilhança condicional na qual o valor de σ^2 é fixado em sua estimativa de máxima verossimilhança

$$\hat{\sigma}^2 = \hat{\sigma}^2_{\hat{\rho}} = \frac{y' \hat{R}^{-1}_{\hat{\rho}} y}{n}.$$

As funções diferem considerando uma maior região do espaço paramétrico conforme mostrado no gráfico do centro. Entretanto, na região do espaço paramétrico relevante para inferências no entorno do ponto de máximo as funções são quase indistinguíveis. Isto é mais um reflexo da quase ortogonalidade entre os parâmetros ao redor do máximo da função. No gráfico da deviance mostrada à esquerda e linhas indicam os cortes para obtenção das perfilhadas e condicionais.

Para finalizar consideramos o modelo AR1 mais geral, que inclui um termo para descrever a média do processo. Uma forma de escrever tal modelo é:

$$\begin{aligned} y_{y+1} &= \alpha + \rho y_t + e_{t+1} \\ e_{t+1} &\sim \mathrm{N}(0, \sigma^2) \end{aligned} \tag{2.10}$$

A média do processo é dada por:

$$\mathrm{E}[y_t] = \mu = \frac{\alpha}{1-\rho},$$

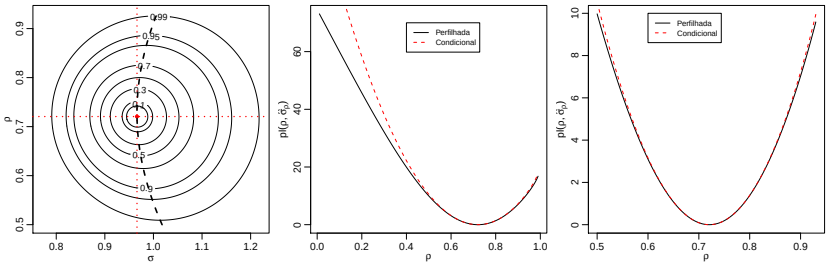

Figura 2.11: Superfície de deviance (esquerda) com indicações dos cortes para obtenção das perfilhadas e condicionais. Deviances perfilhadas e condicionais para o parâmetro ρ do modelo AR1 com $\mu = 0$.

e o modelo acima pode ser então reescrito na forma:

$$y_{y+1} = (1-\rho) + \rho y_t + e_{t+1}. \tag{2.11}$$

Para a expressão da verossimilhança, simplesmente acrescenta-se o termo de média a 2.8 que fica:

$$l(\underline{\theta}) = l(\sigma, \rho) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma|) - \exp\{\frac{1}{2}(y-\mu)'\Sigma^{-1})y-\mu)\}. \tag{2.12}$$

O código retornando o negativo da função de verossimilhança é implementado em 2.22 e a seguir são mostradas as estimativas para os dados aqui considerados.

Código 2.21: Função de verossimilhança para o modelo AR1.

```
llAR1mv3 <- function(par, dados){
    ## par: vetor com valores de (mu, sigma, rho), nesta ordem
    n <- length(dados)
    mu <- par[1]; sigma <- par[2]; rho <- par[3]
    iS <- diag(1+rho^2, n)
    diag(iS)[1] <- diag(iS)[n] <- 1
    iS[abs(row(iS)-col(iS))==1] <- -rho
    ymu <- dados - (1-rho)*mu
    return(-0.5*(-n*log(2*pi) - 2*n*log(sigma) + log(1-rho^2) -
                 drop(crossprod(ymu, iS %*% ymu))/sigma^2))
}
```

```
par3 <- optim(c(0,1,0.5), llAR1mv3, dados=y)
unlist(par3[1:2])
```

```
      par1       par2       par3      value
 1.7118165  0.9499772  0.6633920 137.0519734
```

Existem pacotes e funções específicas para ajustar modelos de séries temporais no R, descritos na *Time Series Task View*[1] As funções `ar()` e `arima()` estão entre as disponíveis e produzem resultados como mostrado a seguir, que diferem dos anteriores pois incluem a média na estimação o modelo com três parâmetros (μ, σ, ρ).

```
 (fit.ar <- ar(y, order.max=1, method="mle"))
```

```
Call:
ar(x = y, order.max = 1, method = "mle")

Coefficients:
     1
0.6635

Order selected 1  sigma^2 estimated as  0.9025
```

```
 with(fit.ar, x.mean)
```

```
[1] 0.57605
```

```
 (fit.arima <- arima(y, order=c(1,0,0), method="ML"))
```

```
Call:
arima(x = y, order = c(1, 0, 0), method = "ML")

Coefficients:
         ar1  intercept
      0.6635     0.5761
s.e.  0.0734     0.2769

sigma^2 estimated as 0.9025:  log likelihood = -137.05,  aic = 280.1
```

Note que enquanto nosso código retorna as estimativas de (μ, σ, ρ) na parametrização definida em 2.11 as funções `ar()` e `arima()` retornam estimativas de (α, σ^2, ρ) definida em 2.10.

$$0.5761 = \hat{\alpha} = (1 - \hat{\rho})\hat{\mu} = (1 - 0.6634) \cdot 1.7118 = 0.5762$$
$$0.9025 = \hat{\sigma}^2 = 0.95^2 = 0.9025$$

Aproveitamos esta última função para ilustrar e funcionalidade do pacote **bbmle** e de função genérica de ajuste de modelos `mle2()`. A função recebe o negativo da log-verossimilhança do modelo desejado, tal como em 2.22 neste exemplo. Internamente a função `optim()` é utilizada por *default* na otimização e outras podem ser selecionadas. Entretando a função "envoltório" `mle2()` prepara os resultados de forma que várias explorações do ajuste tais como intervalos de confiança e verossimilhnaças perfilhadas podem ser facilmente obtidas, sem a necessidade de programações adicionais.

O ajuste produz os mesmos resultados obtidos anteriormente por outros métodos.

[1] *http://cran.r-project.org/web/views/TimeSeries.html*

Código 2.22: Função de verossimilhança para o modelo AR1 modeficada para uso com `bbmle::mle2()`.

```
llAR1.bb <- function(mu, sigma, rho, dados){
    n <- length(dados)
    iS <- diag(1+rho^2, n)
    diag(iS)[1] <- diag(iS)[n] <- 1
    iS[abs(row(iS)-col(iS))==1] <- -rho
    ymu <- dados - (1-rho)*mu
    return(-0.5*(-n*log(2*pi) - 2*n*log(sigma) + log(1-rho^2) -
                 drop(crossprod(ymu, iS %*% ymu))/sigma^2))
}
```

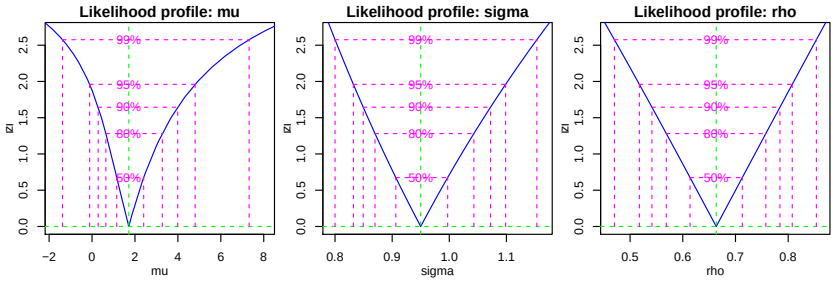

Figura 2.12: Deviances perfilhadas para o ajuste do modelo AR1.

```
require(bbmle)
ar1bb <- mle2(llAR1.bb, start=list(mu=0, sigma=1, rho=0.5),
              data=list(dados=y))
coef(ar1bb)
```

```
      mu     sigma       rho
1.7120298 0.9499764 0.6634720
```

```
logLik(ar1bb)
```

```
'log Lik.' -137.052 (df=3)
```

Gráficos das funções das deviances perfilhadas (expressos na escala de suas raízes quadradas), e respectivos intervalos de confiança são facilmente obtidos por funções auxiliares.

2.10 Exemplo - 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 é $\underline{\theta} = (\mu, \sigma)^\top$, 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 a funçãode

log-verossimilhança é uma superfície. 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,

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

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 quadrática coincide com a forma exata na direção deste parâmetro. 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 uma transformação não linear como $\psi = \log(\sigma)$. Nos remetemos à discussão na Sessão 2.8 para mais detalhes e implicações. Na prática, se intervalos baseados na aproximação quadrática serão utilizados (por vezes a partir de hessianos numéricos), o mais recomendado então é reparametrizar a verossimilhança como função de ψ para uma obter uma forma mais próxima à simetria. Pode-se então obter intervalos assintóticos 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

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

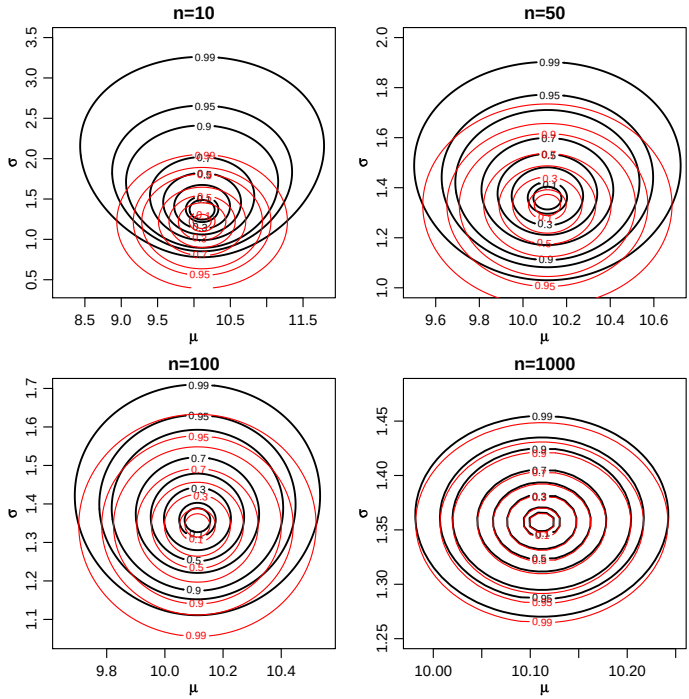

Figura 2.13: Deviance exata (linha sólida) e aproximada (linha tracejada) para diferentes tamanhos de amostra - Distribuição Normal.

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 tenderá a ser melhor quanto maior for o tamanho da amostra. Para exemplificar esta ideia a Figura 2.13 apresenta o gráfico da função *deviance* bidimensional em (μ, σ) para o caso do modelo gaussiano para quatro tamanhos de amostra, n=10, 50, 100 e 1000.

Na Figura 2.13 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 modelos, como no caso dos modelos lineares generalizados (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 outras palavras, estimar a média é mais simples que estimar a 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 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 2.23: Função para log-verossimilhança perfilhada dos parâmetros μ e σ da distribuição gaussiana.

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

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

```
set.seed(123)
y10 <- rnorm(10,10,1.5)
grid.mu <- seq(9, 11.3, length=200)
```

```
grid.sigma <- seq(0.65, 2.7, length=200)
## Condicional para mu:
mu.cond <- sapply(grid.mu, pl.sigma, sigma=sqrt(var(y10)*9/10), dados=y10)
## Condicional para sigma:
sigma.cond <- sapply(grid.sigma, pl.mu, mu=mean(y10), dados=y10)
```

Para obter o perfil de verossimilhança, por exemplo para σ precisamos de uma malha de valores de σ e para cada valor nesta malha encontrar o valor digamos $\hat{\mu}_\sigma$ que maximiza a verossimilhança perfilhada. Para este exemplo existem formas fechadas para os estimadores, portanto basta aplicar a expressão do estimador de um parâmetro para cada valor na malha de valores do parâmetro sendo perfilhando. Entretanto para ilustração utilizamos uma forma mais geral, adequada para casos onde não há expressões fechadas, na qual maximizamos a função utilizando procedimentos numéricos, o que implica em uma otimização numérica de um parâmetro para cada valor na grade do parâmetro que está sendo perfilhado. 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 para o conjunto de 10 dados. O gráfico da esquerda da Figura 2.14 mostra a superfície de deviance com as linhas tracejadas indicando os cortes para obtenção das deviances perfilhadas dos gráficos do centro e a direita. Nestes gráficos são também mostradas as funções deviance condicionadas no MLE (linha sólida).

```
mu.perf <- matrix(0, nrow=length(mu), ncol=2)
for(i in 1:length(mu)){
mu.perf[i,] <- unlist(optimize(pl.mu,c(0,200),
                    mu=mu[i],dados=y10,maximum=TRUE))}
sigma.perf <- matrix(0, nrow=length(sigma), ncol=2)
for(i in 1:length(sigma)){
sigma.perf[i,] <- unlist(optimize(pl.sigma,c(0,1000),
                sigma=sigma[i],dados=y10,maximum=TRUE))}
```

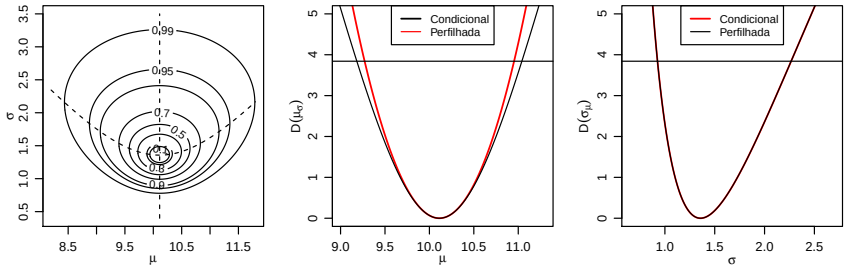

Figura 2.14: Deviance conjunta, perfilhada e condicional para μ e σ - Distribuição Normal.

A Figura 2.14 ilustra que a *deviance* perfilhada e a *deviance* condicional coincidem para o parâmetro σ porém não para o parâmetro de média μ.

Isto reflete o fato de que ao perfilhar σ o valor maximizado $\hat{\mu}_\sigma = \hat{\mu}$ e não depende de σ. Já no perfil de μ cada um dos valores maximizados $\hat{\sigma}_\mu$ dependem dos valores de μ. 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 problema 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.

2.10.1 Dados intervalares

Quando definimos a função de verossimilhança no início deste capítulo, mencionamos que os 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 2.3

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 2.1 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 2.24: Função para log-verossimilhança para dados pontuais e intervalares de distribuição gaussiana.

```
nllnormI <- function(par, yp, YI) {
    ll1 <- sum(dnorm(yp, mean = par[1], sd = par[2], log = T))
    L2 <- pnorm(YI, 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.

```
x <- c(72.6, 81.3, 72.4, 86.4, 79.2, 76.7, 81.3)
t(xI <- 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(x), sd(x)))
```

```
[1] 78.557143  5.063219
```

```
(ests <- optim(ini, nllnormI, y=x, YI=xI)$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 2.25 que pode ser usada com outras densidades com dois parâmetros. Por conveniência definimos também a função

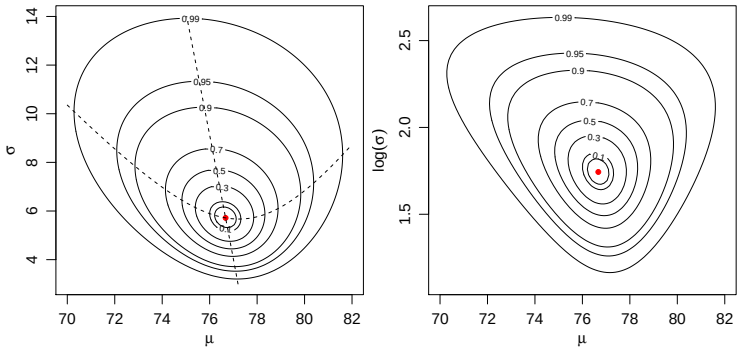

Figura 2.15: Deviances de (μ,σ) e $(\mu,\log(\sigma))$, dados intervalares - Distribuição Normal.

em forma vetorizada que utilizaremos com o comando `outer()` na obtenção das superfícies.

Código 2.25: 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 da Figura 2.15 mostra superfícies de verossimilhança na escala da *deviance* e é obtido com os comandos a seguir. As linhas tracejadas indicam o local do corte na superfície de deviance para obter a verossimilhança perfilhada. O gráfico da direita usa a parametrização $\log(\sigma)$. O aspecto talvez mais importante é notar que, diferentemente dos gráficos 2.13, com dados intervalares a superfície não mais exibe ortogonalidade entre os parâmetros.

```
mu <- seq(70,82, l=100)
sigma <- seq(3, 14, l=100)
devMS <- outer(mu, sigma, FUN=devSurf, llFUN=nllnormI,
               est=ests, yp=x, YI=xI)
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)
```

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

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

Neste exemplo fizemos a suposição de distribuição gaussiana para os dados, mas os mesmos princípios e procedimentos são aplicáveis a outras distribuições. O procedimento pode ser usado com dados puramente intervalares como por exemplo dados agrupados. Suponha que os dados sejam provenientes de alguma fonte da qual se tenha apenas a distribuição (tabela) de frequências. Podemos então definir a verossimilhança como no exemplo e obter as estimativas mesmo sem ter acesso aos dados originais, ainda que com menor precisão.

Dados intervalares são muitas vezes tratados pelo termo *dados censurados*, refletindo o fato de que o dado real não é observado devido a alguma restrição (censura). Tais dados podem ocorrer, por exemplo, devido a limites de detecção de aparelhos que podem ser incapazes de obter medidas acima (censura à direita) e/ou abaixo (censura à esquerda) de certos limites. Outra situação são medições que por alguma razão só podem ser feitas entre dois valores (censura intervalar). Dados censurados são discutidos em diversas áreas e entre elas são discutidos detalhadamente no escopo de análise de sobrevivência. Recomendamos os textos de Giolo & Colosimo (2006) e Carvalho et al. (2011) para leitores interessados no assunto.

2.10.2 Informação de cada dado

Na sessão anterior mostramos como dados pontuais e intervalares podem ser combinados na verossimilhança. Entretanto, a informação contida em cada um deles não é a mesma. É intuitivo que um dado pontual contém mais informação do que um intervalar, em especial em distribuições

com um parâmetro de dispersão como a normal. Este fato pode ser visto e descrito na verossimilhança que pode ser feita para cada observação individualmente. A mais informativa vai ser mais "fechada", ou seja exibir uma maior curvatura.

Para ilustrar vamos considerar o seguinte exemplo, adaptado de Pawitan (2001). Considere $Y \sim N(\theta, 1)$ e as as seguintes observações:

1. $y = 2.45$,
2. $0.9 < y < 4$,
3. $y_{(5)} = 3.5$ é o máximo de um grupo de cinco outras observações.

Sejam $\phi(\cdot)$ e $\Phi(\cdot)$ a densidade e densidade acumulada da normal padrão, respectivamente. A verossimilhança para cada uma das observações é calculada da seguinte forma:

$$
\begin{aligned}
L(\theta; y) = \phi(y-\theta) &\equiv \frac{1}{\sqrt{2\pi}} \exp\{-\frac{1}{2}(y-\theta)^2\}; \\
L_1 = L(\theta; y = 2.45) = \phi(y-\theta) &= \frac{1}{\sqrt{2\pi}} \exp\{-\frac{1}{2}(2.45-\theta)^2\}; \\
L_2 = L(\theta; 0{,}9 < y < 4) &= \Phi(4-\theta) - \Phi(0{,}9-\theta); \\
L_3 = L(\theta; y_{(5)} = 3.5) &= 5\{\Phi(y_{(n)}-\theta)\}^{5-1}\phi(y_{(5)}-\theta).
\end{aligned}
$$

Note que a última verossimilhança decorre de um argumento multinomial e com

$$F(y) = P(Y_{\{n\}} \leq y) = P[Y_{\{i\}} < y \; \forall i \neq n \text{ e } Y_{\{n\}} = y]$$

Os códigos para obtenção das verossimilhanças são mostrados a seguir.

```
theta.vals <- seq(-0.5, 5.5, l=201)
L1 <- function(theta) dnorm(2.45, m=theta, sd=1)
L1.vals <- L1(theta.vals)
plot(theta.vals, L1.vals/max(L1.vals), ty="l", col=2, lty=2,
     xlab=expression(theta), ylab=expression(L(theta)))
##
L2 <- function(theta)
        pnorm(4,mean=theta,sd=1)-pnorm(0.9,mean=theta,sd=1)
L2.vals <- L2(theta.vals)
lines(theta.vals, L2.vals/max(L2.vals), ty="l",  lty=5,
      col="darkolivegreen")
##
L3 <- function(theta)
        5*pnorm(3.5,m=theta,s=1)^4 * dnorm(3.5,m=theta,s=1)
L3.vals <- L3(theta.vals)
lines(theta.vals, L3.vals/max(L3.vals), ty="l", lty=4, col=4)
```

Pode-se ainda considerar a função de verossimilhança conjunta das três observações que, assumindo independência, é dada pelo produto da verossimilhanças individuais $L(\theta) = L1 \cdot L2 \cdot L3$.

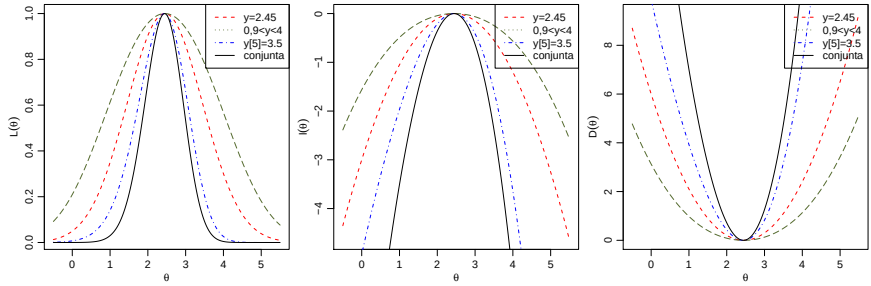

Figura 2.16: Verossimilhanças relativas, log-verossimilhanças e deviances para as observações individuais e a conjunta.

```
L4 <- function(theta)
  L1(theta) * L2(theta) * L3(theta)
L4.vals <- L4(theta.vals)
lines(theta.vals, L4.vals/max(L4.vals), ty="l")
legend("topright", c("y=2.45", "0,9<y<4", "y[5]=3.5", "conjunta"),
       lty=c(2,3,4,1), col=c("red","darkolivegreen","blue","black"))
```

Curvas da função de log-verossimilhança $l(\theta) = \log[L(\theta)]$ podem ser obtidas notando que, em geral, este é um cálculo computacionalmente mais adequado e estável. Outra alternativa é traçar curvas da função deviance $D(\theta) = -2[l(\theta) - l(\hat{\theta})]$. Nos gráficos a seguir utilizamos um valor máximo computado para a sequência de valores para o parâmetro como uma aproximação de $l(\hat{\theta})$. As funções de verossimilhança, log-verossimilhança e deviance (escalonadas) são mostradas na Figura 2.16. Notamos no gráfico as diferentes curvaturas para cada tipo de dado. O intervalar é o menos informativo, seguido pelo pontual. O máximo é mais informativo pois, além de ser pontual, temos também a informação de sua ordenação.

A estimativa do parâmetro pode ser obtida de forma usual maximizando a função de (log)verossimilhança conjunta das observações.

```
ll.ex3 <- function(theta){
  l1 <- -0.5 * (log(2*pi) + (2.45 - theta)^2)
  l2 <- log(pnorm(4, mean=theta, sd=1) - pnorm(0.9, mean=theta, sd=1))
  l3 <- log(5) + 4*pnorm(3.5, mean=theta, sd=1, log=T) +
    dnorm(3.5, mean=theta, sd=1, log=T)
  return(l1+l2+l3)
}
optimize(ll.ex3, interval=c(0,5), maximum=TRUE)
```

```
$maximum
[1] 2.442739

$objective
[1] -1.544042
```

2.11 Exemplo - 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 distribuição 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 $E(Y) = a \cdot s$ e $V(Y) = a \cdot s^2$. 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 inicialmente 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,

$$\begin{aligned} U(a) &= -n \log(s) - n \frac{\Gamma'(a)}{\Gamma(a)} + \sum_{i=1}^{n} \log y_i \\ U(s) &= -\frac{na}{s} + \frac{1}{s^2} \sum_{i=1}^{n} y_i. \end{aligned}$$

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

$$\left\{ \begin{array}{ll} \log(s) + \Psi(a) & = \frac{1}{n} \sum_{i=1}^{n} \log y_i \\ a \cdot s & = \bar{y} \end{array} \right.$$

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

$$\hat{s} = \frac{\overline{y}}{a}. \tag{2.13}$$

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:

$$l_s(a) = -na\log\frac{\overline{y}}{a} - n\log\Gamma(a) - \frac{a}{\overline{y}}\sum_{i=1}^{n} y_i + (a-1)\sum_{i=1}^{n}\log y_i \tag{2.14}$$

$$U_s(a) = -n\log(\overline{y}/a) - n\frac{\Gamma'(a)}{\Gamma(a)} + \sum_{i=1}^{n}\log y_i \tag{2.15}$$

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 2.14. Alternativamente, podemos obter a estimativa igualando a equacão 2.15 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.

Em geral, os métodos numéricos requerem valores iniciais para os parâmetros para inicializar o algoritmo. No caso da Gamma há uma escolha possível dos valores iniciais que seriam os estimadores pelo método dos momentos. Denotando $\hat{\sigma_Y}^2 = \sum_{i=1}^{n}(Y_i - \bar{Y})^2/n$, estes estimadores são dados por:

$$\hat{a}_M = \frac{\bar{Y}}{\hat{s}_M} \quad \text{e} \quad \hat{s}_M = \frac{\hat{\sigma}_Y^2}{\bar{Y}}.$$

Uma aproximação seria substituir σ_Y^2 pela variância amostral. O uso de bons valores iniciais garante um melhor comportamento dos algoritmos numéricos.

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 observada e/ou esperada.

Derivando as funções escore temos,

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

Logo, a matriz de informação observada é dada por,

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

A matriz esperada é obtida tomando a esperança da matriz observada, lembrando que $E(Y) = a \cdot s$, 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 2.17 apresenta a superfície de log-verossimilhança e sua aproximação quadrática em escala de *deviance* para facilitar a construção e visualização do gráfico. Os dados utilizados foram gerados da distribuiição Gama, com parâmetros $a = 10$ e $s = 5$.

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 $a \cdot s$, além disso fica 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 2.14 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

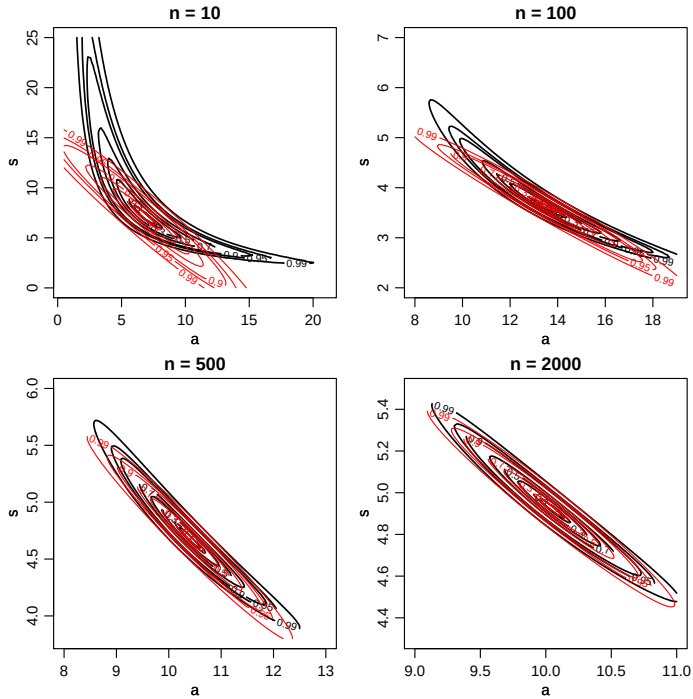

Figura 2.17: Deviance exata e aproximada por tamanho de amostra - Distribuição Gama.

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 diferenciável 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:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{2.16}$$

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 2.27.

Código 2.27: Algoritmo genérico de Newton-Raphson.

```
NewtonRaphson <- function(initial, escore, hessiano, tol=0.0001,
                  max.iter, n.dim, print=FALSE, ...){
  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 densas, 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. A partir de 2.15 temos que:

$$U_s(a) = -n\log(\overline{y}/a) - n\Psi(a) + \sum_{i=1}^{n}\log y_i$$

$$U_s'(a) = \frac{n}{a} - n\Psi'(a)$$

em que $\Psi'(a)$ é a função trigama que é a derivada segunda do logaritmo da função gama.

Escrevendo estas funções em R temos o código 2.28.

Código 2.28: 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)}
```

Gerando 100 valores da distribuição Gama com parametros $a = 10$ e $s = 5$, obtemos os valores iniciais aproximados ou exatos correspondendo aos estimadores dos momentos:

```
set.seed(123)
y100 <- rgamma(100,shape=10,scale=5)
My <- mean(y100) ; Vy <- var(y100)
(initAprox <- c(My^2/Vy , Vy/My))
```

```
[1] 13.296869  3.679325
```

```
n <- length(y100) ; Vy <- Vy * (n-1)/n
(init <- c(My^2/Vy , Vy/My))
```

```
[1] 13.431181  3.642532
```

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

```
(a.hat <- NewtonRaphson(initial = init[1], escore = escore ,
       hessiano=hessiano, max.iter=100, n.dim=1, y=y100))
```

```
[1] 13.53678
```

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 iniciais.

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

```
(s.hat <- mean(y100)/a.hat)
```

```
[1] 3.614117
```

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,

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

poderíamos usar também a matriz de informação observada que é assintoticamente equivalente. O código 2.29 implementa a matriz de informação esperada e observada e constrói os intervalos de confiança assintóticos para a e s.

Código 2.29: 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=y100)
```

```
         [,1]      [,2]
[1,]  7.666854  27.66927
[2,] 27.669273 103.63604
```

```
Io(a = a.hat, s = s.hat, y=y100)
```

```
          [,1]      [,2]
[1,]  7.666854  27.66927
[2,] 27.669273 103.63604
```

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=y100))))
 (ic.a <- a.hat + c(-1,1)*qnorm(0.975)*erro.padrao[1])
```

```
[1]  9.829956 17.243600
```

```
 (ic.s <- s.hat + c(-1,1)*qnorm(0.975)*erro.padrao[2])
```

```
[1] 2.605898 4.622336
```

Outra forma é a construção de intervalos baseados no perfil de verossimilhança. As funções em 2.30 implementam o perfil de verossimilhança para os parâmetros a e s, respectivamente.

Código 2.30: Funções para a log-verossimilhança perfilhada em relação aos parâmetros a e s da distribuição Gama.

```
perf.a <- function(s, a, dados){
   ll <- sum(dgamma(dados, shape=a, scale=s, log=TRUE))
   return(ll)}
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 de forma similar ao mostrado no Exemplo 2.10. A Figura 2.18 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=y100,maximum=TRUE)$objective}
## Perfil para s
vero.perf.s <- c()
for(i in 1:length(grid.s)){
vero.perf.s[i] <- optimize(perf.s,c(0,1000),
                  s=grid.s[i],dados=y100,maximum=TRUE)$objective}
## Condicional para a
```

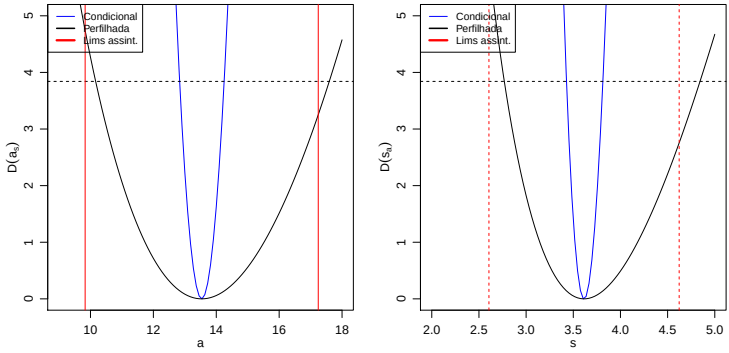

Figura 2.18: Deviance perfilhada, condicional e limites do intervalo de confiança assintótico para a e s - Distribuição Gama.

```
vero.cond.a <- sapply(grid.a,perf.a,s=s.hat,dados=y100)
## Condicional para sigma
vero.cond.s <- sapply(grid.s, perf.s, a= a.hat , dados=y100)
```

Como mostra a Figura 2.18 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.

2.11.1 Parametrizações para Gama

Vamos explorar este exemplo para ilustrar o efeito de diferentes parametrizações no formato 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:

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

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 a parametrização original usada na linguagem **S** e sua primeira implementação no programa S-PLUS[2]. 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
```

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

As expressões ficam então:

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

Função escore:

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

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

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

Parametrização 3: Aitkin (2010) menciona ainda duas parametrizações sendo a primeira considerada a mais usual, e sendo a mesma que é implementada no R. Uma segunda é parametrizada por α e $\mu = \alpha\beta$ com o segundo parâmetro correspondente à média da variável.

Esta segunda parametrização tem propriedades interessantes para inferência. A primeira é a ortogonalidade na matriz de informação entre α e μ. Além disto em geral μ é o usualmente o parâmetro de interesse para inferências e α 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).

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

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:

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

O EMV é solução conjunta de:

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

Nesta parametrização, a partir das expressões de $\frac{d^2l}{d\mu^2}$ e $\frac{d^2l}{d\alpha^2}$ obtemos que os parametros são 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 2.31: 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 2.31 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* o que requer o valor máximo da verossimilhança. Assim, obtemos as estimativas de máxima verossimilhança para as 3 parametrizações, usando os códigos abaixo:

```
mod1 <- optim(c(1,1), neglLik, amostra=am, modelo=1)$par
mod2 <- optim(c(1,1), neglLik, amostra=am, modelo=2)$par
mod3 <- optim(c(1,1), neglLik, amostra=am, modelo=3)$par
cbind(mod1, mod2, mod3)
```

```
         mod1     mod2     mod3
[1,] 4.6926293 4.693253 4.693953
[2,] 0.4152213 2.408705 1.948283
```

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 2.25 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 2.19 mostramos as superfícies para cada parametrização nas escalas originais e logarítmicas dos parâmetros. Simetria e ortogonalidade da superfície facilitam e melhoram o desempenho de algoritmos 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. Há necessidade de uso intensivo de ferramentas do cálculo diferencial para obter a função escore e a matriz de informação. Destacou-se que nem sempre é fácil ou mesmo possível obter analiticamente as quantidades necessários para inferência.

2.12 Exemplo - 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 $y_1, y_2, \ldots, y_n$ para 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:

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

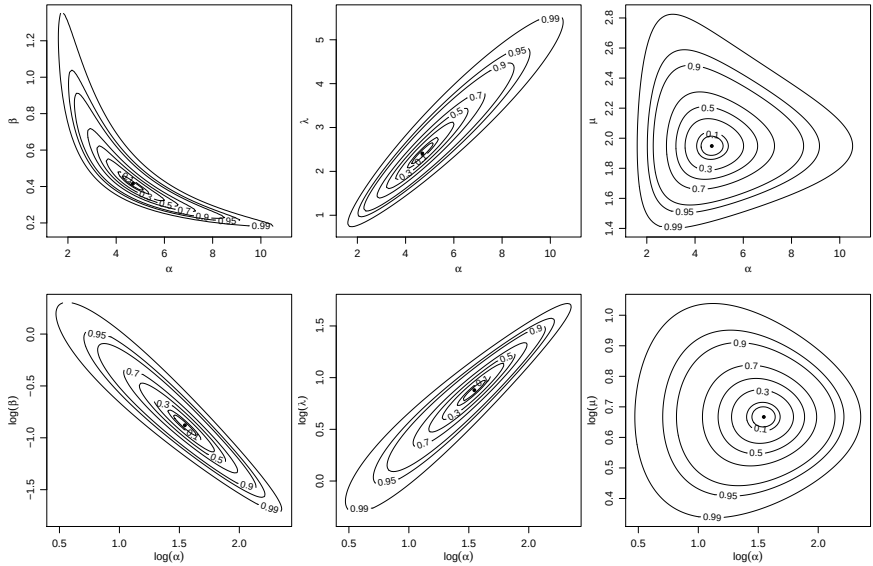

Figura 2.19: Superfícies de deviance sob diferentes parametrizações - Distribuição Gama.

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

$$L(\phi,p) = \prod_{i=1}^{n} \frac{\Gamma(y_i+\phi)}{\Gamma(\phi)y_i!} p^{\phi}(1-p)^{y_i}$$

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

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 obtém-se as equações escore:

$$U(\phi) = n\log p + \sum_{i=1}^{n} \Psi(y_i+\phi) - n\Psi(\phi)$$

$$U(p) = \frac{n\phi}{p} - \sum_{i=1}^{n} y_i/(1-p)\ .$$

A matriz de informação é obtida calculando-se as derivadas:

$$\begin{aligned}
\frac{\partial^2 l(\phi,p)}{\partial \phi^2} = \frac{\partial U(\phi)}{\partial \phi} &= \frac{\partial}{\partial \phi}\left[n\log p + \sum_{i=1}^{n} \Psi(y_i+\phi) - n\Psi(\phi)\right] \\
&= \sum_{i=1}^{n} \Psi'(y_i+\phi) - n\Psi'(\phi) \\
\frac{\partial^2 l(\phi,p)}{\partial p^2} = \frac{\partial U(p)}{\partial p} &= \frac{\partial}{\partial p}\left[\frac{n\phi}{p} - \sum_{i=1}^{n} y_i/(1-p)\right] \\
&= -\frac{n\phi}{p^2} - \sum_{i=1}^{n} y_i/(1-p)^2 \\
\frac{\partial l(\phi,p)}{\partial \phi \partial p} = \frac{\partial U(\phi)}{\partial p} &= \frac{\partial}{\partial p}\left[n\log p + \sum_{i=1}^{n} \Psi(y_i+\phi) - n\Psi(\phi)\right] \\
&= \frac{n}{p}.
\end{aligned}$$

Com os resultados pode-se 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 de forma análoga ao que acontece no modelo Gama. Entretanto, não vamos utiizar a verossimilhança concentrada para poder 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, o que fazemos em código que inicialmente reflete a forma que "escrevemos no papel". Adiante veremos como usar as funções residentes do *R* para implementar um código mais eficiente. Uma amostra simulada de tamanho $n = 100$ e valores dos parâmetros $\phi = 100$ e $p = 0.8$ fornece os dados utilizados.

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

Código 2.32: 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 através do algoritmo de Newton-Raphson, o próximo passo é implementar a função escore e, na sequência, a matriz de segundas derivadas ou Hessiana.

Código 2.33: 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.

2.13 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 computacional e/ou 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 2.10, 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 2.34: Função de log-verossimilhança para a distribuição gaussiana.

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

Note que retornamos o negativo da log-verossimilhança simplesmente porque, por padrão, a função `optim()` minimiza a função objetivo. Informamos um vetor em que cada posição corresponde a cada parâmetro a ser estimado. Para proceder o processo de estimação precisamos de uma amostra, vamos simular uma amostra de tamanho $n = 100$ do modelo gaussiano com $\mu = 10$ e $\sigma = 2$ apenas para ilustrar o procedimento estimação 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 algoritmo numérico é 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 se tente avaliar a função de verossimilhança em pontos não válidos. Este problema

pode gerar desde simples mensagens de *warnings* até a falha total do algoritmo. O único algoritmo dentro da `optim()` que permite busca em espaços determinados é o `L-BFGS-B` é o que vamos usar neste exemplo. Reparametrizações são muito úteis neste ponto para permitir o uso de outros algorítmos que busquem solução no espaço paramétrico irrestrito. No exemplo de estimação da normal $\sigma \in \Re_+^*$ uma reparametrização conveniente seria $\tau = \log(\sigma) \in \Re$. Recomendamos que o leitor leia a documentação da função e experimente os outros algoritmos.

```
set.seed(123)
dados <- rnorm(100, m=10, s=2)
unlist(est.gauss <- optim(par=c(5, 1), fn = ll.gauss, dados=dados,
             method="L-BFGS-B", lower=c(-Inf, 0.1),
             upper=c(Inf,Inf), hessian=TRUE)[1:2])
```

```
    par1      par2     value
10.18081   1.81648 201.58395
```

A saída da `optim()` retorna uma lista. O primeiro elemento é o valor que maximiza o negativo da log-verossimilhança. Em seguida o valor maximizado que a função toma neste ponto. Tal valor é 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 é obtida numericamente, a qual é importante para a construção dos intervalos de confiança assintóticos. Deste ponto, ainda é preciso uma quantidade razoável de programação para obtenção de intervalos baseados na verossimilhança perfilhada. Uma forma mais rápida de obter praticamente tudo que se deseja do processo de inferência é usar funções disponíveis em pacotes tais como o **bbmle**, que implementam os procedimentos de forma genérica e facilitam todo o procedimento.

Para usar as funcionalidades do pacote **bbmle** precisamos escrever a função de log-verossimilhança de forma ligeiramente diferente. O código abaixo apresenta as funções de log-verossimilhança para os casos Gaussianos, Gama e Binomial Negativo discutidos anteriormente neste Capítulo.

Código 2.35: Funções de log-verossimilhança escritas em formato compatível para estimação com a função `mle2()`.

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

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. Para o modelo gaussiano teríamos o código a seguir.

```
require(bbmle)
est.gauss <- mle2(ll.gauss, start=list(mu=8, sigma=3),
                  data=list(dados=dados), method="L-BFGS-B",
                  lower=list(mu=-Inf, sigma=0.1),
                  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 = 8, sigma = 3), method = "L-BFGS-B",
    data = list(dados = dados), lower = list(mu = -Inf, sigma = 0.1),
    upper = list(mu = Inf, sigma = Inf))

Coefficients:
      Estimate Std. Error z value     Pr(z)
mu    10.18081    0.18165  56.047 < 2.2e-16
sigma  1.81648    0.12844  14.142 < 2.2e-16

-2 log L: 403.1679
```

construção de intervalos de confiança assintóticos,

```
confint(est.gauss,method="quad")
```

```
        2.5 %    97.5 %
mu    9.824786 10.536834
sigma 1.564735  2.068229
```

ou construção de intervalos de confiança perfilhados.

```
confint(est.gauss)
```

```
        2.5 %    97.5 %
mu    9.821316 10.540308
sigma 1.591182  2.100618
```

O código a seguir 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    1.89125  7.1574 8.225e-13
s  0.72285    0.10289  7.0256 2.131e-12
```

```
-2 log L: 474.394
```

```
 confint(est.gamma)
```

```
      2.5 %    97.5 %
a 10.1649563 17.594305
s  0.5538589  0.968749
```

```
 confint(est.gamma,method="quad")
```

```
    2.5 %     97.5 %
a 9.829548 17.2430994
s 0.521193  0.9245035
```

De forma análoga pode-se obter resultados 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   1.736160  4.9733 6.583e-07
p   0.772486   0.035561 21.7228 < 2.2e-16

-2 log L: 3853.87
```

```
 confint(est.nbin,method="quad")  # assintotico
```

```
       2.5 %     97.5 %
phi 5.2315753 12.0371975
p   0.7027875  0.8421846
```

```
 confint(est.nbin)                # perfilhada
```

```
       2.5 %     97.5 %
phi 6.0825640 13.7049514
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 implementa métodos genéricos, além de formatar adequadamente a saída na forma de outras funções padrão do R. Implementações para uso intensivo podem, em algumas situações, ser porteriormente ajustadas para um melhor desempenho, se necessário.

Finalmente, é importante dizer que as implementações feitas neste Capítulo visam primariamente ilustrar conceitos de inferência e o uso da linguagem. Algorítmos e funções mais gerais e eficientes podem ser escritos ou, pode-se ainda em muitos casos, utilizar funções já prontas dsiponibilizadas em pacotes específicos. 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*). Por outro lado os códigos ilustram implementações que podem seguir de guia para programação de modelos de interesse que não estejam já implementados.

Capítulo 3

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:

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

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 3.2.

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 mais usuais 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.

Como material adicional apresentamos a implementação computacional de um modelo simple para um processo de Poisson não homogêneo e um estudo de caso com um modelo de regressão para análise de um experimento com contagens subdispersas.

3.1 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 $\beta_0 + \beta_1 x_i$, em que 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 3.36: 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 3.1 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.

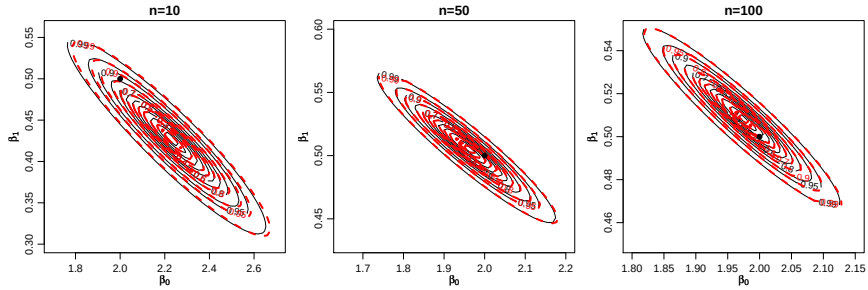

Figura 3.1: Superfície de deviance para diferentes tamanhos de amostra - Regressão Poisson, detalhes no texto.

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 3.37. 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 3.37: 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 2.27, 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 pela 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 os 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.

3.2 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

$$f(y;\mu,\sigma^2) = [2\pi\sigma^2 y(1-y)^3]^{-1/2} \exp\left\{-d(y;\mu)/2\sigma^2\right\}, \quad y \in (0,1), \quad (3.1)$$

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 3.1, 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:

$$\frac{\partial l(\mu_i,\sigma^2)}{\partial \mu_i} = -\frac{1}{2\sigma^2}\frac{\partial}{\partial \mu_i} d(y_i,\mu_i) = \tag{3.2}$$

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

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 3.38: 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)
}
```

Dada a aplicação teremos o vetor $\underline{\beta} = (\beta_0, \beta_1)$ e então o modelo de regressão Simplex em R é apresentado no código abaixo. Definimos a função de ligação *logit* para $h(\cdot) = g^{-1}(\cdot)$ e sua inversa $g(\cdot)$. Note que como o parâmetro σ^2 tem solução analítica e depende apenas de μ_i não precisamos maximizar numericamente em relação a este parâmetro, estamos novamente usando uma log-verossimilhança concentrada. Basta substituirmos seu estimador na expressão da log-verossimilhança. Em nossas implementações este procedimento foi fundamental para conseguir estabilidade nos resultados do algoritmo de maximização da verossimilhança. Retornaremos a esta discussão e apresentamos justificativas na análise bayesiana destes dados na sessão 5.8.2.

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 3.39: 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)
}
```

Código 3.40: 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 3.2 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.

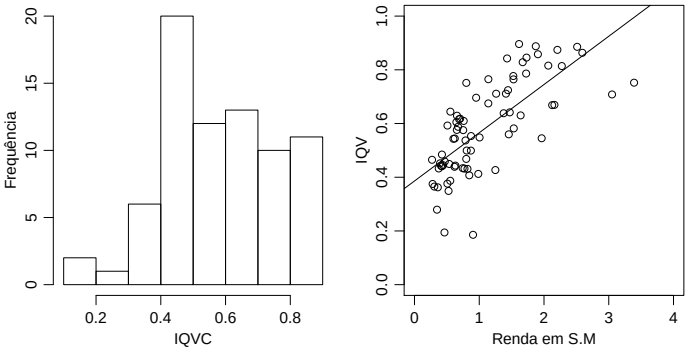

Figura 3.2: Histograma e diagrama de dispersão - IQVC.

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   0.159060 -2.5136   0.01195
b1  0.683628   0.099171  6.8934 5.446e-12

-2 log L: -103.1299
```

```
 confint(simplex.logit.gr, method="quad")
```

```
       2.5 %      97.5 %
b0 -0.7115699 -0.08806427
b1  0.4892567  0.87799989
```

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 3.41: 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 na Figura 3.3 um diagrama de dispersão com as observações e a média predita pelo modelo para a sequência de valores de renda.

3.3 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.

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

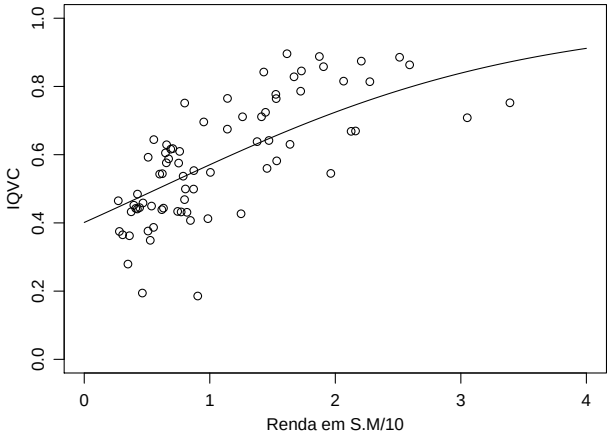

Figura 3.3: Diagrama de dispersão e modelo ajustado - IQVC 2000.

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 de 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 da variável aleatória Poisson são pouco conhecidos. Nenhum dos procedimentos descritos acima para superdispersão ou modelos de efeitos aleatórios como os discutidos no Capítulo 4 são empregados 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).

Na Figura 3.4 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

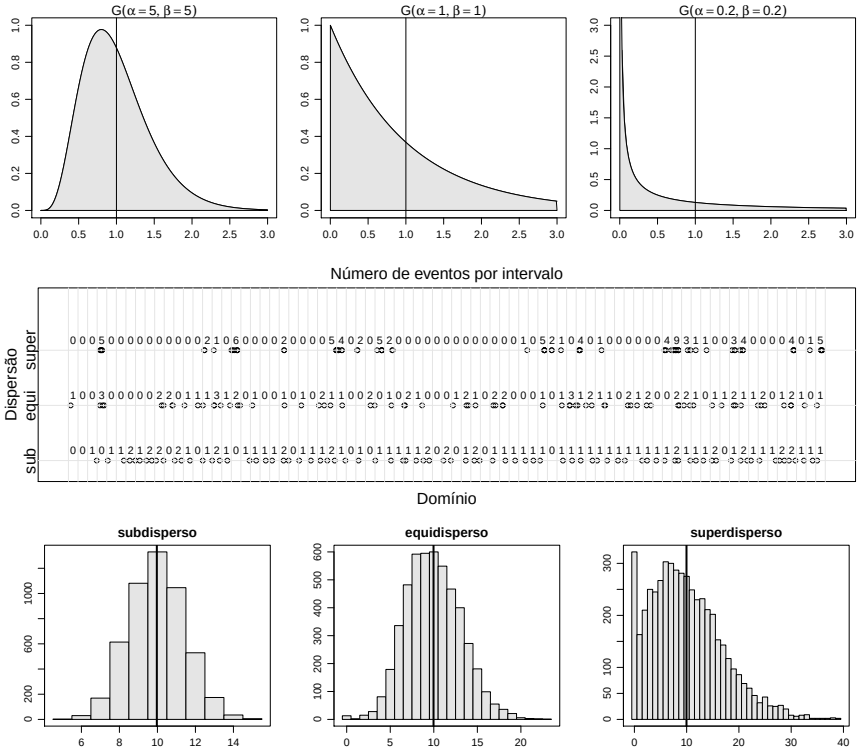

Figura 3.4: Funções densidade de probabilidade para diferentes distribuições de intervalos entre eventos (acima). Eventos distribuídos ao longo de um domínio dividido em intervalos com o número de eventos por intervalo representado (centro). Histogramas do número de eventos por intervalo para cada distribuição de intervalo entre eventos (abaixo).

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

$$P(N = n) = G(1, \alpha n, \beta) - G(1, \alpha(n+1), \beta). \tag{3.3}$$

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 3.3 e estendendo para um modelo de regressão obtemos a seguinte função de log-verossimilhança

$$l(y_i; x_i, \alpha, \gamma) = \sum_{i=1}^{n} \log \left(G(1, \alpha y_i, \alpha \exp(x_i^\top \gamma)) - G(1, \alpha(y_i+1), \alpha \exp\{x_i^\top \gamma)\} \right). \tag{3.4}$$

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

$$E(N|x_i) = \sum_{i=1}^{\infty} G(\alpha i, \alpha \exp(x_i^\top \gamma)). \tag{3.5}$$

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.

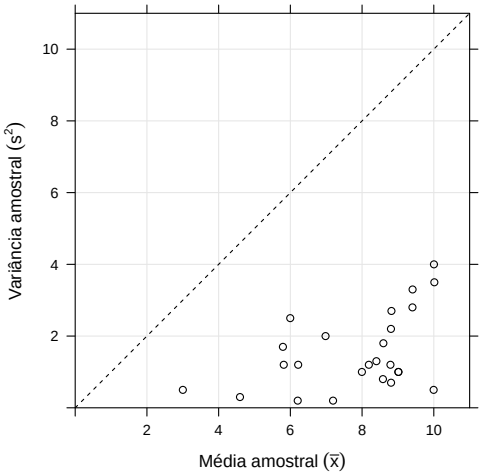

Figura 3.5: Variância amostral em função da média amostral para o número de capulhos do algodão.

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 3.5), 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 3.42.

Código 3.42: 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)))
}
```

A análise exploratória dos dados indica que o valor médio do número de capulhos depende do estágio e do nível de desfolha de forma não adi-

tiva. 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 3.43). 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 3.43: 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)))
}
```

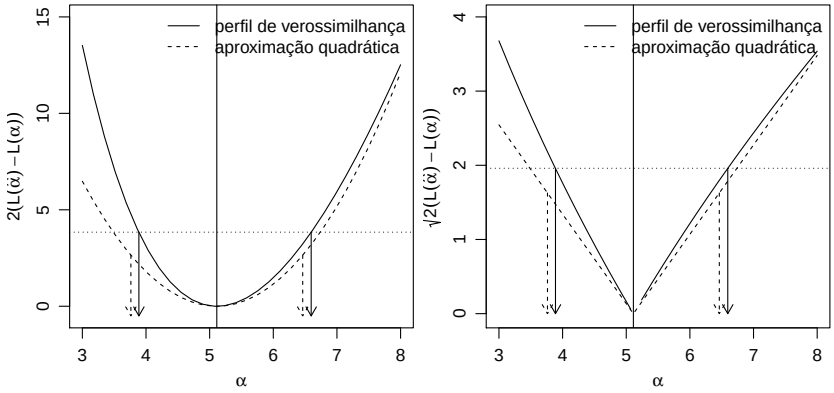

Figura 3.6: Perfil de log-verossimilhança representado pelo valor na diferença na deviance (esq.) e a raíz da diferença na *deviance* (dir.) para o parâmetro α. Setas indicam os limites do intervalo de confiança.

```
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 3.6. 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 uma distribuição mais regular que aquela prevista pela distribuição de Poisson.

Os valores esperados para o número de capulhos foram obtidos pela equação 3.5, bem como os limites do intervalo de predição (Figura 3.7).

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

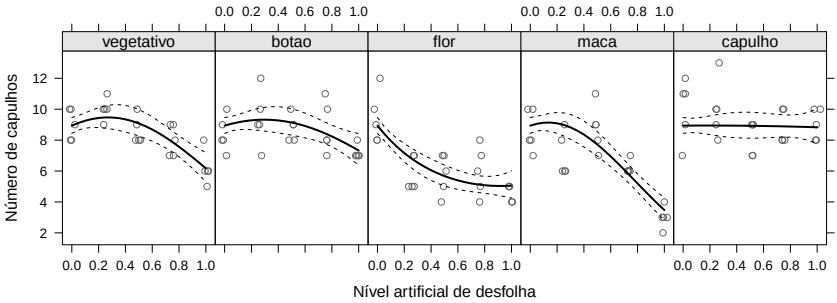

Figura 3.7: Valores observados e preditos acompanhados do intervalo de confiança de 95% para o número de capulhos do algodão em função do nível de desfolha e estágio fenológico.

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 0.68872753  7.42281616 1.146559e-13
(Intercept)            2.234239342 0.02802741 79.71622030 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
```

3.4 Parametrização em modelos não lineares

Um modelo de regressão é não linear quando pelo menos uma derivada do modelo em relação aos parâmetros não for livre dos parâmetros. Em geral, a adoção de modelos de regressão não linear está associado à considerações teóricas sobre a parte mecanística/determinística do fenômeno. Modelos dessa classe são frequentes na física e na química onde os resultados para experimentos simples, como determinar o tempo de queda livre

de um objeto e o produto final de uma reação química, podem ser determinados pelo conhecimento das condições experimentais. Por construção, os modelos não lineares possuem parâmetros com significado de forma que o conhecimento desses parâmetros permite uma interpretação fundamentada do fenômeno por meio dos parâmetros que o determinam/governam.

O modelo logístico é um modelo de regressão não linear largamente usado em diversas áreas. Na área biológica é empregado principalmente para descrever o crescimento ou evolução, como crescimento de seres vivos e evolução de doenças. Devido à grande aplicação por diversas áreas, o modelo apresenta diferentes parametrizações, motivadas pela conveniente interpretação que cada uma delas dá ao fenômeno. Entretanto, como já apresentado nos capítulos anteriores, a parametrização de um modelo têm um importante papel com relação a estabilidade numérica do processo de estimação e a qualidade das inferências estatísticas.

De forma simplificada, o modelo logístico tem a seguinte expressão geral

$$\text{logis}(x, \theta, \beta) = \frac{\theta}{1 + f(\exp\{x\}, \beta)} \tag{3.6}$$

em que x é a variável independente, θ é a assíntota do modelo que representa o valor limite da função quando x tende ao infinito, e $f(\cdot)$ é uma função do vetor de parâmetros β e do exponencial de x. As reparametrizações desse modelo surgem das diferentes formas da função $f(\cdot)$. Vamos aqui considerar quatro parametrizações, a saber

- $f_a(x) = \exp\{(a_1 - x)/a_2\}$;
- $f_b(x) = b_1 \exp\{b_2 x\}$;
- $f_c(x) = \exp\{c_1 + c_2 x\}$;
- $f_d(x) = (-1 + 1/d_1) \exp\{-d_2 x\}$.

Em todas essas parametrizações consideramos θ conhecido e igual a 1 simplesmente para facilitar as ilustrações dos resultados e superfícies de verossimilhança. O vetor β com dois elementos representados por letras arábicas. Todas as expressões são reparametrizações de um mesmo modelo. Para ilustrar, considere o modelo $f_a(\cdot)$ de onde obtemos que

$$\begin{array}{lll} b_1 = \exp\{a_1/a_2\} & c_1 = a_1/a_2 & d_1 = 1/(1 + \exp\{a_1/a_2\}) \\ b_2 = -1/a_2 & c_2 = -1/a_2 & d_2 = 1/a_2. \end{array}$$

Dessa maneira, a partir das estimativas pontuais sob qualquer modelo podemos obter as de qualquer outro aplicando as equações de reparametrização. Este é um fato importante pois podemos ajustar ao dados apenas ao modelo que seja mais adequado para estimação e inferência e também

computacionalmente e a partir deste obter estimativas em outras parametrizações que podem ser desejáveis para interpretações práticas.

Para avaliar a inferência sob diferentes parametrizações, faremos o ajuste de cada uma delas à dados de incidência de ferrugem do pessegueiro, causada pelo fungo *Tranzshcelia discolor*, em função do tempo em plantas da cultivar Chimarrita (figura 3.8). O pomar foi implantado em 2004 em Curitiba (25°, 55′ 10″ S, 49° 57′ 26″ W, 945m), com espaçamento de 1m entre plantas de 2,5m entre linhas. Cada planta foi considerada como uma unidade experimental onde foram avaliadas o número de folhas com presença de ferrugem em relação ao total. Isso foi feito em 6 ramos marcados por planta durante 14 avaliações com intervalo de 10 à 16 dias entre avaliações. Para análise considerou-se a proporção observada de folhas atacadas.

Considerou-se o modelo gaussiano para as observações e a função de log-verossimilhança foi implementada de forma a receber qualquer modelo de regressão não linear (código 3.44). Esta suposição é usual na prática embora, neste contexto, uma resposta binomial poderia ser considerada. As funções em R para os modelos foram criadas e a estimação foi feita pela `optim()`. Após a estimação foram obtidos os contornos de log-verossimilhança para os pares de parâmetros mostrados na Figura 3.9.

Código 3.44: Função de log-verossimilhança para modelos de regressão não linear.

```
ll <- function(th, y, x, model){
  ex <- do.call(model, list(x=x, th=th))
  sd <- sqrt(crossprod(y-ex)/length(x))
  ll <- sum(dnorm(y, mean=ex, sd=sd, log=TRUE))
  ll
}
```

```
# parametrizações
f.a <- function(x, th){ 1/(1+exp((th[1]-x)/th[2])) }
f.b <- function(x, th){ 1/(1+th[1]*exp(th[2]*x)) }
f.c <- function(x, th){ 1/(1+exp(th[1]+th[2]*x)) }
f.d <- function(x, th){ 1/(1+(-1+1/th[1])*exp(-th[2]*x)) }
# dados
y <- dados$inc2; x <- dados$dia
# lista com valores iniciais e modelo
init.list <- list(A=list(par=c(80,13), model=f.a),
                  B=list(par=c(120,-0.06), model=f.b),
                  C=list(par=c(5,-0.06), model=f.c),
                  D=list(par=c(0.008, 0.065), model=f.d))
# lista com termos fixos durante otmização
fixed.list <- list(fn=ll, x=x, y=y, method="BFGS",
                   control=list(fnscale=-1))
# otimização em série dos modelos
op.all <-
```

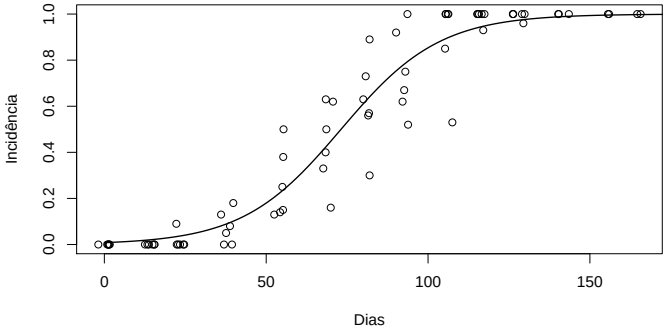

Figura 3.8: Valores observados e ajustados de incidência em função do tempo. As parametrizações são iguais em termos de valores ajustados.

```
   lapply(init.list,
          function(i){
            op <- do.call(optim, c(i, fixed.list)); op
          })
 # estimativas dos parâmetros
 pars <- sapply(op.all, "[[", "par"); pars
```

```
           A           B           C           D
[1,] 73.12710 120.01001195  4.81592735 0.008455968
[2,] 15.23597  -0.06548208 -0.06584788 0.065166770
```

```
 # log-verossimilhança dos modelos
 ll0 <- sapply(op.all, "[[", "value"); ll0
```

```
      A        B        C        D
57.13773 57.13735 57.13700 57.13430
```

```
 # contornos de log-verossimilhança
 leng <- 100
 grid.a <- expand.grid(th1=seq(67,79,l=leng), th2=seq(11,20,l=leng))
 grid.a$ll <- apply(grid.a, 1, ll, y=y, x=x, model=f.a)
 grid.a$dev <- with(grid.a, -2*(ll-ll0["A"]))
 levels <- c(0.05,0.5,0.75,0.9,0.95,0.99)
 qchi <- qchisq(levels, df=2)
 require(lattice)
 contourplot(dev~th1+th2, data=grid.a, at=qchi,
             labels=list(labels=as.character(levels), cex=0.8))
```

Verifica-se que na parametrização A os contornos de `deviance` são praticamente elípticos com eixos paralelos aos eixos cartesianos. Do ponto de vista de otimização, o procedimentos baseados em avaliação de gradiente tem melhor taxa de convergência quando à função objetivo apresenta simetria e ortogonalidade. Do ponto de vista de inferência estatística, por

exemplo na obtenção de intervalos de confiança, a ortogonalidade permite fazer inferência para um parâmetro sem a necessidade de correções considerando a curvatura na direção dos demais parâmetros.

A parametrização *C*, em comparação com *A*, apresentou rotação dos eixos do contorno com inclinação negativa e uma leve assimetria dos contornos. A rotação deve-se ao fato de que as funções de reparametrização entre *A* e *C* são praticamente lineares e a assimetria devido a uma delas ser um quociente. As parametrizações *B* e *D* apresentaram forte desvio da forma elíptica com contornos em formato de "banana". Dentre esses últimos, o modelo *B* parece ser o mais susceptível e pode apresentar problemas numéricos pois apresenta assimetria mais acentuada do que o modelo *D*. Dessa forma, as parametrizações em termos de conveniência para estimação e inferência estatística seguem a ordem de preferência: *A*, *C*, *D* e *B*. Vale relembrar que tanto as estimativas pontuais quanto intervalares do modelo em uma parametrização podem levar aos seus equivalentes em outra parametrização pelo princípio da invariância do estimador de máxima verossimilhança.

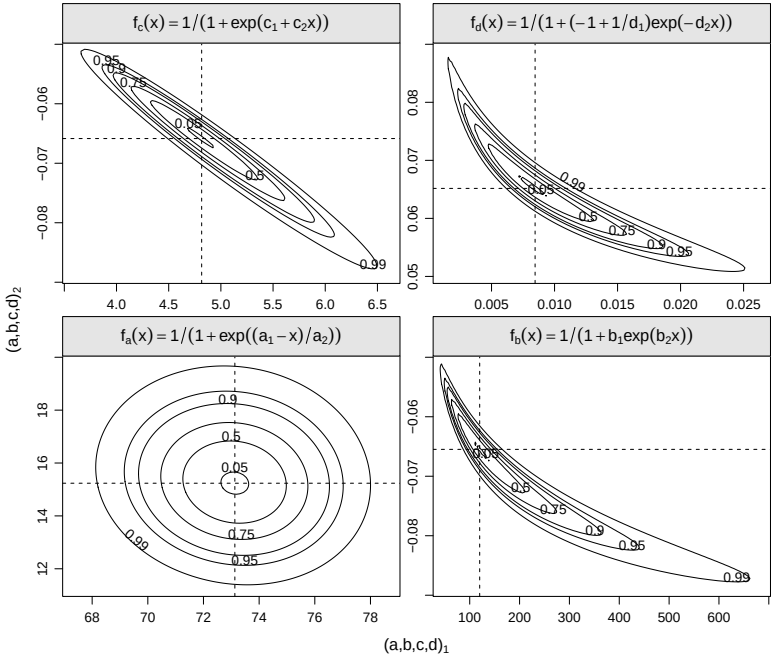

Figura 3.9: Contornos de *deviance* para os parâmetros do modelo logístico em cada uma das parametrizações.

3.5 Processo Pontual

Um **processo pontual** é um tipo de processo estocástico no qual cada realização consiste de um conjunto contável de posições $\boldsymbol{x}_i$ registradas em uma região. Os pontos de ocorrência são chamados de **eventos**. As posições são registradas em tempos e/ou localizações no espaço, embora possam ser anotadas em outros espaços. Tipicamente o objetivo é verificar se há algum padrão que possa ser associado à ocorrência dos pontos. Denotamos por $N(A)$ o número de eventos que ocorre em uma região,

$$N(A) = \#(\boldsymbol{x}_i \in A).$$

O processo é *estacionário* se, para qualquer inteiro k e regiões $A_i : i = 1, \ldots, k$, a distribuição conjunta de $N(A_1), \ldots, N(A_k)$ é invariante a *translações* por uma quantidade arbitrária $\boldsymbol{x}$.

Um processo pontual pode ser caracterizado por sua *função de intensidade* $\lambda(\boldsymbol{x})$ que está associada à probabilidade de ocorrência de eventos na região. As seguintes definições são análogas aos conceitos de média e (co)variância para processos com valores nos reais e caracterizam a ocorrência e interação dos eventos em um processo pontual. Denote por $d\boldsymbol{x}$ uma pequena região contendo o ponto $\boldsymbol{x}$.

Definição 3.1 (Função de intensidade (de primeira ordem))**.** *de um processo pontual é*

$$\lambda(\boldsymbol{x}) = \lim_{|d\boldsymbol{x}| \to 0} \left\{ \frac{E[N(d\boldsymbol{x})]}{|d\boldsymbol{x}|} \right\}$$

Definição 3.2 (Função de intensidade de segunda ordem)**.** *de um processo pontual é*

$$\lambda_2(\boldsymbol{x}_i, \boldsymbol{x}_j) = \lim_{\substack{|d\boldsymbol{x}_i| \to 0 \\ |d\boldsymbol{x}_j| \to 0}} \left\{ \frac{E[N(d\boldsymbol{x}_i) N(d\boldsymbol{x}_j)]}{|d\boldsymbol{x}_i||d\boldsymbol{x}_i|} \right\}$$

Definição 3.3 (Densidade de covariância)**.** *de um processo pontual é*

$$\gamma(\boldsymbol{x}_i, \boldsymbol{x}_j) = \lambda_2(\boldsymbol{x}_i, \boldsymbol{x}_j) - \lambda(\boldsymbol{x}_i)\lambda(\boldsymbol{x}_j).$$

Se o processo pontual é estacionário (e isotrópico no caso de processos espaciais) segue que:

(i) $\lambda(\boldsymbol{x}) = E[N(d\boldsymbol{x})]/|d\boldsymbol{x}| \equiv \lambda = E[N(A)]/|A|$, (constante, em todo A)

(ii) $\lambda_2(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \lambda_2(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|)$ (depende apenas da distância $s = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$)

(iii) $\gamma(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \gamma(s) = \lambda_2(s) - \lambda^2$.

Uma classe particular de processos pontuais são os **processos de Poisson**, nos quais os eventos ocorrem de maneira independente uns dos outros, ou seja, a ocorrência de um evento em uma localização não modifica a probabilidade de ocorrência de eventos em sua vizinhança.

Definição 3.4 (Processo de Poisson)**.** *Seja $\lambda(\boldsymbol{x})$ uma função não negativa, a intensidade de um processo pontual. Então:*

1. *O número de eventos, $N(A)$, em qualquer região A segue uma distribuição Poisson com média*
$$\mathrm{E}[N(A)] = \int_A \lambda(\boldsymbol{x}) d\boldsymbol{x}$$
2. *dado $N(A) = n$, as localizações dos n eventos em A formam uma amostra aleatória independente da distribuição em A com função de densidade de probabilidade proporcional a $\lambda(\boldsymbol{x})$.*

Se sob as condições anteriores $\lambda(\boldsymbol{x}) = \lambda$, para todo $\boldsymbol{x}$, o processo é dito ser um **processo de Poisson homogêneo** (PPH) e corresponde a total aleatoriedade. Neste caso, λ é interpretado como o número esperado de eventos por unidade de tamanho da região, ou seja, o número esperado de eventos na região é $\lambda||A||$ em que $||A||$ é a dimensão da região.

De forma mais geral se $\lambda(\boldsymbol{x})$ não é contante sobre a área, o processo é dito um **processo de Poisson não-homogêneo** (PPI) e o número esperado de eventos em uma região A_i é $\int_{A_i} \lambda(\boldsymbol{x}) d(\boldsymbol{x})$.

Dentre as propriedades dos processos de Poisson estão:

1. Os números de eventos $N(A)$ e $N(B)$ tomados em duas regiões disjuntas são variáveis aleatórias independentes
2. $\mathrm{Var}\{N(A)\}/\mathrm{E}[N(A)] = 1$, em todo A
3. para o processo de Poisson homogêneo, a função de distribuição da distância u de um ponto arbitrário ao evento mais próximo é
$$F(u) = 1 - \exp(-\lambda \pi u^2) : u > 0$$

Em nosso exemplo vamos considerar apenas processos em uma dimensão, por exemplo em um intervalo de tempo, que sejam Poisson homogêneo ou não-homogêneo. Para este último consideramos uma função de densidade $\lambda(t)$ que seja apenas função do tempo. O objetivo é ilustrar como escrever a função de verossimilhança do processo e obter as estimativas.

Inicialmente vamos definir funções para simular realizações dos processos. Para o processo de Poisson homogêneo uma simulação pode ser obtida com os seguintes passos.

1. definir o região para simulação;
2. encontrar o número esperado de eventos na região $\lambda\Delta_t$;
3. simular o número N de eventos de uma Poisson com média $\lambda\Delta_t$;
4. obter as localizações dos eventos simulando de uma distribuição uniforme sobre a região.

Estes passos estão implementados no código 3.45 e a chamada da função simula um processo com $\lambda = 0{,}8$ no intervalo $(0{,}100)$ e portanto com o número esperado de pontos igual a 80.

Código 3.45: Função para simular de um PPH em uma dimensão.

```
simPPH1 <- function(lambda, min, max) {
    Nexp <- lambda * (max - min)
    N <- rpois(1, lambda = Nexp)
    return(runif(N, min = min, max = max))
}
```

```
set.seed(56)
pp1 <- simPPH1(0.8, 0, 100)
(n1 <- length(pp1))
```

```
[1] 77
```

Uma outra maneira de simular de um PPH implementada no código 3.46 é usar o fato de que o intervalo entre eventos possui distribuição exponencial. Desta forma, simula-se sequencialmente os tempos dos eventos e acumulam-se os valores até chegar ao limite do intervalo de tempo desejado. Esta forma pode parecer um pouco mais extensa mas a estratégia de simulações sequenciais é útil por poder ser mais facilmente expandida para outros tipos de processos.

Código 3.46: Função para simular de um PPH em uma dimensão.

```
simPPH2 <- function(lambda, min, max) {
    x <- min
    while (sum(x) <= max) x <- c(x, rexp(1, rate = lambda))
    x <- x[-c(1, length(x))]
    return(cumsum(x))
}
```

Na figura 3.10 são mostradas duas simulações uma com cada uma das funções apresentadas. Sob a suposição de PPH a densidade acumulada dos valores das posições dos eventos é a de uma distribuição uniforme. Os

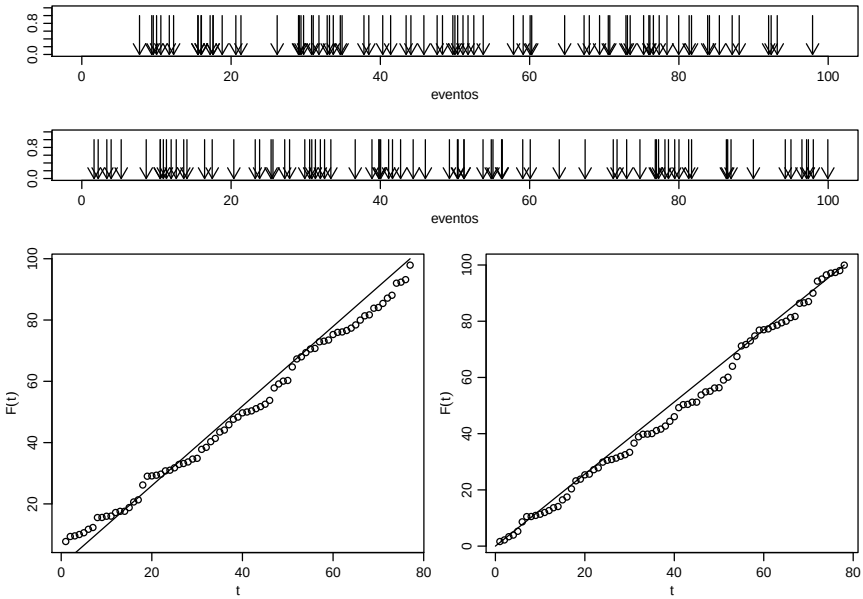

Figura 3.10: Simulações do PPH com as funções apresentadas (superiores) e densidades acumuladas.

gráficos na parte de baixo da figura mostram as distribuições acumuladas e empíricas.

Passamos agora de um processo de Poisson não-homogêneo a considerar um caso simples com $x = t$ e a função de intensidade $\lambda(t) = \exp(\beta_0 + \beta_1 t)$. Este poderia ser um caso no qual estaríamos interessados em avaliar se a taxa de eventos está aumentando com o passar do tempo.

Código 3.47: Função de intensidade de um PPI simples.

```
lambda.x <- function(x, par, log = FALSE) {
    eta <- par[1] + par[2] * x
    if (log)
        return(eta)
    else return(exp(eta))
}
```

Para simular de um PPI podemos usar e estratégia de simulação por rejeição que ilustramos no código 3.48. Para isto simulamos de um processo mais simples (PPH) e submetemos cada evento simulado a uma regra adequada de aceitação/rejeição de tal forma que os eventos aceitos constituem uma simulação do PPI desejado. Os passos são:

1. encontrar o valor $\max[\lambda(t)]$, o máximo de $\lambda(t)$ na região,
2. simular de um PPH na região com $\lambda = \max[\lambda(t)]$,
3. aceitar cada evento simulado com probabilidade $P[\text{aceitar}] = \lambda(t_i)/\max[\lambda(t)]$.

O último passo pode ser descrito como: simular uniformemente no retângulo definido pela região $\times \max[\lambda(t)]$ e selecionar como eventos as abcissas dos pontos para os quais a ordenada é menor do que $\lambda(x)$. No código, adotamos a solução mais geral de maximizar $\lambda(t)$ numericamente, embora para casos em que este valor pode ser encontrado analiticamente, o código pode ser simplificado. No exemplo temos $\max[\lambda(t)] = \lambda(100) = \exp\{-1 + 0{,}015 \cdot 100\} \approx 2$.

Código 3.48: Função para simular de um PPI simples.

```
simPPI <- function(par, min, max) {
    nHPP <- round(optimize(lambda.x, c(min, max), par = par,
        maximum = TRUE)$objective * (max - min))
    X <- runif(nHPP, min, max)
    p.aceita <- lambda.x(X, par) * (max - min)/nHPP
    ppI <- sort(X[runif(nHPP) <= p.aceita])
    attributes(ppI) <- list(simulados = nHPP, aceitos = length(ppI),
        taxa = length(ppI)/nHPP)
    return(ppI)
}
```

A seguir geramos uma realização processo com $\beta_0 = -1$ e $\beta_0 = 0{,}015$ em um intervalo $(0, 100)$, portanto com um número esperado de pontos de 85. Na figura 3.11 a simulação a esquerda mostra claramente o aumento da densidade de pontos com o passar do tempo, o gráfico da direita mostra claramente que o padrão é incompatível com um PPH.

```
set.seed(5665)
pp3 <- simPPI(c(-1, 0.015), 0, 100)
unlist(attributes(pp3))
```

```
 simulados     aceitos        taxa
165.0000000  88.0000000   0.5333333
```

```
(n3 <- length(pp3))
```

```
[1] 88
```

A função de verossimilhança de um processo pontual é obtida considerando as densidades das duas variáveis aleatórias observadas, o número de eventos N e as localizações $\{t\}$. Portanto, para um conjuntos de valores de parâmetros de um modelo considerado, o valor da verossimilhança é dada pela probabilidade de observar o número de eventos

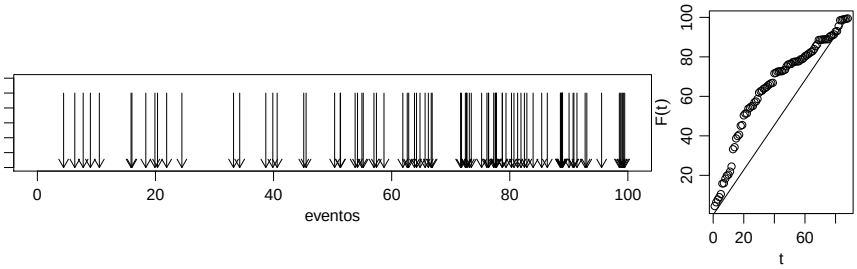

Figura 3.11: Simulação de um PPI (esquerda) e densidade acumuladas (direita).

de fato obtido no conjunto de localizações observadas. Para um processo de Poisson $N \sim P[\int_A \lambda(t)dt]$ e para cada ponto pode ser atribuído $P[\{t_i\}] = \lambda(t_i)/\int_A \lambda(t)dt$.

$$L(\theta) = \frac{e^{-(\int_A \lambda(t)dt)}(\int_A \lambda(t)dt)^N}{N!} \prod_{i=1}^{N} \frac{\lambda(t_i)}{\int_A \lambda(t)dt} \propto e^{-(\int_A \lambda(t)dt)} \prod_{i=1}^{N} \lambda(t_i)$$

$$l(\theta) \propto \sum_{i=1}^{N} \log(\lambda(t_i)) - \int_A \lambda(t)dt$$

Para o PPH o estimador de máxima verossimilhança é intuitivo, tem forma fechada e é dado pela taxa de eventos, ou seja, pelo número de pontos observados dividido pela dimensão $||A||$ da região e a verossimilhança maximizada assume uma forma simples.

$$\hat{\lambda} = \frac{N(A)}{||A||} = \frac{\#(\boldsymbol{x}_i \in A)}{||A||}$$
$$l(\hat{\lambda}) \propto N \log(\hat{\lambda}) - \hat{\lambda}||A||$$

Para os dados da primeira simulação de um HPP temos:

```
 (lambda1.est <- n1/(100-0))
[1] 0.77
 (lambda1.ll <- n1 * log(lambda1.est) - lambda1.est * (100-0))
[1] -97.12509
```

Para um PPI a obtenção do estimador de máxima verossimilhança vai depender da forma de $\lambda(x)$. Para o modelo usado anteriormente para simular de um PPI temos que

$$l(\beta_0, \beta_1) \propto \sum_{i=1}^{N} \log(\beta_0 + \beta_1 t_i) - \int_A \beta_0 + \beta_1 t \; dt.$$

Embora em certos casos seja possível encontrar expressões fechadas para os estimadores, vamos considerar o caso mais geral no qual utilizamos maximização numérica. A expressão da verossimilhança do PPI considerado aqui é definida no código 3.49. Neste código $\lambda(x)$ é integrado numericamente, embora neste exemplo a expressão analítica da integral é facilmente obtida. Se ocorrerem instabilidades numéricas pode-se transformar os valores das localizações dos pontos para o intervalo $(0,1)$. Neste exemplo isto não foi necessário, mas em outras casos não reportados aqui, com diferentes valores dos parâmetros foi fundamental para obtenção das estimativas corretas.

Código 3.49: Função de log-verossimilhança para o PPI simples.

```
nlikIPP <- function(par, PP, min, max) {
    intLx <- integrate(lambda.x, low = 0, upp = 100, par = par)$value
    slxi <- sum(lambda.x(x = PP, par = par, log = T))
    return(-(slxi - intLx))
}
```

As estimativas para a simulação do PPI são obtidas a seguir. Caso seja usado o reescalonamento de coordenadas é necessário transformar os valores dos parâmetros adequadamente. O gráfico à esquerda da Figura 3.12 mostra a superfície de verossimilhança para os dados simulados. Os contornos delimitam, de dentro para fora, regiões de confiança de 5, 10, 30, 50, 70, 90, 95 e 99%. As linhas sólidas indicam o corte para obtenção das verossimilhanças perfilhadas e as pontilhadas das condicionadas no MLE, para cada parâmetro. Os demais gráficos mostram as verossimilhanças individuais perfilhadas e condicionada no MLE, ficando claro que a última subestima dramaticamente a variabilidade por desconsiderar a não ortogonalidade.

```
(ppI.est <- optim(c(0,0.01), nlikIPP, PP = pp3, min=0, max=100)[1:2])
```

```
$par
[1] -1.26651804  0.01965319

$value
[1] 86.31446
```

Possíveis extensões incluem inferências usuais como intervalos de confiança e perfis de verossimilhança. Um resultado de particular interesse é testar se o padrão é homogêneo. Podemos comparar as verossimilhanças maximizadas do PPH e PPI para o mesmo conjunto de dados, por exemplo, para fazer um teste da razão de verossimilhança a fim de determinar se o modelo com taxa não contante é mais compatível com os dados.

```
## PPH
(lambda3.est <- n3/(100-0))
```

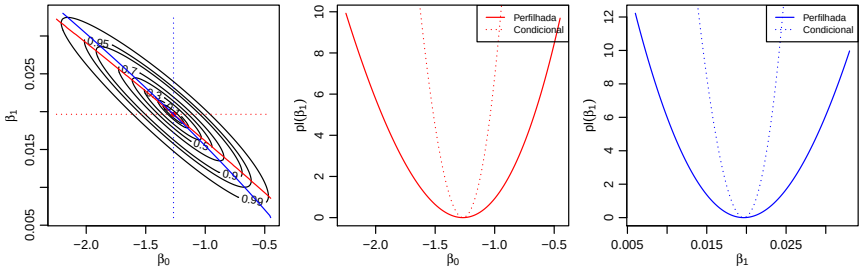

Figura 3.12: Superfície de deviance (esquerda), e perfil de deviances para os parâmetros β_0 (centro) e β_1 (direita) para dados de um PPI.

```
[1] 0.88
 (lambda3.ll <- n3 * log(lambda3.est) - lambda3.est * (100-0))
[1] -99.24934
 -ppI.est$value   ## para PPI
[1] -86.31446
 -2*(lambda3.ll - (-ppI.est$value))
[1] 25.86976
```

Os modelos descritos aqui são básicos e podem ser estendidos de diversas formas. A função $\lambda(t)$ pode ter uma forma mais complexa como, por exemplo, um polinômio ou um *spline* em t. A função intensidade pode ser função de uma outra variável (temperatura, altitude, etc) que não a posição t. Em problemas espaciais no plano as posições dos eventos são dadas pelas coordenadas e a função de intensidade pode ser de duas variáveis $\lambda(x,y)$ e a integração é bidimensional. De forma mais genérica, a intensidade pode ser função de múltiplas coordenadas e/ou variáveis auxiliares o que torna o problema computacionalmente mais complexo por exigir integrações múltiplas. Uma extensão possível é especificar uma função de intensidade que seja aleatória, o que torna o procedimento de inferência bem mais desafiador. Diversos outros processos pontuais que não assumem independência entre eventos são discutidos na literatura. Tais processos podem acomodar interações, tais como atrações ou repulsões, entre eventos e estão além incluem na classe de processos de Poisson.

Como referência inicial para processos pontuais espaciais citamos Diggle (2003). Há diversos pacotes no R [1] implementando métodos para processos pontuais e dentre eles citamos o pacote **spatstat** (Baddeley & Turner, 2005) que além de funções para simular e analisar diversos tipos de processos pontuais possui uma extensa e detalhada documentação.

[1] ver *http://cran.r-project.org/view=Spatial*

Capítulo 4

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 3 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 pode 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 vamos nos limitar a inclusão de efeitos aleatórios gaussianos, como na seguinte especificação:

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

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 observada $[Y] = \int [Y|b][b]db$. As estimativas são obtidas maximizando $[Y]$ 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 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.

4.1 Modelo geoestatístico

O termo geoestatística, refere-se a um conjunto de modelos e métodos para dados com as seguintes características: os valores $Y_i : i = 1, \ldots, n$ são observados em um conjunto finito de localizações amostrais, x_i, em alguma região espacial A, mas que, potencialmente, podem ser medidos em qualquer ponto arbitrário x na área. Cada valor $Y_i = Y(x_i)$ pode ser visto como uma versão ruidosa de um fenômeno espacial contínuo não observável (latente) $S(\cdot)$, nas correspondentes localizações amostrais x_i. O objetivo mais comum neste tipo de análise é "recuperar" o processo latente $S(x)$ o que normalmente pode ser feito obtendo-se predições $\hat{E}[S(x)]$ da média do processo em cada localização. Este procedimento é genericamente conhecido pelo nome de *krigagem*. O modelo geoestatístico como apresentado, dentre diversos outros, em Diggle & Ribeiro Jr. (2007) pode ser definido como um modelo de efeitos aleatórios na forma considerada aqui. Vamos considerar a especificação do modelo para respostas gaussianas, embora a mesma estrutura seja válida com outras distribuições para a variável observável $Y(\cdot)$.

$$\begin{aligned} [Y|b, D] &\sim N(\underline{\mu}, \mathrm{I}\tau^2) \\ \underline{\mu} &= D\underline{\beta} + Z\underline{b} \\ \underline{b} &\sim NMV(\underline{0}, \Sigma_b) \end{aligned} \tag{4.1}$$

em que, D é uma matriz de covariáveis conhecidas com o vetor de parâmetros lineares $\underline{\beta}$ associados a elas, como em um modelo de regressão linear usual. Associamos um efeito aleatório $S(\underline{x})$ a cada posição o que pode ser denotado definindo uma matriz identidade em $Z = \mathrm{diag}(1)$ e um vetor $\underline{b}$, ambos de dimensão igual a n, o número de observações. Portanto o modelo geoestatístico desta forma pode ser interpretado como um modelo de intercepto aleatório com um valor em cada localização. A parte deste modelo que merece mais atenção é a matriz Σ_b que descreve a estrutura da dependência espacial entre as observações. Tipicamente, os elementos desta matriz são dados por uma função de correlação cujo argumento é a distância entre cada par de observações. A literatura especializada apresenta diversas opções para escolha da função de correlação. Na sequência vamos usar a função de correlação exponencial $\rho(||x_i - x_j||) = \exp\{-||x_i - x_j||/\phi\}$ em que o parâmetro ϕ controla a velocidade do decaimento da correlação com o aumento da distância entre localizações. Por exemplo, a matriz conside-

rando apenas três localizações espaciais, toma a seguinte forma:

$$\Sigma_b = \sigma^2 \cdot \begin{bmatrix} 1 & \exp(-u_{12}/\phi) & \exp(-u_{13}/\phi) \\ \exp(-u_{21}/\phi) & 1 & \sigma^2\exp(-u_{23}/\phi) \\ \exp(-u_{31}/\phi) & \sigma^2\exp(-u_{32}/\phi) & 1 \end{bmatrix}$$

em que $u_{ij} = ||x_i - x_j||$ é a distância euclidiana entre as posições espaciais das variáveis $Y(x_i)$ e $Y(x_j)$. Portanto, a matriz de covariância completa parametrizada por σ^2 e ϕ é obtida a partir da matriz de distância entre todas as posições espaciais observadas na amostra. O parâmetro ϕ controla a extensão da dependência espacial entre as observações (alcance), enquanto que o parâmetro σ^2 é a variância dos efeitos aleatórios, nesta caso, o termo espacial do modelo. Em 4.1 é especificado um outro efeito aleatório não correlacionado tal que $\epsilon_i \sim \mathrm{N}(0, \tau^2)$, como o termo de erro no modelo de regressão linear usual. Este último parâmetro de variância ao modelo, por vezes chamado de *efeito de pepita*, e pode ser interpretado como a soma de variações não especiais e de micro escala.

A inferência para os parâmetros envolvidos no modelo $\underline{\theta} = (\underline{\beta}, \underline{\theta}^*)$ com $\underline{\theta}^* = (\sigma^2, \tau^2, \phi)$, pode ser baseada na função de verossimilhança dada pela densidade da distribuição normal multivariada e portanto $Y \sim NMV(D\underline{\beta}, \Sigma)$, com $\Sigma = \Sigma_b + \mathrm{I}\tau^2$. Para ilustrar, a matriz para três observações fica:

$$\Sigma = \begin{bmatrix} \sigma^2+\tau^2 & \sigma^2\exp(-\frac{u_{12}}{\phi}) & \sigma^2\exp(-\frac{u_{13}}{\phi}) \\ \sigma^2\exp(-\frac{u_{21}}{\phi}) & \sigma^2+\tau^2 & \sigma^2\exp(-\frac{u_{23}}{\phi}) \\ \sigma^2\exp(-\frac{u_{31}}{\phi}) & \sigma^2\exp(-\frac{u_{32}}{\phi}) & \sigma^2+\tau^2 \end{bmatrix}.$$

Com isso, tem-se seguinte expressão para a função de verossimilhança,

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

A função de log-verossimilhança fica,

$$l(\underline{\theta}) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(Y - D\underline{\beta})^\top\Sigma^{-1}(Y - D\underline{\beta}). \qquad (4.2)$$

Para estimação dos parâmetros maximizamos 4.2 em relação a $\underline{\theta}$. Temos em $\underline{\theta}$ dois conjuntos de parâmetros, os associados à média ($\underline{\beta}$) e os associados a estrutura de variância e covariância (σ^2, τ^2, ϕ). 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 parâmetro de escala, a derivação não é tão trivial ou mesmo pode não ter expressão analítica fechada e vai depender do modelo da função de correlação. Derivando a função 4.2 em

relação aos $\underline{\beta}$ e igualando a zero, chegamos ao estimador de máxima verossimilhança:

$$\hat{\underline{\beta}} = (D^\top \Sigma^{-1} D)^{-1} (D^\top \Sigma^{-1} Y) \tag{4.3}$$

Substituindo 4.3 em 4.2 obtemos a função de log-verossimilhança concentrada apenas nos parâmetros que definem a estrutura de variância e covariância do modelo.

$$l^*(\underline{\theta}^*) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}\hat{e}^\top \Sigma^{-1}\hat{e} \tag{4.4}$$

com $\hat{e} = (Y - D\hat{\underline{\beta}})$. Os três parâmetros em $\underline{\theta}^*$ indexam a matriz Σ, logo é necessário derivá-los usando cálculo matricial. É possível mostrar que a função escore é dada por,

$$\frac{\partial l^*(\underline{\theta}^*; \underline{Y})}{\partial \theta_i^*} = -\frac{1}{2} Tr\left[\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta_i^*}\right] - \frac{1}{2}\hat{\underline{e}}^\top \left[-\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta_i^*}\Sigma^{-1}\right]\hat{\underline{e}}, \quad i = 1, \ldots, 3. \tag{4.5}$$

em que as matrizes $\dfrac{\partial \Sigma}{\partial \theta_i^*}$ são obtidas derivando cada elemento a matriz Σ em relação ao respectivo parâmetro. Para exemplificar, com duas observações. a derivada de Σ em relação ao parâmetro σ é a matriz:

$$\frac{\partial \Sigma}{\partial \sigma} = 2\sigma \begin{bmatrix} 1 & \exp(-\frac{u_{12}}{\phi}) \\ \exp(-\frac{u_{21}}{\phi}) & 1 \end{bmatrix}.$$

Para o parâmetro τ obtemos:

$$\frac{\partial \Sigma}{\partial \tau} = 2\tau \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

finalmente em relação ao parâmetro ϕ

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

Utilizamos esses resultados para implementação do modelo geoestatístico gaussiano. Iniciamos definindo a função `montaSigma()` que constrói a matriz Σ do modelo geoestatístico a partir dos parâmetros e da matriz de distâncias euclidianas entre as localizações.

No código 4.51 definimos uma função para simular dados segundo o modelo considerado. Na função `simula.geo()` começamos simulando as coordenadas x com localizações no quadrado unitário. Com as coordenadas, calculamos a matriz U de distância entre todos os pontos a partir da

Código 4.50: Função para construir a matriz de variância-covariância para o modelo geoestatístico com função de correlação exponencial.

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

qual montamos a matriz Σ de covariância. Obtemos uma simulação da distribuição normal multivariada por $Y = D\underline{\beta} + \Sigma^{1/2}z$ em que $\underline{z}$ são escores da normal padrão $N(0,1)$ e $\Sigma^{1/2}$ é alguma raiz da matriz de covariâncias. Utilizamos aqui $\Sigma^{1/2} = R'$ tal que $\text{Var}(\text{R}'\underline{z}) = \text{R}'\text{Var}(z)R = R'R = \Sigma$ em que R é a parte superior da decomposição de Cholesky calculada no R por `chol()`. Desta forma, geramos dados do modelo geoestatístico com função de correlação exponencial.

Código 4.51: Função para simular do modelo geoestatístico.

```
simula.geo <- function(beta, s, t, phi, n){
  locs <- data.frame(cX = runif(n), cY = runif(n))
  U <- dist(locs, diag=TRUE, upper=TRUE)
  Sigma <- montaSigma(s=s, t=t, phi=phi, Umat=U)
  z <- rnorm(n)
  Y = beta + crossprod(chol(Sigma), z)
  return(cbind(locs, Y=Y))
}
```

Vamos obter uma simulação supondo que a estrutura de média é composta por apenas um parâmetro $\beta_0 = 50$ e definimos $\sigma^2 = 2^2$, $\tau^2 = 1^2$ e $\phi = 0.25$. A seguinte chamada simula 125 amostras do modelo geoestatístico usando o código 4.51.

```
set.seed(12)
dt <- simula.geo(b=50, s=2, t=1, phi=0.25, n=125)
```

O próximo passo para inferência é a definição da função de log-verossimilhança concentrada como no código 4.52. Com isto podemos maximizar a log-verossimilhança diretamente através da `optim()` ou qualquer outra forma de maximização numérica.

Este código ilustra passo a passo e didaticamente os cálculos envolvidos na expressão 4.4. Entretanto, o código é pouco eficiente computacionalmente por fazer algumas operações desnecessariamente e/ou de maneira pouco eficiente. Tipicamente a função de verossimilhança é avaliada diversas vezes em algum procedimento numérico. Como a matriz U é constante

Código 4.52: Função de log-verossimilhança para o modelo geoestatístico.

```
ll.geo <- function(s, t, phi, dados){
 U <- dist(dados[,1:2], diag=TRUE, upper=TRUE)
 Sigma <- montaSigma(s=s, t=t, phi=phi, Umat=U)
 D <- as.vector(rep(1,length=nrow(dados)))
 invSD <- solve(Sigma, D)
 bhat <- solve(crossprod(invSD, D),crossprod(invSD,dados$Y))
 require(mvtnorm)
 ll = dmvnorm(dados$Y, mean=D%*%bhat, sigma=Sigma, log=TRUE)
 return(-ll)
}
```

deve ser informada como argumento, evitando ser recalculada desnecessariamente a cada iteração. A obtenção de $\hat{\beta}$ por 4.3 requer a inversão da matriz de covariância que pode ser escrita na forma da solução de um sistema. No código acima $\Sigma^{-1}D$ é computado por `solve(Sigma, D)`. Ainda sim há computações desnecessárias pois para resolver este sistema é feita uma decomposição de `Sigma` que é repetida dentro da chamada `dmvnorm()`. Isto é relevante uma vez que as operações com Σ são as mais caras na computação deste modelo. Desta forma, reescrevemos a função no código 4.53 fazendo ainda generalizações para nomes de variáveis e definição da matriz D e incluindo a opção para estimar os parâmetros na escala logarítmica.

Código 4.53: Redefinição da função de log-verossimilhança para o modelo geoestatístico.

```
ll.geo <- function(s, t, phi, modelo, Umat, dados, logpars = F) {
    if (logpars) {
        s <- exp(s)
        t <- exp(t)
        phi <- exp(phi)
    }
    mf <- model.frame(modelo, dados)
    y <- model.response(mf)
    D <- model.matrix(modelo, mf)
    Sigma <- montaSigma(s = s, t = t, phi = phi, Umat = Umat)
    R <- chol(Sigma)
    invRD <- backsolve(R, D, transpose = TRUE)
    invRy <- backsolve(R, y, transpose = TRUE)
    bhat <- solve(crossprod(invRD), crossprod(invRD, invRy))
    invRe <- invRy - invRD %*% bhat
    nll <- drop(length(y) * log(2 * pi)/2 + sum(log(diag(R))) +
        crossprod(invRe)/2)
    return(nll)
}
```

Como temos as expressões 4.5 dos gradientes analíticos, vamos implementá-las para melhorar o desempenho do algoritmo de maximização numérica. Veremos o efeito na comparação dos tempos computacionais para convergência com e sem o uso do gradiente analítico. Para utilizar o gradiente precisamos primeiro implementar três funções com as matrizes de derivadas em relação a cada um dos parâmetros em $\underline{\theta}^*$.

Código 4.54: Funções para definição dos gradientes do modelo geoestatístico.

```
## Derivada de sigma, tau e phi
deriv.s <- function(s, t, phi, Umat){
  Sigma.s <- 2*s*as.matrix(exp(-Umat/phi))
  diag(Sigma.s) <- 2*s
  return(Sigma.s)}
deriv.t <- function(s, t, phi, Umat){
  return(diag(2*t, nrow(as.matrix(Umat))))}
deriv.phi <- function(s, t, phi, Umat){
  return(s^2 * as.matrix(Umat) * exp(-as.matrix(Umat)/phi)/phi^2)}
```

Implementamos a função escore completa em 4.55.

Tanto na função escore como na log-verossimilhança concentrada retornamos o negativo da função para compatibilidade com a função `mle2()` que por *default* minimiza a função objetivo.

Com tudo implementado utilizamos o conjunto de dados simulados e ajustamos o modelo usando a função `mle2()` do pacote **bbmle** por conveniência. Alternativamente, poderíamos usar diretamente a função `optim()` com qualquer um de seus algoritmos, ou mesmo outros maximizadores disponíveis no R. Nos comandos a seguir, estimamos os parâmetros sem e depois com o uso do gradiente analítico pelo algoritmo `L-BFGS-B` e comparamos os tempos computacionais.

```
require(bbmle)
system.time(est1 <- mle2(ll.geo, start = list(s = 1, t = 0.1,
    phi = 0.1), method = "L-BFGS-B", lower = list(s = 0,
    t = 0, phi = 0), data = list(dados = dt, modelo = Y ~
    1, Umat = dist(dt[, 1:2], upper = T, diag = T))))
```

```
   user  system elapsed
  0.844   1.144   0.743
```

```
system.time(est2 <- mle2(ll.geo, gr = escore, start = list(s = 1,
    t = 0.1, phi = 0.1), method = "L-BFGS-B", lower = list(s = 0,
    t = 0, phi = 0), data = list(dados = dt, modelo = Y ~
    1, Umat = dist(dt[, 1:2], upper = T, diag = T))))
```

```
   user  system elapsed
  0.900   0.660   0.741
```

Neste caso, o uso do gradiente analítico diminuiu o tempo para a maximização da log-verossimilhança. Este ganho pode ser expressivo, prin-

Código 4.55: Função escore para o modelo geoestatístico.

```
escore <- function(s, t, phi, modelo, Umat, dados, logpars = F) {
    if (logpars) {
        s <- exp(s)
        t <- exp(t)
        phi <- exp(phi)
    }
    mf <- model.frame(modelo, dados)
    y <- model.response(mf)
    D <- model.matrix(modelo, mf)
    Sigma <- montaSigma(s = s, t = t, phi = phi, U = Umat)
    R <- chol(Sigma)
    invRD <- backsolve(R, D, transpose = TRUE)
    invRy <- backsolve(R, y, transpose = TRUE)
    bhat <- solve(crossprod(invRD), crossprod(invRD, invRy))
    invRe <- invRy - invRD %*% bhat
    e.hat <- y - D %*% bhat
    Sigma1 <- chol2inv(R)
    S1D <- Sigma1 %*% deriv.s(s = s, t = t, phi = phi, U = Umat)
    U.s <- 0.5 * (sum(diag(S1D)) - crossprod(e.hat, S1D %*%
        Sigma1) %*% e.hat)
    T1D <- Sigma1 %*% deriv.t(s = s, t = t, phi = phi, U = Umat)
    U.t <- 0.5 * (sum(diag(T1D)) - crossprod(e.hat, T1D %*%
        Sigma1) %*% e.hat)
    P1D <- Sigma1 %*% deriv.phi(s = s, t = t, phi = phi,
        U = Umat)
    U.phi <- 0.5 * (sum(diag(P1D)) - crossprod(e.hat, P1D %*%
        Sigma1) %*% e.hat)
    return(c(U.s, U.t, U.phi))
}
```

cipalmente com grandes bases de dados e/ou procedimentos computacionalmente intensivos. Entretanto, o uso do gradiente nem sempre é vantajoso ou proporciona um incremento de velocidade da mesma ordem deste exemplo, pois a avaliação da função escore também exige cálculos matriciais. Para um melhor desempenho o código das funções de verossimilhança e escore podem (e devem!) ser reescritos para aproveitar na escore os cálculos já realizados na avaliação função de verossimilhança. O mecanismos do R de criar e acessar dados de **ambiente** (*environments*) podem ser usados aqui. Fazemos esta implementação nos complementos *online*.

Informações do ajuste são resumidas a seguir.

```
summary(est1)
```

```
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 = dt, modelo = Y ~
        1, Umat = dist(dt[, 1:2], upper = T, diag = T)), lower = list(s = 0,
```

```
       t = 0, phi = 0))

Coefficients:
    Estimate Std. Error z value     Pr(z)
s    1.91184    0.45939  4.1617 3.159e-05
t    1.12830    0.12806  8.8107 < 2.2e-16
phi  0.33305    0.20243  1.6452   0.09993

-2 log L: 471.4555
```

Com as estimativas dos parâmetros de covariância pode-se obter a estimativa de β (neste caso um escalar) por 4.3 e sua variância $\text{Var}(\hat{\beta}) = (D^\top \Sigma^{-1} D)^{-1}$ como mostrado a seguir. Note que este cálculo já é feito internamente na função que avalia a verossimilhança.

```
D <- model.matrix(Y ~ 1, data=dt)
Dmat <- as.matrix(dist(dt[,1:2], upper=T, diag=T))
Sigma <- montaSigma(s=coef(est1)[1], t=coef(est1)[1],
                    phi=coef(est1)[1], Umat=Dmat)
R <- chol(Sigma)
invRD <- backsolve(R, D, transpose=TRUE)
invRy <- backsolve(R, dt[,3], transpose=TRUE)
(drop(beta.est <- solve(crossprod(invRD),crossprod(invRD, invRy))))
```

```
[1] 49.42382
```

```
(drop(var.beta.est <- solve(crossprod(invRD))))
```

```
[1] 2.679826
```

Para finalizar o procedimento de inferência no modelo geoestatístico a Figura 4.1 apresenta os perfis de verossimilhança para os parâmetros que indexam a matriz de variância/covariância do modelo. Observa-se assimetria em ϕ e σ. Estes parâmetros são não ortogonais na parametrização usada o que pode ser visto em verossimilhanças conjuntas (não mostradas aqui). Parametrizações alternativas são sugeridas na literatura e usualmente utilizam um novo parâmetro definido pelo quociente entre ϕ e σ. O parâmetro é usualmente ortogonal aos demais τ e por vezes a estimativa está próxima ao limite do espaço paramétrico. Mais notadamente o perfil de verossimilhança para ϕ cresce muito lentamente ou deixa de crescer para valores a direita. Isto reflete o fato deste parâmetro estar relacionado às distâncias entre localizações na área e não há informação estatística para distâncias além dos limites da área.

Verossimilhanças com aspecto distante do quadrático são comuns para alguns parâmetros de modelos com efeitos espaciais. Pode-se tentar atenuar os efeitos fazendo reparametrizações, por exemplo, estimar o $\log(\tau)$ ou alguma outra função adequada, mas em geral há pouca informação na amostra sobre certos parâmetros do modelo. Não há uma "receita" geral para todos os modelos, mas a inspeção de superfícies e verossimilhanças perfilhadas podem sugerir as melhores parametrizações.

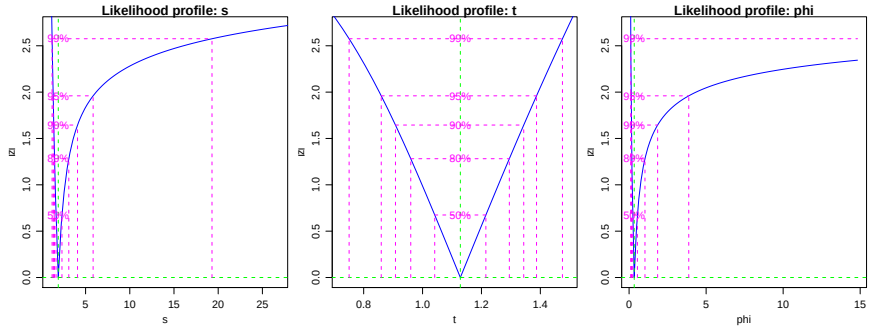

Figura 4.1: Perfis de verossimilhança para o modelo geoestatístico.

4.2 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 podem 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 in-

tegração 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

$$L(\underline{\beta}, \Sigma, \phi) = \prod_{i=1}^{N} f_i(\underline{y}_i|\underline{\beta}, \Sigma, \phi), \tag{4.6}$$

e sob a suposição de independência entre os grupos temos que

$$L(\underline{\beta}, \Sigma, \phi) = \prod_{i=1}^{N} \int \prod_{j=1}^{n_i} f_{ij}(y_{ij}|\underline{b}_i, \underline{\beta}, \phi) f(\underline{b}_i|\Sigma) d\underline{b}_i. \tag{4.7}$$

A principal dificuldade em maximizar 4.6 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 4.1 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 aplicados na estimação de alguns modelos simples para medidas repetidas e dados longitudinais não gaussianos. Vamos iniciar com um modelo simples mais que serve de exemplo para apresentação dos métodos.

4.2.1 Simulação da Poisson com intercepto aleatório

Em todos os métodos de integração numérica que serão apresentados vamos utilizar o modelo de Poisson com intercepto aleatório definido em 4.8 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 4.56: 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 devem ser simuladas. 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 simuladas 10 unidades amostrais e, 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 4.57: 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 um vetor de valores e 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 4.2 apresenta o gráfico do integrando avaliado para cada uma das 10 unidades amostrais simuladas, o eixo y foi padronizado para poder colocar todas as funções no mesmo gráfico.

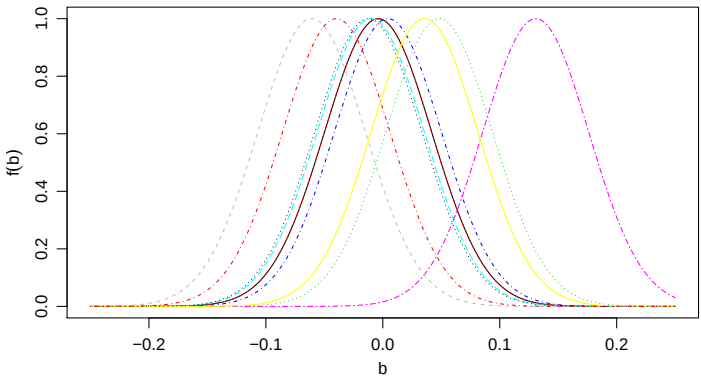

Figura 4.2: Integrando de acordo com unidade amostral - Modelo Poisson com intercepto aleatório.

4.3 Técnicas de integração numérica

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

$$\begin{aligned} Y_{ij}|b_i &\sim P(\lambda_i) \\ \log(\lambda_i) &= \beta_0 + b_i \\ b_i &\sim N(0, 1/\tau^2), \end{aligned} \tag{4.8}$$

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:

$$
\begin{aligned}
f_i(y_{ij}|b_i,\beta_0) &= \int_{-\infty}^{\infty} \frac{\exp\{-\lambda_i\}\lambda_i^{y_{ij}}}{y_{ij}!}\left(\frac{\tau}{2\pi}\right)^{1/2}\exp\{-\frac{\tau^2}{2}b_i^2\}db_i \\
&= \int_{-\infty}^{\infty} \frac{\exp\{-\exp(\beta_0+b_i)\}\exp\{(\beta_0+b_i)^{y_{ij}}\}}{y_{ij}!} \\
&\qquad \left(\frac{\tau}{2\pi}\right)^{1/2}\exp\{-\frac{\tau^2}{2}b_i^2\}db_i. \qquad (4.9)
\end{aligned}
$$

A integral em 4.9 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 usar esta integral 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 deve ser própria com limites finitos e fixados. Este não é o caso na equação 4.9. No uso destes métodos não resolvemos a integral na reta real, mas sim em um domínio finito adequado da função no integrando. Por exemplo, se é razoável assumir que 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 trapezoidal de Simpson, a Quadratura Gaussiana usando os polinômios de Hermite, próprios para a integração na reta real, os 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.

4.3.1 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.

$$I(f) \approx \frac{[f(a)+f(b)]}{2}(b-a) \tag{4.10}$$

A expressão em 4.10 é extremamente simples de ser usada, requer apenas duas avaliações da função integrando. Sua versão em R pode ser escrita como se segue.

Código 4.58: 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.

4.3.2 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 determinados 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:

$$p(x) = \alpha + \beta(x - x_1) + \lambda(x - x_1)(x - x_2) \tag{4.11}$$

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 4.11 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 4.59: 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 4.2 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 de 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.

4.3.3 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 representando $f(x)$ 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 ponderada 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 usados 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 4.60: 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 chamado de Quadratura Adaptativa de Gauss-Hermite, que veremos adiante. A Figura 4.3 ilustra o problema da definição dos pontos de integração.

Pela Figura 4.3 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,
```

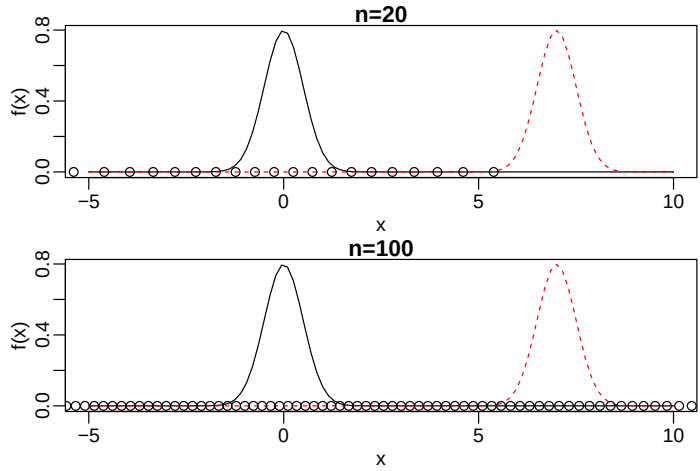

Figura 4.3: Espalhamento dos pontos de integração pelo método de Gauss-Hermite.

```
                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.

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 = 2, prec.pars=4, log=FALSE))
```

```
[1] -20.0701
```

Código 4.61: 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)
}
```

4.3.4 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 $w_1 = 1$, 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 4.62: 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$value) * (exp((n.dim/2)*log(2*pi) -
                  0.5*determinant(-temp$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 4.63: 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 quantidades 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.

4.3.5 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,

$$I = \int_D f(x)dx \tag{4.12}$$

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 4.12 é equivalente a

$$I = \int_D \frac{f(x)}{p(x)} p(x)dx.$$

Essa integral corresponde 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 naturais 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 a cada rodada do algoritmo, obtém-se diferentes valores para a integral o que é indesejável para maximização numérica. 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 sequências de baixa discrepância, estão disponíveis no pacote **fOptions**, são elas `Halton` e `Sobol`. Afora esta escolha de pontos de baixa discrepâncias 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 4.64: 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 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 integração 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.

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.

Código 4.65: 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)
}
```

4.4 Modelo Poisson com intercepto aleatório

Na seção 4.2.1 simulamos dados de um modelo Poisson com intercepto aleatório definido em 4.8. 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 como no código 4.66.

No código 4.67 definimos uma função genérica que capta um conjunto de dados e monta a log-verossimilhança, já integrada de acordo com uma das opções de integração apresentadas anteriormente. Essa função vai ser usada para diversos modelos com efeitos aleatórios apresentados neste texto.

A função `veroM()` foi definida de forma genérica. Alternativamente, pode-se definir uma função específica para cada modelo. Para o modelo Poisson com intercepto aleatório definimos o código 4.68.

Usando a função `mle2()` para estimar os parâmetros via o algoritmo `BFGS`, e aproximação de Laplace, temos o seguinte.

```
system.time(P.laplace <- mle2(mod.Poisson,start=list(b0=0,tau=log(1/4)),
  data=list(integral="LAPLACE",otimizador = "BFGS", n.dim=1,
```

Código 4.66: 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)}
```

```
   dados=dados, pontos=NA)))
```

```
   user  system elapsed
  2.392   0.012   2.406
```

```
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   2.00696    0.07272 27.5986 < 2.2e-16
tau  1.62067    0.29635  5.4687 4.534e-08

-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 exemplo todos os métodos apresentaram valores muito próximos do obtido pela aproximação de Laplace, mais isto não significa que todos possuem o mesmo comportamento numérico. Por exemplo, o método Monte Carlo, requer muitos pontos para convergência, o que o torna muito lento pois a cada iteração estamos re-sorteando de uma gaussiana

Código 4.67: Função de verossimilhança genérica com opções de método 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)))
}
```

Código 4.68: 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)
}
```

multivariada. Alternativamente e de forma mais eficiente, podemos sortear apenas uma vez e usar os mesmos pontos em todas as iterações do algoritmo numérico. O mesmo se aplicada a todos os outros métodos.

Para implementações mais eficientes devemos abandonar ou alterar a função genérica apresentada aqui apenas como exemplo didático. Algoritmos mais eficientes 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 dos métodos de integração numérica.

O mesmo ajuste feito anteriormente utilizando Laplace fornece o resultado a seguir usando quadratura de Gauss-Hermite.

```
system.time(P.GH <- mle2(mod.Poisson,start=list(b0=0,tau=log(1/4)),
  data=list(integral="GH",pontos=100, n.dim=1, dados=dados)))
```

```
   user  system elapsed
  6.916   0.012   6.928
```

```
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   0.291655  5.6081 2.046e-08

-2 log L: 491.7102
```

Repetimos ainda o ajuste com quasi-Monte Carlo com os pontos de Sobol. Neste exemplo, obtemos resultados semelhantes porém com tempos computacionais diferentes.

```
system.time(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)))
   user  system elapsed
  2.836   0.000   2.837
 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   0.072648 27.6227 < 2.2e-16
tau 1.622157   0.296599  5.4692 4.521e-08

-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 os princípios básicos e fundamentos dos algoritmos. Na sequência, vamos discutir alguns modelos com mais elementos, como 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.

4.5 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 4.4 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 4.4 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 $\underline{\theta} = (\beta_0, \sigma^2, \tau^2)$.

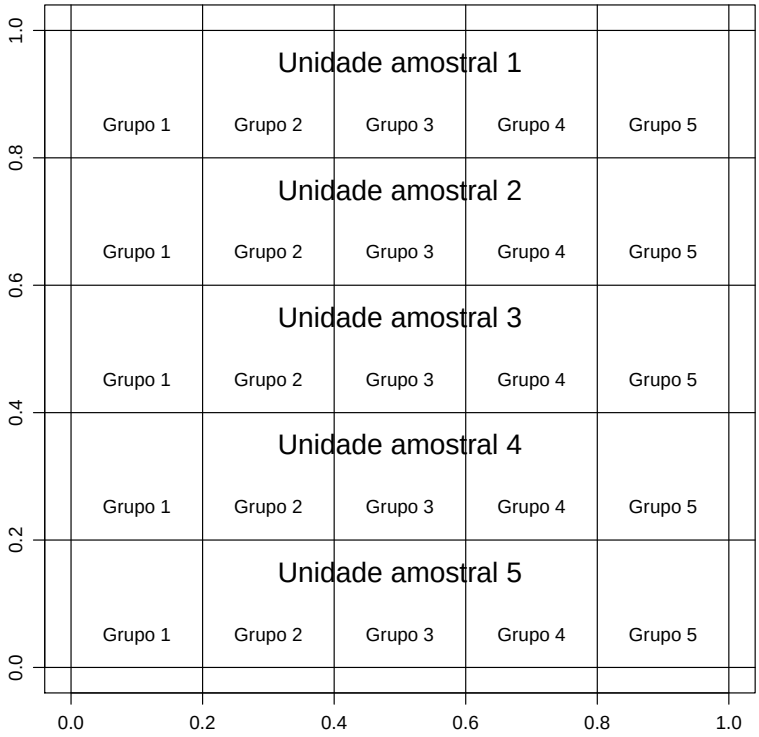

Figura 4.4: Estrutura de um delineamento com efeito aninhado.

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 4.5 ilustra a estrutura do experimento.

Código 4.69: 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
```

A seguir escrevemos uma função com a estrutura do modelo.

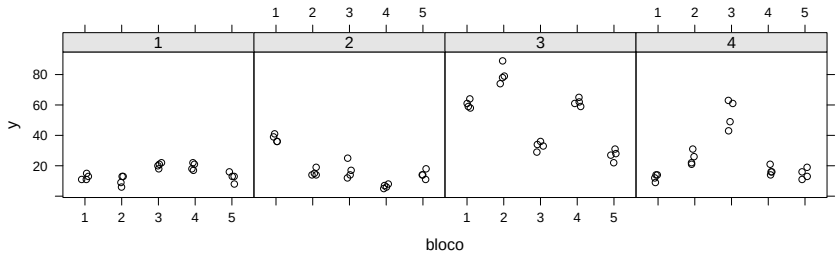

Figura 4.5: Análise descritiva modelo com efeito aninhado.

Código 4.70: 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 $21^6 = 85766121$ vezes, o que é 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 4.67 para o caso do Modelo Poisson com efeito aninhado usando apenas a integração por Laplace.

Código 4.71: 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 4.72: 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     3.08991    0.23763 13.0029 < 2.2e-16
sigma  0.87431    0.43609  2.0049 0.0449776
tau    0.69383    0.18578  3.7348 0.0001879

-2 log L: 554.7749
```

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
2.397217 2.001369
```

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 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    2.62416308 3.555667
sigma 0.01958071 1.729036
tau   0.32971773 1.057945
```

note que a saída acima apresenta os intervalos de confiança para os parâmetros de variância reparametrizados. 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] 0.3482492 4.4461843
```

```
ic.tau
```

```
[1] 1.272643 2.730096
```

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 <- numDeriv::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    0.0564693597  0.001143706  0.0003406432
sigma 0.0011437058  0.190177598 -0.0069385315
tau   0.0003406432 -0.006938531  0.0345126480
```

```
solve(Io)     ## Richardson
```

```
           [,1]         [,2]          [,3]
[1,] 0.0564693597  0.001143706  0.0003406432
[2,] 0.0011437058  0.190177598 -0.0069385315
[3,] 0.0003406432 -0.006938531  0.0345126480
```

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] 2.852282 3.327548
```

```
ic.sigma.H
```

```
[1] 0.3482492 4.4461843
```

```
ic.tau.H
```

```
[1] 1.272643 2.730096
```

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.

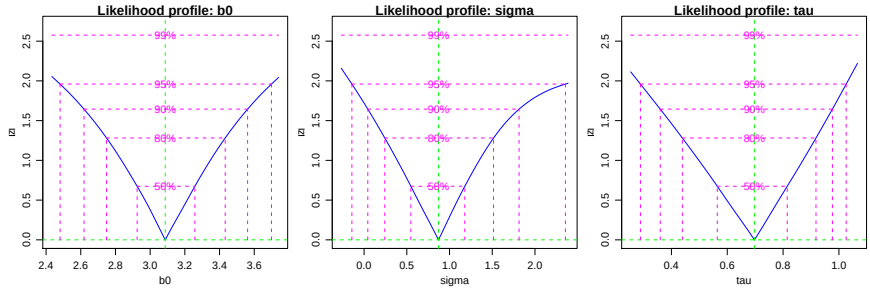

Figura 4.6: Perfil de verossimilhança - Modelo Poisson com efeito aninhado.

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     2.852282 3.327548 0.3482492  4.446184 1.272643 2.730096
perfil.b0 2.480810 3.699131 0.8672940 10.548728 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.

4.6 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 adicionamos 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 4.73: 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 4.7 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 (4.73).

```
dados <- rbeta.model(ID = 10, tempo = 15, beta.fixo = c(0.5, 0.8), prec.pars=c(
```

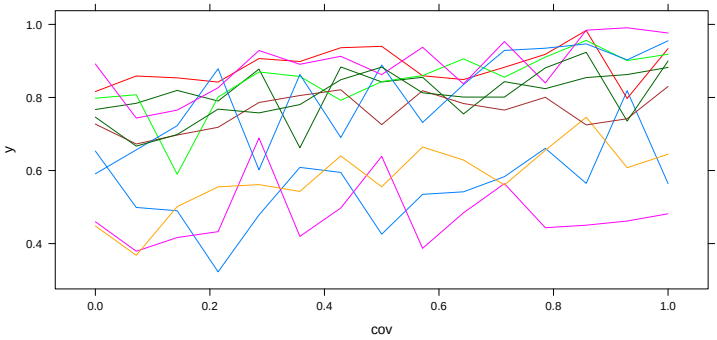

Figura 4.7: Trajetórias por unidade amostral - Modelo Beta longitudinal.

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 4.74: 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 4.67, podemos colocar o modelo da forma apropriada para utilizar a função `mle2()`.

Código 4.75: 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 da 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      0.71712    0.19654  3.6488 0.0002635
b1      0.99116    0.18803  5.2712 1.355e-07
sigmaI -0.52810    0.24664 -2.1412 0.0322565
sigmaS -0.74294    0.34620 -2.1460 0.0318745
rho     1.29939    1.10523  1.1757 0.2397245
phi     3.54610    0.12361 28.6889 < 2.2e-16

-2 log L: -334.7962
```

Os intervalos de confiança podem ser obtidos na escala reparametrizada. Usamos os resultados dos Teoremas 2.4 e 2.5 para obter intervalos aproximados na escala original dos parâmetros.

```
confint(ajuste, method="quad")
```

```
           2.5 %      97.5 %
b0      0.3319139  1.10232406
b1      0.6226191  1.35969605
sigmaI -1.0115037 -0.04470382
sigmaS -1.4214754 -0.06439914
rho    -0.8668164  3.46559315
phi     3.3038397  3.78836338
```

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 diretamente a função de reparametrização aos limites do intervalo. Isto é justificado pela propriedade de invariância da verossimilhança. A Figura 4.8 apresenta os perfis de verossimilhança dos parâmetros do modelo Beta longitudinal.

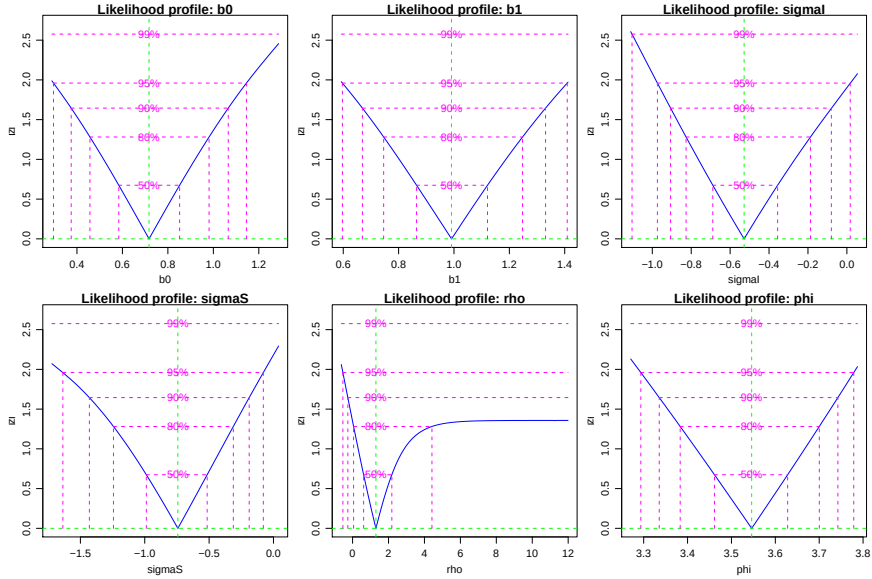

Figura 4.8: Perfil de verossimilhança - Modelo Beta longitudinal.

4.7 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:

$$P(Y_{ij}|\theta_j) = (1 - c_i) + \frac{c_i}{1 + \exp\{-\alpha_i(\theta_j - \beta_i)\}}.$$

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 4.76: 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 4.77: 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 4.78: 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 4.79: 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**.

```
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

-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
```

4.8 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,

$$\begin{aligned} y_t &= \theta_t + v_t, \quad v_t \sim N(0, V) \\ \theta_t - \mu &= \phi(\theta_{t-1} - \mu) + w_t, \quad w_t \sim N(0, W) \end{aligned} \tag{4.13}$$

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

$$\begin{aligned} y_t &= F_t\theta_t + v_t \\ (\theta_t - \mu) &= G_t(\theta_t - \mu) + w_t. \end{aligned} \tag{4.14}$$

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.

4.8.1 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. Para obter a verossimilhança, basta conhecer as médias e variâncias condicionais de $[y_t| \dots]$, que dependem de ψ (Schweppe, 1965). 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. Esta abordagem é explorada por diversos autores como, por exemplo, Akaike (1978), Harvey & Phillips (1979) Jones (1980), Gardner et al. (1980) e Harvey (1981).

O filtro de Kalman, 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:

$$\theta_t^{t-1} = G\theta_{t-1}^{t-1} \tag{4.15}$$

$$P_t^{t-1} = GP_{t-1}^{t-1}G^{'} + W \tag{4.16}$$

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:

$$\begin{aligned} e_t &= y_t - F_t\theta_t^{t-1} \\ Q_t &= F_tP_t^{t-1}F_t^{'} + V \\ K_t &= P_t^{t-1}F_t^{'}Q_t^{-1} \\ \theta_t^t &= \theta_t^{t-1} + K_te_t \\ P_t^t &= P_t^{t-1} - K_tF_tP_t^{t-1} \end{aligned}$$

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:

$$J_{t-1} = P_{t-1}^{t-1}G^{'}(P_t^{t-1})^{-1} \tag{4.17}$$

$$\theta_{t-1}^n = \theta_{t-1}^{t-1} + J_{t-1}(\theta_t^n - \theta_t^{t-1}) \tag{4.18}$$

$$P_{t-1}^n = P_{t-1}^{t-1} + J_{t-1}(P_t^n - P_t^{t-1})J_{t-1}^{'} \tag{4.19}$$

Com e_t e Q_t obtidos no passo de filtragem do filtro de Kalman e considerando que ambos dependem de ψ, temos

$$l(\psi) = -\frac{1}{2}\sum_{t=1}^{n}\log|Q_t| - \frac{1}{2}\sum_{t=1}^{n}e_t^{'}Q_t^{-1}e_t. \tag{4.20}$$

4.8.2 Um exemplo simples

Vamos considerar o caso mais simples, descrito no início desta seção, considerando $\mu = 0$. Neste caso, temos um intercepto que evolui dinamicamente. São três os parâmetros deste modelo: 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 em R contendo: (i) as informações básicas sobre a dimensão da série n, m e k, (ii) elementos y, F, x, V, W e G, que representam os componentes do modelo y_t, F_t, θ_t, V, W e G, respectivamente. 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, armazenamos 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 4.80: Estrutura para o modelo com intercepto dinâmico.

```
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:

Considerando que temos y_t univariado, a função para calcular o negativo da log-verossimilhança pode ser simplesmente:

Código 4.81: 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])
}
```

```
nlldlm1 <- function(dlm) -sum(dnorm(dlm$e, 0.0, sqrt(dlm$S), log=TRUE))
```

A maximização 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.

Código 4.82: Função de verossimilhança do modelo com intercepto dinâmico.

```
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)
  return(nlldlm1(dlm))
}
```

Agora, vamos definir iniciais e usar a função `optim()` para encontrar as estimativas. Vamos usar o método `L-BFGS-B` que considera restrição no espaço paramétrico definida 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 4.83: 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 })
  }
  return(dlm) # retorna componentes x.s e P.s atualizados

}
```

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, 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)))
```

Na Figura 4.9 podemos visualizar os valores de θ_t simulados e estimados 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))
```

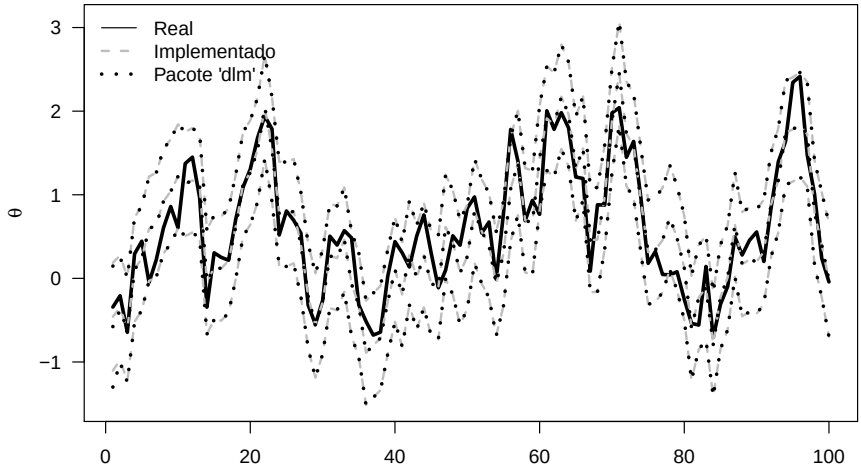

Figura 4.9: Estado latente simulado e estimado e respectivos intervalos de confiança obtidos pelas funções implementadas e as do pacote **dlm**.

4.8.3 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
### 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 no código 4.84 uma função que crie uma lista contendo as componentes do modelo. Esta lista é semelhante à criada no exemplo anterior, porém, para o contexto de regressão dinâmica. No código 4.85 definimos uma função para a filtragem.

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 4.84: 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)
}
```

Código 4.85: 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])
}
```

Código 4.86: 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 4.17, 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 para 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 função no código 4.87.

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)
```

Código 4.87: 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])
}
```

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 0.1636 0.1972 0.1099 0.7995 0.6412 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 4.10, 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 acrescentaremos 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.

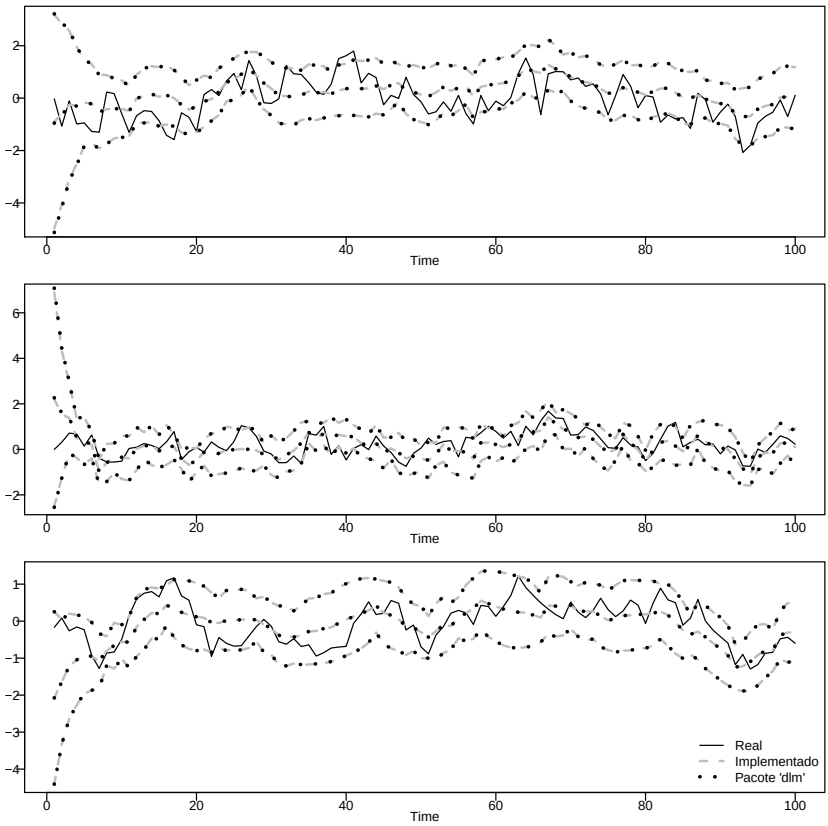

Figura 4.10: Coeficientes de regressão simulados, e estimados (e respectivos intervalos de confiança) via funções implementadas e via funções do pacote 'dlm'.

Capítulo 5

Inferência Bayesiana

A inferência estatística pode ser conduzida sob diferentes abordagens. Diferentes autores podem adotar critérios distintos nesta divisão. Neste material vamos distinguir simplesmente entre procedimentos que (i) usam apenas a função de verossimilhança (ou alguma aproximação desta) para inferência, ou seja, toda a informação sobre os parâmetros utilizada para inferência provém exclusivamente da verossimilhança e (ii) métodos que incorporam informações externas à verossimilhança. No segundo caso vamos considerar mais especificamente o procedimento bayesiano que incorpora informações sobre o parâmetro externas à amostra associando a estes uma distribuição de probabilidades (distribuição a *priori*, ou simplesmente *priori*). Para uma introdução a este tema recomendamos Box & Tiao (1992). Desta forma a evidência sobre um parâmetro é a contida na verossimilhança, ponderada para os possíveis valores dos parâmetros, com pesos dados pela *priori*.

A partir deste ponto passamos a tratar da abordagem bayesiana. Vantagens e desvantagens podem ser atribuídas a ambas, assim como a variações e extensões. Neste texto, não estamos interessados em nos aprofundarmos neste debate ou em discutir ou advogar os méritos de cada uma delas. Nosso objetivo é simplesmente ilustrar implementação para diferentes modelos estatísticos.

Métodos de inferência bayesianos são em geral implementados através de métodos computacionalmente intensivos e diversos *softwares* para este fim estão disponíveis. A ideia desta parte do livro é apresentar os conceitos básicos de inferência Bayesiana, e na maioria dos casos implementa-se inferência para os modelos via *Métodos de Monte Carlo via Cadeias de Markov* (*MCMC*). Reanalisamos alguns dos modelos que foram implementados via verossimilhança nos capítulos anteriores. Neste texto, vamos dar ênfase ao

programa JAGS (*Just Another Gibbs Sampler*) que pode ser usado de forma independente mas que possui uma interface que possibilita seu uso a partir do R através do pacote **rjags** ou do pacote **dclone**.

Por vezes utilizamos outros pacotes residentes no R como o **MCMCpack**, que implementa alguns modelos, e o **coda**, para visualização das cadeias. Outras opções de implementação direta em R ou de interfaces com outros programas são descritas na página da *Bayesian Task View*[1]. Uma nova ferramente é o pacote **LaplacesDemon** que apresenta uma grande quantidade de algoritimos MCMC, porém não exploramos este pacote no decorrer do livro.

Em inferência Bayesiana, a escolha das distribuições a *priori* é um tema de grande relevância. Quando possível optamos por adotar *priori's* vagas e próprias. Não vamos nos preocupar aqui com a busca de *priori's* conjugadas, *não informativas* (em algum sentido) ou de referência, tais como *priori's* de Jeffreys. Este tipo de *priori* são particulares a classes de modelos e aqui estamos interessados em algoritmos gerais para apresentar as ideias e não em especificidades de classes particulares de modelos. É claro que com isto sacrificamos a eficiência da implementação computacional.

5.1 Inferência Bayesiana

A inferência estatística é uma ferramenta cujo objetivo principal é fornecer subsídios, de forma quantitativa, para a tomada de decisões. Isso é feito a partir de dados coletados, quer sejam em estudos amostrais, experimentais ou observacionais. A inferência estatística Bayesiana, ou simplesmente inferência Bayesiana (IB), além da informação contida nos dados, também considera informações prévias à coleta dos dados. A informação prévia, sobre os parâmetros, é chamada de informação à *priori*. Ambas as informações, à *priori* e dos dados, são combinadas usando o teorema de Bayes, gerando a informação à *posteriori* sobre os parâmetros. O teorema de Bayes fornece uma expressão para a probabilidade condicional de A dado B, que é na sua notação mais simples expresso por:

Teorema 5.1 (Teorema de Bayes)**.**

$$P(A|B) = \frac{P(B|A)P(A)}{P(A)}$$

A construção de modelos baseados em inferência Bayesiana usa o teorema de Bayes combinando dados e informação a *priori* sobre os parâmetros do modelo. Na equação 5.1 troque B pelas observações $\underline{y}$, A pelo conjunto

[1] *http://cran.r-project.org/web/views/Bayesian.html*

de parâmetros $\underline{\theta}$ e $P(\cdot)$ indistintamente por funções de probabilidade ou densidade probabilidade, e denote $[\cdot]$ por a distribuição do termo entre colchetes, resultando em:

$$[\underline{\theta}|\underline{y}] = \frac{[\underline{y}|\underline{\theta}]\ [\underline{\theta}]}{[\underline{y}]}$$

em que $[\underline{y}]$ será discutida na sequência, $[\underline{\theta}]$ é a distribuição conjunta *a priori* para o vetor de parâmetros $\underline{\theta}$, definida antes de "observar" (ou independentemente de) $\underline{y}$. O modelo adotado determina a expressão de $[\underline{y}|\underline{\theta}]$, a verossimilhança de $\underline{y}$ e $[\underline{\theta}|\underline{y}]$ é a distribuição a *posteriori* (ou simplesmente *posteriori*) conjunta para o vetor de parâmetros $\underline{\theta}$ que expressa a informação (incerteza) sobre o conjunto de parâmetros após combinar a informação dos dados (pela verossimilhança) e a informação *a priori*. O denominador

$$[\underline{y}] = \int [\underline{y}|\underline{\theta}][\underline{\theta}]d\underline{\theta}$$

define a verossimilhança marginal $\underline{y}$, análoga à definida no capítulo 4 em modelos com efeitos aleatórios. Note novamente o problema da integral contida nesta função. O interessante no caso da inferência Bayesiana é que em certos algoritmos como os de MCMC, esta quantidade não precisa ser calculada para fins de inferência sobre os parâmetros do modelo. Como não dependendo dos parâmetros, pode simplesmente ser denotada como uma constante de proporcionalidade c. Desta forma, a *posteriori* de interesse torna-se,

$$[\underline{\theta}|\underline{y}] \propto [\underline{y}|\underline{\theta}][\underline{\theta}]$$

Esta forma pode ser vista como a distribuição conjunta dos parâmetros não normalizada, dada pelo produto da verossimilhança com a *priori*. A *posteriori* de interesse na grande maioria das vezes não tem expressão fechada, o que torna necessário métodos para aproximação desta distribuição, sejam por aproximações analíticas e ou numéricas ou por simulação estocástica. Neste último caso infere-se sobre a *posteriori* a partir de simulações de seus valores. Dentre estes métodos os algoritmos mais utilizados são o amostrador de Gibbs e o algoritmo de Metropolis Hastings que fazem parte da *suite* de algorítimos sob o nome genérico de MCMC. Está fora do escopo e objetivos deste texto, discutir sobre a construção desta grande classe de algoritmos. Vamos apenas ilustrar implementações simples e utilizar implementações já existentes, para exemplificar a aplicação dos métodos de inferência em alguns modelos.

Resumindo, inferência Bayesiana tem três componentes, são eles:

1. $[\underline{\theta}]$ é a distribuição conjunta a *priori* para o vetor de parâmetros $\underline{\theta}$, usando portanto distribuição de probabilidades sobre os parâmetros

como uma forma de refletir a iformação (incerteza/ignorância) sobre $\underline{\theta}$ antes de coletar os dados;

2. $[\underline{y}|\underline{\theta}]$ é a verossimilhança, que contém toda a informação sobre $\underline{\theta}$ proveniente dos dados $\underline{y}$ a ser utilizada;

3. $[\underline{\theta}|\underline{y}]$ é a distribuição conjunta a *posteriori* que expressa a informação (incerteza) sobre o vetor de parâmetros após combinar a informação a *priori* e a contida na verossimilhança.

Nas próximas Seções mostramos os fundamentos de alguns procedimentos associados a procedimentos bayesianos. Na Seção 5.2 exploramos um exemplo de elicitação de *priori*'s enquanto que em 5.3 apresentamos operações básicas com a *posteriori*. Inferência por simulação Monte Carlo é discutida em 5.4 e por Monte Carlo via cadeias de Markov (MCMC) em 5.5. Nas demais seções revisitamos e implementamos inferência bayesiana para alguns dos modelos já apresentados nos capítulos anteriores. Vamos usar duas implementações computacionais de algoritmos MCMC genéricos. A primeira e mais simples é a função `MCMCMetrop1R()` implementada no pacote **MCMCpack** que traz um algoritmo de Metropolis usando *proposal* gaussiana. A segunda é o programa *JAGS - Just Another Gibbs Sample* junto com os pacotes **rjags** e **dclone** que entre outras coisas possuem uma interface para usar o **JAGS** diretamente a partir do R. De acordo com a complexidade do modelo usaremos a ferramenta de software que julgarmos mais adequada para ilustração.

5.2 Elicitação de uma distribuição *a priori*

Os primeiros passos para fazer inferência bayesiana são a especificação do modelo probabilístico para os dados e a atribuição de uma distribuição de probabilidades para os parâmetros desse modelo, ou seja, a elicitação da *priori*. O uso de distribuições *a priori* hierárquicas, ou hiper-prioris é também comum na prática. Isto é, considera-se os parâmetros das distribuições *a priori* como hiper-parâmetros e atribui-se distribuições também para os hiper-parâmetros. Nesta seção, nós vamos considerar um exemplo de elicitação de *priori*.

5.2.1 Exemplo: *priori* para uma proporção

Um pesquisador quer mais informações sobre um parâmetro θ que assume valores no intervalo $(0, 1)$. Para isso, ele coleta dados e pede para um estatístico analisar. O estatístico quer saber, antes de proceder a análise dos

dados, o que o pesquisador sabe *a priori* sobre esse parâmetro. Após discussões, o estatístico representa as informações do pesquisador acerca do parâmetro na Tabela 5.1.

Tabela 5.1: Informação fornecida pelo pesquisador.

	Prob. pesquisador
$0 < \theta < 0.05$	0.0500
$0.05 < \theta < 0.10$	0.3500
$0.1 < \theta < 0.3$	0.5000
$0.3 < \theta < 0.5$	0.0900
$0.5 < \theta < 1$	0.0100

Para proceder com a inferência bayesiana, o estatístico precisa representar esta informação através de um modelo probabilístico. Para resolver esse problema, vamos considerar que uma distribuição $Beta(a,b)$ é adequada para modelar o conhecimento sobre θ com densidade na forma:

$$f(\theta|a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^a(1-\theta)^b. \tag{5.1}$$

O problema então é encontrar os parâmetros (a,b) da distribuição beta que melhor descrevam a informação fornecida pelo pesquisador. Portanto, deseja-se encontrar uma densidade beta que forneça probabilidades próximas às probabilidades provenientes da crença do pesquisador. Esse problema resume-se à buscar em um espaço bi-dimensional, a e b, parâmetros de uma densidade beta. Vamos considerar duas soluções:

1. buscar a solução que minimize a soma da diferença quadrática entre cada uma das probabilidades provenientes do pesquisador e as respectivas probabilidades obtidas pela densidade beta;

2. buscar a solução que minimize a soma da diferença relativa (elevada ao quadrado) de cada uma das probabilidades dadas pelo pesquisador e as respectivas probabilidades obtidas pela densidade beta.

Em R, podemos fazer:

```
breaks <- c(0, 0.05, 0.1, 0.3, 0.5, 1) # limites das classes
p.pesq <- c(.05, .35, .5, .09, .01) ### probs atribuidas
 fun <- function(pars) ### funcão p/ diferença quadrática
  sum((diff(pbeta(breaks, pars[1], pars[2])) - p.pesq)^2)
### "a" e "b" que minimizam a diferença quadrática
(res <- optim(c(1,1), fun, lower=rep(1e-5, 2), method='L')$par)
[1]  4.240215 33.920425
funr <- function(pars) ### funcão p/ diferença relativa ao quadrado
  sum((diff(pbeta(breaks, pars[1], pars[2]))/p.pesq-1)^2)
### "a" e "b" que minimizam o quadrado da diferença relativa
(resr <- optim(c(1,1), funr, lower=rep(1e-5,2), method='L')$par)
```

```
[1]  2.388608 10.672337
```

A solução encontrada é $a = 4.24$ e $b = 33.92$, considerando diferença quadrática e $a = 2.389$ e $b = 10.67$, considerando diferença relativa ao quadrado. Na Tabela 5.2 são mostradas as probabilidades segundo a distribuição beta atribuídas às classes segundo parâmetros obtidos com cada critério. O modelo 1 é aquele que minimiza a diferença quadrática entre as probabilidades, enquanto o modelo 2 minimiza a diferença relativa ao quadrado. Para o modelo 1, notamos que a aproximação não é muito boa nas classes com menores probabilidades, mas é boa nas classes com probabilidades maiores. Para o modelo 2, o ajuste é melhor nas probabilidades pequenas.

Tabela 5.2: Probabilidades fornecida pelo pesquisador e pelos modelos

	Prob. pesquisador	Prob. mod. 1	Prob. mod. 2
$0 < \theta < 0.05$	0.0500	0.0879	0.0623
$0.05 < \theta < 0.10$	0.3500	0.3769	0.1697
$0.1 < \theta < 0.3$	0.5000	0.5333	0.6319
$0.3 < \theta < 0.5$	0.0900	0.0019	0.1295
$0.5 < \theta < 1$	0.0100	0.0000	0.0066

Na Figura 5.1, as áreas das barras no histograma correspondem às probabilidades provenientes do pesquisador e as densidades dos dois modelos beta encontrados. Com isto pode-se consultar o pesquisador sobre qual das duas opções melhor reflete a crença sobre o parâmetro.

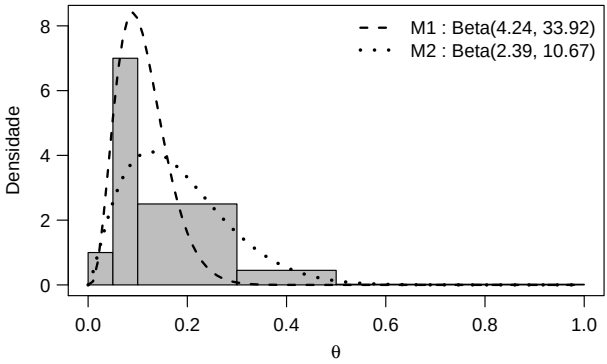

Figura 5.1: Histograma da informação fornecida pelo pesquisador e curva das duas distribuições betas consideradas.

Neste exemplo, procurou-se fornecer ao pesquisador uma representação da sua crença sobre o parâmetro usando o modelo beta e esta escolha não é casual. A beta é uma distribuição conveniente por pelo menos três razões.

Primeiro porque seu intervalo de variação coincide com o do parâmetro. Além disto, se os dados coletados tiverem distribuição Bernoulli, binomial ou binomial negativa, usando uma distribuição beta como *priori* obtém-se a distribuição *a posteriori* em forma fechada. Finalmente, nesses casos, a *posteriori* é também uma distribuição beta e estamos diante de um caso de *conjugação*, ou seja, *priori* e *posteriori* possuem distribuições na mesma família com apenas valores diferentes para os parâmetros (a, b). A computação neste caso é trivial, bastante atualizar os valores dos parâmetros na posteriori e a atualização é dada pela verossimilhança.

Suponha que o pesquisador considere as distribuições beta inadequadas para representar o seu conhecimento a *priori* e que deseje que o conhecimento a *priori* seja representado pelas probabilidades apresentadas na Tabela 5.1. Então, precisamos encontrar a distribuição a posteriori considerando essas probabilidades. Neste caso a posteriori não será conjugada e sequer possui forma fechada e a inferência será mais trabalhosa. Voltaremos a esta possibilidade mais adiante mas por agora vamos prosseguir com os resultados para *priori* beta.

5.3 Distribuição a posteriori

Vamos ilustrar o cálculo da distribuição *a posteriori*, através de um exemplo. Considere o caso em que o pesquisador coletou dados e que esses dados assumem os valores zero ou um. Neste caso, fixado o número de amostras, podemos considerar um modelo de Bernoulli. Ou seja, para a amostra aleatória $\underline{y} = \{y_1, y_2, \ldots, y_n\}$ temos que $y_i \in \{0,1\}$, $i = 1, 2, \ldots, n$, e que a densidade $Bernoulli(\theta)$, $0 < \theta < 1$ pode ser considerada para y_i,

$$f(y_i|\theta) = \theta_i^y (1-\theta)^{1-y_i}.$$

Queremos fazer inferência sobre o parâmetro θ. No exemplo tentou-se descrever o conhecimento *a priori* do pesquisador usando uma densidade da família beta (5.1). A informação dos dados é representada pela verossimilhança de θ,

$$L(\theta|\underline{y}) = \theta^{sy}(1-\theta)^{n-sy} \ ,$$

em que $sy = \sum_{i=1}^{n} y_i$. Consequentemente, a distribuição a posteriori é

$$f(\theta|\underline{y},a,b) = \frac{\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^{a+sy}(1-\theta)^{(b+n-sy)}}{C}$$

com

$$\begin{aligned} C &= \int_0^1 \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^{a+sy}(1-\theta)^{(b+n-sy)}d\theta \\ &= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\int_0^1 \theta^{a+sy}(1-\theta)^{(b+n-sy)}d\theta \\ &= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\frac{\Gamma(a+sy)\Gamma(b+n-sy)}{\Gamma(a+b+n)} \end{aligned} \tag{5.2}$$

portanto,

$$f(\theta|\underline{y},a,b) = \frac{\Gamma(a+b+n)}{\Gamma(a+sy)\Gamma(b+n-sy)}\theta^{a+sy}(1-\theta)^{(b+n-sy)}$$

ou seja, $[\theta|\underline{y}] \sim Beta(a_n, b_n)$ com $a_n = a + sy$ e $b_n = b + n - sy$.

Neste exemplo, é possível identificar a distribuição a *posteriori* apenas observando o seu "núcleo" $\theta^{a+sy}(1-\theta)^{b+n-sy}$, que são termos que dependem de θ que são o núcleo de uma densidade $Beta(a_n, b_n)$.

Obtida a distribuição a *posteriori* de um parâmetro, temos toda a informação disponível, (*a priori* e trazida pelos dados) representada por uma distribuição de probabilidades. Assim, podemos calcular resumos descritivos ou probabilísticas sobre o parâmetro, ou simplesmente visualizar essa distribuição.

Vamos supor que o pesquisador prefere adotar a distribuição *a priori* beta encontrada minimizando as diferenças relativas da área a baixo da curva com a informação da Tabela 5.1, identificada como `modelo 2` na Figura 5.1. Consideremos que num total de $n = 30$ amostras coletadas pelo pesquisador, a soma foi $sy = 7$. A distribuição a posteriori de θ pode ser vista na Figura 5.2, juntamente com a distribuição *priori* beta considerada e a informação *a priori* do pesquisador colocada na forma de histograma.

Também podemos calcular a média dada por $a_n/(a_n + b_n)$, variância dada por $a_n b_n/((a_n + b_n)^2(a_n + b_n + 1)$ e probabilidades sobre θ.

```
a.n <- resr[1]+7
b.n <- resr[2]+30-7
## média a posteriori
a.n/(a.n+b.n)
```

```
[1] 0.2180307
```

```
## mediana
qbeta(0.5, a.n, b.n)
```

```
[1] 0.2136373
```

```
## desvio padro a posteriori
sqrt((a.n*b.n)/((a.n+b.n)^2*(a.n+b.n+1)))
```

```
[1] 0.06220521
```

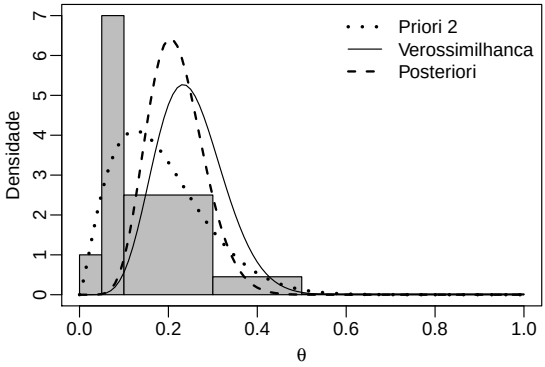

Figura 5.2: Histograma da informação fornecida pelo pesquisador, distribuição a priori considerada, verossimilhança (escalonada) e distribuição a posteriori.

```
## Prob(theta<0.1)
pbeta(0.1, a.n, b.n)
```

```
[1] 0.01430607
```

```
## Intervalo de Credibilidade (quantis) de 95%
qbeta(c(0.025, 0.975), a.n, b.n)
```

```
[1] 0.1095469 0.3512603
```

Em diferentes problemas a distribuição *a posteriori* pode ou não possuir a forma de uma densidade de probabilidades conhecida. No exemplo isto poderia se o pesquisador adotasse as densidades *a priori* como as apresentadas e optasse por adotar alguma outra para qual não é possível deduzir a forma analítica para a *posteriori*. De forma mais geral, há modelos para os quais não há *prioris* que levem a *posteriori* em forma analítica fechada, conjugadas ou não.

Na próxima seção, vamos descrever alguns algoritmos comumente utilizados para obter aproximações da distribuição a *posteriori* quando esta não possui forma conhecida. Entre as ilustrações, revisitamos o exemplo apresentado aqui adotando a *priori* na forma apresentada na Tabela 5.1.

5.4 Aproximações baseadas simulação de Monte Carlo

Os métodos descritos nesta seção, aplicam-se na situação em que a distribuição à *posteriori* não possui forma conhecida. Nesta seção apresentaremos rapidamente os algoritmos mais comuns de simulação não sequencial, isto é, simulações de Monte Carlo (MC).

5.4.1 Método de Monte Carlo Simples

O método de Monte Carlo é uma classe de algoritmos de simulação não sequencial, isto é, cada valor simulado é feito de forma independente dos anteriores. Neste caso, as amostras simuladas são independentes entre si.

Inicialmente, vamos considerar que deseja-se calcular alguma quantidade que possa ser expressa como:

$$I = \int_a^b g(\theta) d\theta$$

e que não podemos obter o resultado analiticamente. Se os valores de θ estão confinados a um intervalo $[a,b]$ podemos reescrever essa expressão como

$$I = \int_a^b (b-a) g(\theta) \frac{1}{b-a} d\theta = (b-a) \int_a^b g(\theta) \frac{1}{b-a} d\theta = (b-a) E[g(\theta)].$$

em que $I(g(\theta))$ é uma esperança calculada considerando uma distribuição Uniforme(a, b).

Portanto, o problema se resume ao cálculo de uma esperança que pode ser "estimada" a partir de uma amostra $\{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)}\}$, simulada de uma distribuição $U(a,b)$. Portanto, podemos estimar I usando o estimador de média

$$\hat{I} = \frac{(b-a)}{N} \sum_{i-1}^{N} g(\theta^{(i)})$$

que é não viesado. Vamos considerar o exemplo da seção anterior, em que temos uma distribuição à *posteriori* $Beta(a_n, b_n)$ para θ. Suponha que queremos calcular a probabilidade de $P[0{,}3 < \theta < 0{,}5] = \int_a^b g(\theta) d\theta$ e neste caso $g(\theta)$ é esta distribuição beta. Por integração de Monte Carlo simulamos valores θ_i de uma distribuição $U(0{,}3; 0{,}5)$, avaliamos para cada um deles o valor $g(\theta_i)$ da densidade $\beta(a_n, b_n)$ e estimamos I pela multiplicação da amplitude do intervalo pela média $\overline{g}$ destes valores.

```
set.seed(1); N <- 10000
u <- runif(N, 0.3, 0.5)
g.th <- dbeta(u, a.n, b.n)
mean(g.th)*(0.5 - 0.3)
```

```
[1] 0.1026874
```

Neste caso podemos obter o valor exato da integral, que é

```
pbeta(0.5, a.n, b.n) - pbeta(0.3, a.n, b.n)
```

```
[1] 0.1018369
```

Este método que descrevemos e exemplificamos é chamado de método de Monte Carlo simples. A seguir, passamos a algoritmos que permitem obter uma amostra da distribuição *posteriori* desejada, $[\theta|y]$. As técnicas de simulação são de grande importância quando essa distribuição não possui forma conhecida.

5.4.2 Amostragem por rejeição

O método da amostragem por rejeição aplica-se na situação em que não é possível (ou não sabemos) simular diretamente da distribuição de interesse. Ele consiste em retirar amostras de uma distribuição $g(\theta)$ que tenha o mesmo suporte que a distribuição de θ, $p(\theta)$. Porém, nem todos os valores simulados são aceitos e usa-se uma regra de aceitação de tal forma que os valores aceitos forneçam uma amostra de $p(\theta)$. Os valores são aceitos com uma probabilidade igual a $p(\theta)/(Mg(\theta))$, onde M é maior ou igual ao máximo de $g(\theta)$ em relação à θ. Portanto a exigência aqui é a de que $p(\theta) \leq Mg(\theta)$ para todo θ.

Para o caso do exemplo da seção anterior, simulamos uma amostra de θ, $\{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)}\}$. Com essa amostra, podemos, por exemplo, aproximar a probabilidade de $\theta \in (0{,}3; 0{,}5)$ pela proporção de $\theta's$ simulados pertencentes a esse intervalo.

Precisamos encontrar uma densidade $g(\theta)$ da qual seja fácil de simular valores e M tal que

$$\frac{p(\theta)}{g(\theta)} \leq M\,, \quad \forall \theta. \tag{5.3}$$

Consideremos $g(\theta) = U(0{,}1)$. Neste caso $g(\theta) = 1 \ \forall \ \theta \in (0,1)$. Como conhecemos $p(\theta)$, a distribuição de interesse, sabemos encontrar seu valor máximo. Basta visualizar $p(\theta)$, ou encontrar seu máximo explicitamente:

```
(M <- exp(-optimize(function(x) -dbeta(x, a.n, b.n, log=TRUE),
             c(0,1))$objective))
```

```
[1] 6.426808
```

Assim, tomando por exemplo, $M = 6.5$, garantimos 5.4.2. Entretanto, nem sempre é simples encontrar uma função $g(\theta)$ e M satisfatórios.

No código 5.88 definimos uma função para implementar o método da rejeição para o exemplo. As amostras são sorteadas de uma $U[0{,}1]$ e aceitas com probabilidade $p(\theta_i)/(Mg(\theta_i))$ até que o número de amostras especificado seja atingido. Não guardamos as amostras rejeitadas mas, para ilustração contamos em `des` o número de rejeitadas.

Código 5.88: Função para implementar o método da rejeição no exemplo considerado.

```
rej01u <- function(nsim=10000, M) {
  res <- double(nsim)
  des <- 0             ## contador da amostras rejeitadas
  for (i in 1:nsim){
     repeat {
      cand <- runif(1)
      p <- dbeta(cand, a.n, b.n)/M
      if (runif(1)<p) break
      else des <- des+1 ## contador da amostras rejeitadas
     }
    res[i] <- cand
   }
   attr(res, "descartadas") <- des
  return(res)
}
```

```
set.seed(1)
am.rj <- rej01u(N, M=M)
attributes(am.rj)$descartadas
```

```
[1] 54474
```

No gráfico à esquerda de Figura 5.3 vemos as densidade objetivo e a $g(\cdot)$ uniforme (escalonada por M) da qual as amostras são retiradas. Como se vê há uma grande área na uniforme escalonada que não está sob $p(\cdot)$, o que explica a alta taxa de rejeição. Na figura do centro são mostrados o histograma da amostra obtida de $p(\theta)$ junto à curva sobreposta que é a densidade verdadeira que neste exemplo ilustrativo é conhecida.

A partir de uma amostra da distribuição a *posteriori*, inferências sobre os parâmetros são obtidas por estatísticas descritivas da amostra. Por exemplo, podemos aproximar probabilidades sobre θ, medidas resumo e intervalos de credibilidade. Por exemplo, a probabilidade $P(0{,}3 < \theta < 0{,}5)$ é estimada simplesmente contando-se quantos valores da amostra estão neste intervalo.

```
mean(am.rj>0.3 & am.rj<0.5)                    ## estimada/amostra
```

```
[1] 0.1005
```

```
pbeta(0.5, a.n, b.n) - pbeta(0.3, a.n, b.n) ## exata
```

```
[1] 0.1018369
```

e o intervalo de 95% de credibilidade baseado em quantis:

```
quantile(am.rj, c(0.025, 0.975))   ## estimado/amostra
```

```
    2.5%     97.5%
0.1089951 0.3487988
```

```
qbeta(c(0.025, 0.975), a.n, b.n) ## exato
```

```
[1] 0.1095738 0.3513049
```

Os valores não são exatamente iguais aos verdadeiros mas aproximam-se a medida que se aumenta o número de amostras simuladas da distribuição a *posteriori*.

O intervalo obtido a partir da simulação é simplesmente dado pelos quantis da amostra. Há outras possibilidades como os intervalos baseados em regiões de maior densidade (intervalos *HPD*) que possuem melhores propriedades e serão utilizados mais adiante.

5.4.3 Amostragem por importância

A amostragem por função de importância é uma alternativa ao método da rejeição quando não é possível determinar um valor razoável M. A essência do algoritmo é amostrar valores segundo pesos que são computados para cada um deles. Neste algoritmo, também necessitamos de uma distribuição $g(\theta)$ de referência, da qual seja fácil de simular.

A ideia é simular $\{\tilde{\theta}^{(1)}, \tilde{\theta}^{(2)}, \ldots, \tilde{\theta}^{(kN)}\}$ de $g(\theta)$, ou seja, simular de $g(\theta)$, k vezes o número de amostras desejado de $p(\theta)$ e calcular os pesos

$$w_i = \frac{p(\tilde{\theta}^{(i)})}{g(\tilde{\theta}^{(i)})}.$$

Os pesos w_i podem ser maiores que 1 e portanto a probabilidade de aceitação é

$$p_i = \frac{w_i}{\sum_{i=1}^{kN} w_i}.$$

A partir dessas probabilidades, seleciona-se N amostras, com ou sem reposição, dessas kN simuladas. Recomenda-se usar $k = 20$. Uma desvantagem deste método é a necessidade de simular kN amostras previamente e descartar a maioria delas. Outra é a necessidade de armazenar a 'população' de kN valores.

Em alguns casos esta dificuldade pode ser contornada através de uma ***discretização*** do suporte de θ. Suponhamos $\theta \in (\alpha_1, \alpha_m)$. Seja a sequencia $\alpha_1, \alpha_2, \ldots, \alpha_m$, com $\theta_i - \theta_{i-1} = 1/m$, $i = 2, 3, \ldots, m$. Como esta sequencia é um malha regular de valores, podemos definir como probabilidade de seleção de α_i,

$$p_{\alpha_i} = \frac{p(\alpha_i)}{\sum_{i=1}^{m} p(\alpha_i)}$$

e selecionar com reposição os elementos dessa sequencia com essas probabilidades. Neste caso, é adequando escolher m grande o suficiente.

Considerando o exemplo, vamos simular uma amostra da distribuição a *posteriori* $[\theta|\underline{y}]$ usando esse método. Como $\theta \in (0, 1)$, vamos usar a simplificação descrita no parágrafo anterior. Além disso, como vamos usar

a função `sample()`, esta função padroniza as probabilidades internamente, por isso a implementação do algoritmo fica bastante simples.

Código 5.89: Função para implementar a amostragem por importância no exemplo considerado.

```
sir01 <- function(nsim=10000, m=1000) {
  u <- seq(0, 1, length=m)
  w <- dbeta(u, a.n, b.n)
  return(sample(u, size=nsim, replace=TRUE, prob=w))
}
```

Aplicando essa função e aproximando por este método a probabilidade e intervalo de credibilidade calculados anteriormente temos os resultados a seguir.

```
am.sir <- sir01(N)
mean(am.sir>0.3 & am.sir<0.5)
```

```
[1] 0.1038
```

```
quantile(am.sir, c(0.025, 0.975))
```

```
    2.5%     97.5%
0.1111111 0.3533534
```

Na Figura 5.3, vemos no gráfico da direita o histograma e densidade empírica dessa amostra com a sobreposição da densidade verdadeira $[\theta|\underline{y}]$.

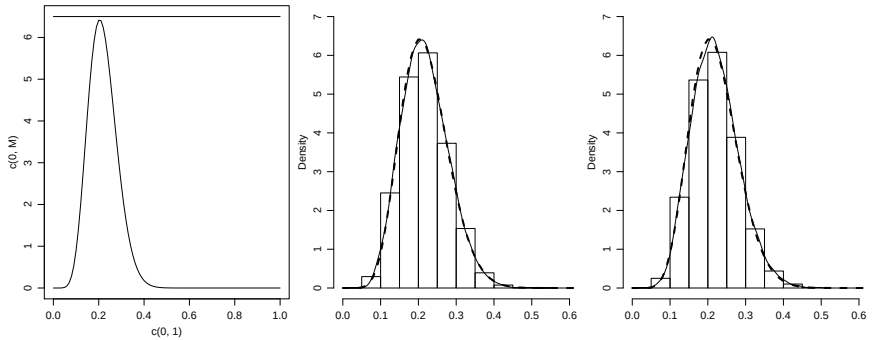

Figura 5.3: Amostras de θ simuladas considerando o método da rejeição (esquerda e centro) e amostragem por importância (direita).

Neste caso usamos `dbeta()` para computar os pesos da *posteriori*, mas de forma mais geral esta quantidade pode ser computada com a expressão da *posteriori* não normalizada dada pelo produto da *priori* pela verossimilhança $[\theta|y] \propto [\theta][y|\theta]$. Desta forma esta estratégia é bastante simples e conveniente, porém limitada a problemas com pequeno número de parâmetros

devido ao tamanho da a malha de discretização crescer exponencialmente com o número de dimensões (parâmetros).

5.5 Métodos de MCMC

Os algoritmos de simulação de Monte Carlo via Cadeia de Markov, (*Markov Chain Monte Carlo* - MCMC), constituem uma classe de algoritmos de simulação sequencial gerando uma cadeia que após atingir convergência passa a gerar amostras da distribuição objetivo. Assim como os algoritmos de rejeição, esta classe de algoritmos se baseia em propor valores para o parâmetro que são aceitos por uma regra associada a valores *posteriori*. Entretanto os valores são simulados de forma simples sequencialmente em uma cadeia e tipicamente cada valor depende do anterior. A regra de rejeição evita o cálculo da constante normalizadora da *posteriori* e portanto depende da avaliação somente da *priori* e da verossimilhança que são conhecidas e especificadas no modelo adotado.

Iniciar o algoritmo, é necessário um valor inicial. Assim, faz-se necessário descartar os primeiros valores simulados para evitar dependência do valor inicial. Além disto, tipicamente as amostras iniciais não são ainda amostras da distribuição objetivo. A cadeia apenas atinge a distribuição estacionária, que é a de interesse, após um período de convergência. Essas amostras iniciais descartadas são as amostras simuladas durante o período de aquecimento da cadeia (*burnin*). Devido a simulação sequencial, os valores sucessivos da amostra obtida apresentam dependência entre si. Portanto, para evitar armazenar informações (parcialmente) redundantes, pode-se optar por reter apenas valores a cada certo número de amostras simuladas. Este procedimento é chamado de raleamento (*thinning*).

5.5.1 Algoritmo de Metropolis-Hastings

Metropolis et al. (1953) propôs uma modificação no algoritmo de Monte Carlo, para aproximar uma integral. Este artigo tornou-se um marco no tema. Esse artigo já tinha mais de 20 mil citações no momento que escrevíamos o texto. A ideia básica do algoritmo é, propor $\tilde{\theta}^{(i+1)}$ a partir de $\theta^{(i)}$, considerando um incremento ϵ_i, isto é,

$$\tilde{\theta}^{(i+1)} = \theta^{(i)} + \epsilon_i$$

iniciando com o valor inicial $\theta^{(0)}$. Assim, este algoritmo é dito ser um algoritmo de passeio aleatório (*random walk*) de Metropolis.

O valor proposto $\tilde{\theta}^{(i+1)}$ é aceito como novo valor, isto é, como θ_{i+1}, com probabilidade igual a

$$p = min\left\{1, \frac{p(\tilde{\theta}^{(i+1)}|y)}{p(\theta^{(i)}|y)}\right\}.$$

Nesta expressão $p(\cdot|y)$ é o valor da densidade da *posteriori* avaliado no valor do parâmetro. Entretanto, o cálculo da quantidade acima é simples pois só precisamos avaliar razão das *posteriori*'s, ou seja, só computamos as *posteriori*'s não normalizadas uma vez que as constantes normalizadoras se cancelam. Vejamos isto com mais detalhes: temos que $[\theta_i|y] = [\theta_i][Y|\theta_i]/[Y]$ e somente o numerador depende do parâmetro e portanto no cálculo da razão para os dois valores de θ que define taxa de aceitação o denominador se cancela.

Se a probabilidade de aceitação é $p = 1$, o valor proposto é igual ou mais provável que o anterior e aceita-se $\tilde{\theta}^{(i+1)}$. Se $p < 1$, o novo valor é menos provável que o anterior e é aceito com probabilidade p, caso contrário retorna-se ao valor anterior. Esta última regra evita que sejam aceitos apenas valores próximos da moda da distribuição, e possibilita a varredura do espaço paramétrico de θ. Desta forma o algoritmo *priori*za, isto é, visita e aceita com maior frequência, valores na região de maior densidade da *posteriori*, mas também visita e aceita, com menor frequência, valores em regiões de menor densidade.

Uma generalização desse algoritmo foi proposta por Hastings (1970). Esta generalização foi no sentido de flexibilizar a possível distribuição proposta, denominemos $q(\theta^{(i+1)}|\theta^{(i)})$. No algoritmo proposto por Metropolis et al. (1953), a distribuição proposta era simétrica, no sentido de que $q(\tilde{\theta}^{(i+1)}|\theta^{(i)}) = q(\theta^{(i)}|\tilde{\theta}^{(i+1)})$. Com a generalização, o valor proposto $\tilde{\theta}^{(i+1)}$ é aceito com probabilidade

$$p = min\left\{1, \frac{p(\tilde{\theta}^{(i+1)})q(\theta^{(i)}|\tilde{\theta}^{(i+1)})}{p(\theta^{(i)})q(\tilde{\theta}^{(i+1)}|\theta^{(i)})}\right\} \tag{5.4}$$

de forma que o algoritmo *random walk* de Metropolis é um caso particular do algoritmo de Hastings.

Outro caso particular, o *independence sampler*, utiliza $q(\tilde{\theta}^{(i+1)}|\theta^{(i)}) = q(\tilde{\theta}^{(i+1)})$, isto é, a distribuição proposta não depende do valor anterior, $\theta^{(i)}$. Nesta situação, similar ao algoritmo de amostragem por importância, é aconselhável escolher a distribuição proposta o mais próxima possível da distribuição de interesse. Neste sentido, uma boa distribuição proposta faz com que a probabilidade de aceitação seja constante. Também, os valores propostos quase não são rejeitados, fazendo com que as amostras sucessivas sejam independentes. Além disso, para distribuições de interesse

com caldas pesadas, recomenda-se que a distribuição proposta tenha caldas ainda mais pesadas.

Consideremos a aplicação do algoritmo de Metropolis-Hastings para a simulação de amostras da distribuição a *posteriori*. Denotando a *priori* por $pr(\cdot)$ e a verossimilhança por $L(\cdot)$, a distribuição de interesse a *posteriori* de θ e $p(\theta|y)$, é

$$p(\theta|y) = \frac{pr(\theta) \times L(\theta)}{preditiva},$$

e o cálculo da razão é simplificado pois a distribuição preditiva não depende de θ.

$$\frac{p(\tilde{\theta}^{(i+1)}|y)}{p(\theta^{(i)}|y)} = \frac{pr(\tilde{\theta}^{(i+1)}) \times L(\tilde{\theta}^{(i+1)})}{pr(\theta^{(i)}) \times L(\theta^{(i)})} \tag{5.5}$$

Vamos agora revisitar o exemplo da seção anterior considerando como *priori* as probabilidades na Tabela 5.1. A dificuldade em considerar o conhecimento exatamente como expressado pelo pesquisador é que a tabela define função de probabilidade por intervalos, do tipo "escada" para θ. Isso torna conveniente abandonar o tratamento analítico para obtenção da *posteriori* em favor de algum procedimento numérico. Vamos adotar o algoritmo de Metropolis-Hastings para obter simulações da *posteriori* de θ.

Inicialmente vamos definir as funções de distribuição a *priori* e a *posteriori* (não normalizada) de θ, ambas na escala *log*, para uma melhor estabilidade numérica.

```
breaks <- c(0, 0.05, 0.1, 0.3, 0.5, 1)
ld.prio <- log(c(0.1, 0.35, 0.5, 0.09, 0.01))
prio.th <- function(theta)
  ld.prio[findInterval(theta, breaks)]
post.th <- function(theta, y, n)
  prio.th(theta) + y*log(theta) + (n-y)*log(1-theta)
```

Vamos considerar uma distribuição proposta $U(max\{0,\theta_i - e\}, min\{1,\theta_i + e\})$. Conforme visto, devemos ter cuidado na escolha de e. Escolher e muito grande fará com que a maioria dos valores propostos sejam descartados. Escolher e muito pequeno fará com que a maioria dos valores propostos sejam aceitos, porém, isso faz com que o espaço paramétrico não seja varrido de forma eficiente. A literatura sugere escolher e de tal forma que aproximadamente 30% dos valores propostos sejam aceitos. Após algumas tentativas, em valor razoável neste exemplo foi $e = 0.3$.

Uma função com um algorítimo Metropolis-Hastings para o exemplo é implementada no código 5.90. A função calcula a taxa de rejeição 5.4 em escala logaritmica e, neste caso, é preciso garantir que os valores estejam em $[0, 1]$. Usamos o mecanismo "..."da linguagem para passar argumentos a `post.th()`.

Código 5.90: Função para implementar o algoritmo de Metropolis-Hastings no exemplo considerado.

```
mh.fun <- function(e=0.1, ini=0.5, nsim=10000, ...) {
  res <- double(nsim+1)
  res[1] <- ini
  rej <- 0
  for (i in 1:nsim) {
    ab <- c(max(0, res[i]-e), min(1, res[i]+e))
    res[i+1] <- runif(1, ab[1], ab[2])
    qij <- dunif(res[i+1], ab[1], ab[2], log=TRUE)
    ab <- c(max(0, res[i+1]-e), min(1, res[i+1]+e))
    qji <- dunif(res[i], ab[1], ab[2], log=TRUE)
    p <- (post.th(res[i+1], ...)+qji)-(post.th(res[i],...)+qij)
    if (runif(1)>exp(p)) {### rejeita
      res[i+1] <- res[i]
      rej <- rej+1
    }
  }
  attr(res, "tx.aceita") <- 1-rej/nsim
  return(res)
}
```

A seguir retiramos amostras usando essa função e verificar a taxa de aceitação.

```
am.mh <- mh.fun(0.3, nsim=25000, y=7, n=30)
attr(am.mh, "tx.aceita")
```

```
[1] 0.34612
```

Vamos considerar como *burnin* e descartar os primeiros 5000 valores simulados, e para os demais reter um a cada 20 valores simulados (*thin*=20) resultando em uma amostra de 1000 valores que supõe-se serem independentes embora isto não seja estritamente necessário. Como usual em inferência por amostragem, podemos estimar quantidades de interesse com essa amostra selecionada.

```
am.sel <- am.mh[seq(5001, length(am.mh), 20)]
mean(am.sel>0.3 & am.sel<0.5)     # média da posterior
```

```
[1] 0.06393606
```

```
quantile(am.sel, c(0.025, 0.975)) # int.(quantis) de credibilidade
```

```
     2.5%     97.5%
0.1224445 0.3753140
```

Na Figura 5.4 podemos visualizar o histograma a *priori* dada pelo pesquisador e o histograma das amostras simuladas. Adicionamos a curva da verossimilhança padronizada, para fins de comparação. O histograma da amostra da *posteriori* fica entre a *priori* e a verossimilhança.

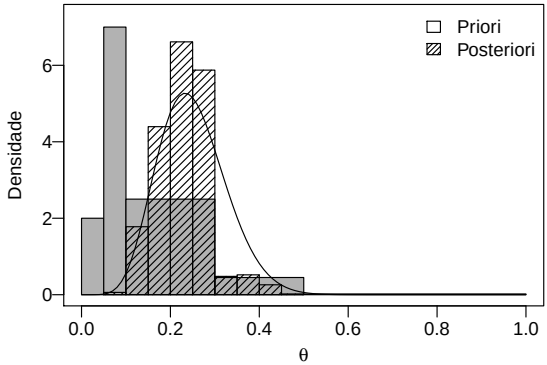

Figura 5.4: Histograma (cinza) da *priori* curva da verossimilhança padronizada e histograma (tracejado) das amostras da *posteriori*.

5.5.2 Algoritmo amostrador de Gibbs

O algoritmo amostrador de Gibbs (Geman & Geman, 1984), foi proposto para a aproximação de distribuições a *posteriori* por Gelfand & Smith (1990). Este algoritmo pode ser utilizado para simular de distribuições multivariadas de forma desconhecida de um vetor de parâmetros $\theta = \{\theta_1, \theta_2, \ldots, \theta_k\}$, a partir de simulações sequenciais das distribuições condicionais individuais $[\theta_i|\theta_{-i}]$. O resultado que justifica o algoritmo é o de que após um certo período as simulações da cadeia convergem para distribuição estacionária desejada.

A simulação de cada parâmetro θ_i é feita a partir da distribuição condicional nos demais parâmetros e portanto, em um esquema de amostragem sequencial, a distribuição é avaliada no estado atual dos demais parâmetros. Podemos resumir esse algoritmo nos seguintes passos:

- considere valores iniciais $\{\theta_1^{(0)}, \theta_2^{(0)}, \ldots, \theta_k^{(0)}\}$
- para $i = 1,2,...,N$
  - simule $\theta_1^{(i)}$ de $p(\theta_1^{(i-1)}|\theta_2^{(i-1)}, \theta_3^{(i-1)}, \ldots, \theta_k^{(i-1)})$
  - simule $\theta_2^{(i)}$ de $p(\theta_2^{(i-1)}|\theta_1^{(i)}, \theta_3^{(i-1)}, \ldots, \theta_k^{(i-1)})$
  - $\vdots$
  - simule $\theta_k^{(i)}$ de $p(\theta_k^{(i-1)}|\theta_1^{(i)}, \theta_2^{(i)}, \ldots, \theta_{k-1}^{(i)})$

Portanto, o algoritmo de Gibbs é um tipo de MCMC com taxa de aceitação de 100%. A simulação a partir das distribuições condicionais de cada

parâmetro pode ser generalizada caso a distribuição condicional de alguns dos parâmetros não tenha forma conhecida. Neste caso, pode-se usar o algoritmo de Metropolis-Hastings para estes parâmetros e neste caso dizemos que implementamos Metropolis-Hastings dentro de Gibbs. Na próxima seção veremos um exemplo de aplicação do algoritmo amostrador de Gibbs, com uma pequena avaliação e comparação com o algoritmo de Metropolis-Hastings.

5.6 MCMC para distribuição gaussiana

Nesta seção, a título de ilustração e comparação, implementamos em uma situação na qual conhecemos as distribuições condicionais: (i) o algoritmo amostrador de Gibbs ; (ii) o algoritmo de Metropolis-Hastings. No exemplo simulamos amostras da distribuição *posteriori* dos parâmetros da distribuição gaussiana. Neste caso, com escolhas adequadas de *priori*s, as distribuições marginais à *posteriori* possuem formas analíticas conhecidas e não seria necessário simular da distribuição à *posteriori*. Porém, usamos o exemplo para comparar resultados dos algoritmos de simulação com a distribuição exata. Implementaremos os métodos utilizando o R, embora já esteja disponível implementações mais eficientes e gerais, nas funções `MCMCmetrop1R()` e `MCMCregress()` do pacote **MCMCpack**. Mais adiante ilustraremos o uso deste pacote em outros exemplos.

Considere uma amostra aleatória $\underline{y} = \{y_1, y_2, ..., y_n\}$ sob independência condicional com $Y_i \sim N(\mu, \phi^{-1})$, com (μ, ϕ), parâmetros de média e precisão, respectivamente, desconhecidos.

Vamos propor uma distribuição conjugada (Degroot, 1970) *Normal-Gama (NG)* à *priori* para os dois parâmetros, $[\mu,\phi] \sim NG(m_0, n_0, a, b)$. Temos então

$$[\mu,\phi] \propto \phi^{a-1} \exp\{-b\phi\}\phi^{1/2} \exp\{-\frac{n_0/2}{\phi}(\mu - m_0)^2\}$$

que é a distribuição à *priori* conjunta para $\{\mu,\phi\}$. Isso é equivalente às seguintes *priori*s individuais: $\lambda \sim Gamma(a, b)$, $\mu|\phi \sim Normal(m, (n_0\phi)^{-1})$. Integrando $\mu|\phi$ em relação a ϕ, obtemos $\mu \sim t(2a, m_0, (a/b)^{-1})$, isto é, uma distribuição t com $2a$ graus de liberdade, média m_0 e variância $(a/b)^{-1}$.

Com este último resultado, podemos tomar decisões sobre a escolha dos parâmetros da distribuição à *priori*. Por exemplo, escolher a e b que expressam a incerteza a *priori* sobre μ.

A distribuição à *posteriori* é

$$[\mu,\phi|\underline{y}] \propto \phi^{a+n/2-1} \exp\left\{-\frac{1}{2}\phi(ns^2 + 2b)\right\} \phi^{1/2} \exp\left\{-\frac{1}{2}n\phi((\mu - \overline{y})^2 + n_0 m_0^2)\right\}$$

que pode ser escrita na forma:

$$\begin{aligned}[\mu,\phi|\underline{y}] &\propto \phi^{a+n/2-1}\exp\left\{-\frac{1}{2}\phi(ns^2+2b+nn_0\frac{(m_0-\overline{y})^2}{n_0+n})\right\} \\ &\quad \phi^{a_n-1}\exp\left\{-\frac{1}{2}\phi(n_0+n)(\mu-m_n)^2\right\} \qquad (5.6)\end{aligned}$$

com

$$\begin{aligned} m_n &= (n\overline{y}+n_0m_0)/(n_0+n) \\ n_n &= n_0+n \\ a_n &= a+\frac{n}{2} \\ b_n &= b+ns^2/2+(\overline{y}-m_0)^2nn_0/(2(n_0+n)), \end{aligned}$$

são os parâmetros da distribuição conjunta a *posteriori* de $\{\mu,\phi\}$, ou seja,

$$\begin{aligned}[\phi|\underline{y}] &\sim Gamma(a_n,b_n) \\ [\mu|\phi,\underline{y}] &\sim N(m_n,(n_n\phi)^{-1})\end{aligned}$$

e notamos que ϕ não depende de μ à *posteriori*, mas μ depende de ϕ. Neste caso podemos obter a distribuição condicional $[\mu|\phi,\underline{y}]$ em relação a ϕ que é uma distribuição t com $2a_n$ graus de liberdade, média m_n, e variância $(n_na_n/b_n)^{-1}$.

$$[\mu|\underline{y}] \sim t(2a_n,m_n,(n_na_n/b_n)^{-1}),$$

Este é um exemplo onde conhecemos as distribuições marginais à *posteriori*. Consequentemente podemos calcular intervalos de credibilidade e medidas de interesse acerca dos parâmetros. Na prática, são poucos os modelos nos quais conhecemos as distribuições marginais. Porém, pelo algoritmo de Gibbs podemos utilizar as distribuições condicionais, isto é, de um parâmetro condicionado nos demais. Estas, por sua vez, podem ter expressões de funções de densidades conhecidas ou não. Se as conhecemos, podemos usar o amostrador de Gibbs para obter amostras da distribuição à *posteriori*. Se não conhecemos, podemos usar o algoritmo de Metropolis ou Metropolis-Hastings.

Funções auxiliares

Vamos considerar as distribuições marginais exatas de cada parâmetro para fins de comparação. A densidade Gamma encontra-se implementada na função `dgamma()`. Embora a densidade t também esteja implementada no R mas vamos por ilustração definir a seguir, com `ni` graus de liberdade, parâmetro de não centralidade `mu`, e variância `s2`. O valor da densidade é calculado na escala logarítmica e exponenciado ao final.

```
dt.g <- function(x, ni, mu, s2, log=FALSE) {
  res <- lgamma((ni+1)/2) + (ni/2)*log(ni)-
    (lgamma(ni/2)+0.5*log(pi*s2)) - ((ni+1)/2)*log(ni+((x-mu)^2)/s2)
  return(if (log) res else exp(res))
}
```

Agora, vamos definir a função *priori* $\times$ *verossimilhana* necessária para o algoritmo de Metropolis-Hastings. Esta função é escrita com *default* na escala logarítmica. Os argumentos são: `x` - vetor de dados; `mu` e `phi` - valores correntes dos parâmetros; e parâmetros das distribuições à *priori*: `m`, `v`, `a` e `b`.

```
lpvNorm <- function(x, mu, phi, m0, n0, a, b, log=TRUE)
  dnorm(mu, m0, sqrt(1/(n0*phi)), log=log) +
  dgamma(phi, a, b, log=log) +
  sum(dnorm(x, mu, sqrt(1/phi), log=log))
```

Anteriormente obtivemos intervalos de credibilidade baseado em quantis, ou seja valores que deixam a mesma massa de probabilidade acima e abaixo dos limites do intervalo. Entretanto, esta definição não é única e a seguir veremos uma outra forma, em geral preferida, que são os intervalos de credibilidade de mais alta densidade (*hight probability density - HPD*). Para obter o intervalo HPD para uma função de densidade, definimos a função a seguir. A função se baseia no fato de que os limites do intervalo terão o mesmo valor de densidade e faz então a procura destes limites para o nível desejado. Essa implementação é restrita a funções com uma única moda no intervalo (`a`, `b`). Por exemplo de chamada da função obtemos o intervalo de mais alta densidade em uma $N(0,1)$, procurado entre os limites $(-7,7)$ que fornece valores bem conhecidos.

Código 5.91: Função para calcular o intervalo de mais alta densidade para uma função.

```
hpd.f <- function(fun, interval, prob=0.95, ...) {
 ## fun: funcão a para a qual sera encontrada o intervalo HPD.
 ## a, b: limites, inferior e superior para o intervalo
 ## prob: area (probabilidade, se densidade) do intervalo
 ## ...: argumentos adicionais passados para fun
  m <- optimize(function(y)-log(fun(y,...)), interval=interval)$min
  q <- function(h,a,b)
    uniroot(function(y) fun(y, ...)-h, interval=c(a,b))$root
  d <- optimize(function(h, ...)
                (integrate(fun, q(h,interval[1],m),
                           q(h,m,interval[2]), ...)$value-prob)^2,
                interval=c(0, fun(m, ...)), ...)$minimum
  c(lower=q(d,interval[1],m), upper=q(d,m,interval[2]))
}
```

```
hpd.f(dnorm, c(-7,7), prob=c(0.95))
```

```
   lower    upper
-1.95981  1.95981
```

Um exemplo comum em inferência estatística é encontrar o intervalo HPD considerando uma distribuição assimétrica. Por exemplo, para uma χ^2 com 10 graus de liberdade

```
(hpdc10 <- hpd.f(function(x) dchisq(x,10), c(0,100)))
   lower    upper
2.41369 18.86136
```

que, por definição, tem densidade igual nos limites

```
dchisq(hpdc10, 10)
     lower      upper
0.01322021 0.01322023
```

e amplitude menor que o intervalo baseado nos quantis.

```
qchisq(c(0.025, 0.975), 10)
[1]  3.246973 20.483177
```

Vamos agora definir uma outra função para a construção dos intervalos HPD agora a partir de de amostras de uma distribuição. Nossa implementação assume que a distribuição desejada é aproximada pela empírica da amostra e baseia-se agora na propriedade de que o HPD é o menor de todos os intervalos. A implementação encontra os limites de um intervalo com os valores da amostra ordenada. Desta forma varremos os intervalos possíveis para o nível desejado e selecionamos o menor deles.

Código 5.92: Função para calcular o intervalo de mais alta densidade para uma amostra.

```
hpd.mcmc <- function(x, prob=0.95) {
  x <- sort(x) ; n <- length(x)
  n.amp <- max(1, min(n-1, round(prob*n)))
  amp <- sapply(1:(n-n.amp), function(i)
                                x[i+n.amp]-x[i])
  id <- order(amp)[1]
  c(lower=x[id], upper=x[id+n.amp])
}
```

Existem outros algoritmos para intervalos HPD como, por exemplo, implementado a função `HPDinterval()` do pacote **coda** do R. Este pacote implementa métodos específicos para análise e diagnóstico de cadeias de Markov.

Para comparação com o intervalo anteriormente obtido para função da χ^2_{10}, vamos obter intervalos HPD como nossa função e a do pacote **coda** o intervalo baseado em quantis a uma amostra. Obtemos também o comprimento dos intervalos.

```
set.seed(1)
x <- rchisq(10000, 10)
(IC1 <- hpd.mcmc(x))
   lower     upper
2.537584 18.853464
```

```
require(coda); (IC2 <- as.vector(HPDinterval(as.mcmc(x))))
[1]  2.537584 18.853464
(IC3 <- quantile(x, prob=c(0.025, 0.975)))
    2.5%     97.5%
3.253889 20.418937
unname(c(diff(IC1), diff(IC2), diff(IC3)))
[1] 16.31588 16.31588 17.16505
```

e observamos que o resultado é próximo ao obtido pela função `hpd.f()`.

Amostrador de Gibbs

A função a seguir ilustra a implementação do o algoritmo amostrador de Gibbs para o caso gaussiano com distribuição a *priori* NormalGamma. Para gerar as amostras do *burnin* sem guardá-las usamos aqui um "truque" de chamar a função dentro dela mesma dentro do `if()`. Desta forma são geradas inicialmente todas as amostras do *burnin* de depois apenas a última delas é usada como valor inicial da amostra definitiva. Implementações alternativas podem ser feitas. Esta função retorna uma `matrix` $nsim \times 2$, da classe `mcmc` com atributos dessa classe definidos no pacote **coda**. As colunas contém as amostras do parâmetro e de precisão, respectivamente.

Algoritmo de Metropolis-Hastings

O algoritmo para simular μ_t e ϕ_t tem a estrutura do algoritmo de Gibbs. Porém, a cada passo é feita uma proposta de um novo valor, que pode ser aceito ou não. A distribuição proposta para μ_t é

$$[\mu_t] \sim N(\mu_{t-1}, \epsilon_\mu).$$

Neste caso, $[\mu_t|\mu_{t-1}] = [\mu_{t-1}|\mu_t]$ simplificando termos no cálculo do quociente para decisão de aceitação.

No caso de ϕ a distribuição proposta utilizada foi a Uniforme centrada no valor atual do parâmetro e truncada em zero, $U(max\{0, \phi_{t-1} - \epsilon_\phi\}, \phi_{t-1} + \epsilon_\phi)$. Uma alternativa a impor este tratamento na borda do espaço paramétrico seria escrever o algoritmo com uma reparametrização com $\log(\phi)$ e ao final exponenciando os valores obtidos. Neste caso, na decisão de aceitação do novo valor deve-se levar em conta que ϕ_t/ϕ_{t-1} nem sempre será igual à ϕ_{t-1}/ϕ_t.

No código 5.94 implementamos um algoritmo de Metropolis-Hastings. É necessário definir valores de ϵ_μ e ϵ_ϕ de forma que o percentual de aceitação fique em torno de 25% a 35%. Isto é normalmente feito por tentativa e erro chamando-se a função diversas vezes e verificando a aceitação. A função para implementar esse algoritmo é dada a seguir e retorna um objeto com a mesma estrutura do retornado pelo código 5.93.

Código 5.93: Algoritmo amostrador de Gibbs para o caso normal com *priori* NormalGamma.

```
gibbs.norm <- function(y, ini=c(0,1), m0=0, n0=0.001, a0=0.001,
                       b0=0.001, nsim=10000, nburn=5000){
  ## y: a amostra de dados
  ## ini: valores iniciais média e variância.
  ## m0, n0, a e b: parâmetros da distr. a priori.
  ## nsim: número de amostras a serem simuladas da
  ##       distr a posteriori, apos o burnin.
  ## nburn: numero de amostras iniciais a serem descartadas
  if (nburn>0) {
    res <- gibbs.norm(y, ini=ini, m0=m0, n0=n0,
                      a0=a0, b0=b0, nsim=nburn, nburn=0)
    return(gibbs.norm(y, ini=res[nrow(res),], m0=m0, n0=n0,
                      a0=a0, b0=b0, nsim=nsim, nburn=0))
  }
  n <- length(y)
  sx <- sum(y)
  xbar <- sx/n
  s2 <- mean((y-xbar)^2)
  mu <- as.double(rep(ini[1], nsim+1))
  phi <- as.double(rep(ini[2], nsim+1))
  a.n <- a0 + n/2
  n.n <- n0 + n
  b.n <- b0 + (n*s2 + (n*n0)*((xbar-m0)^2)/n.n)/2
  m.n <- (sx + n0*m0)/n.n
  for (i in 1:nsim) {
    phi[i+1] <- rgamma(1, a.n, b.n)
    mu[i+1] <- rnorm(1, m.n, sqrt(1/(n.n*phi[i+1])))
  }
  return(structure(cbind(mu=mu[-1], phi=phi[-1]),
                   class='mcmc', mcpar=c(1,nsim,1)))
}
```

Código 5.94: Função que implementa o algoritmo de Metropolis-Hastings para o caso normal com *priori* NormalGamma.

```
mh.norm <- function(y, ini=c(0,1), m0=0, n0=0.001,
                    a=0.001, b=0.001, sprop=c(1,1), nsim=1000) {
  ## y: dados ; ini: valores iniciais para os parâmetros.
  ## m0, n0, a, b: parâmetros da distribuição à priori.
  ## sprop: vetor com o desvio-padrão da distribuição
  ##        normal proposta para mu e a semi-amplitude da
  ##        distribuição uniforme proposta para sigma2.
  ## nsim: número de amostras da distribuicão à posteriori.
  n <- length(y)
  mu <- as.double(rep(ini[1], nsim+1))
  phi <- as.double(rep(ini[2], nsim+1))
  rej <- c(mu=0, phi=0)
  ab <- c(max(0,phi[1]-sprop[2]), phi[1]+sprop[2])
  for (i in 1:nsim) {
    mu[i+1] <- rnorm(1, mu[i], sprop[1])
    k <- lpvNorm(y, mu[i+1], phi[i], m0, n0, a, b) -
      lpvNorm(y, mu[i], phi[i], m0, n0, a, b)
    if (exp(k)<runif(1)) { ## rejeita mu[i+1]
      mu[i+1] <- mu[i]
      rej[1] <- rej[1]+1
    }
    phi[i+1] <- runif(1, ab[1], ab[2])
    d.2 <- dunif(phi[i+1], ab[1], ab[2], log=TRUE)
    ab <- c(max(0, phi[i+1]-sprop[2]), phi[i+1]+sprop[2])
    d.1 <- dunif(phi[i], ab[1], ab[2], log=TRUE)
    num <- lpvNorm(y, mu[i+1], phi[i+1], m0, n0, a, b) + d.1
    den <- lpvNorm(y, mu[i+1], phi[i], m0, n0, a, b) + d.2
    if (exp(num - den)<runif(1)) { ### rejeita phi[i+1]
      phi[i+1] <- phi[i]
      ab <- c(max(0, phi[i]-sprop[2]), phi[i]+sprop[2])
      rej[2] <- rej[2]+1
    }
  }
  structure(cbind(mu=mu[-1], phi=phi[-1]), tx.aceita=1-rej/nsim)
}
```

A procura do valor ϵ de calibração da cadeia pode ser incorporada ao algoritmo. Vamos ilustrar isto implementando uma adaptação automática para as variâncias das distribuições propostas. Dado um número de amostras, `nsim`, a serem extraídas, é definido um tamanho de bloco, `atualiza`. Assim, define-se o número de blocos de simulação. A cada bloco simulado, é testado se a taxa de aceitação está adequada. Assim, a variância da distribuição proposta é corrigida a cada bloco de forma que no próximo bloco será utilizada uma variância mais adequada.

A função `brmh()` permite que seja passada uma função para o cálculo de

um fator de atualização da variância da distribuição proposta. Como exemplo, seja a seguint regra: Se a taxa de aceitação x do último bloco for menor que 0.3, o fator será $0.1 + x * 3$. Ou seja, o fator estará no intervalo $(0.1, 1)$, isto é, a variância da proposta será diminuida. Se a taxa de aceitação x no último bloco for maior que 0.3, o fator será $x/0.3$. Neste caso, o fator estará no intervalo $(1, 1/0.3)$, isto é, a variância da proposta será aumentada. Vamos definir uma função para implementar essa regra:

```
ats.f <- function(x) ifelse(x<0.3, 0.1+x*3, x/.3)
```

O código 5.95 implementa a adaptação, juntamente com a inclusão do período de *burnin*.

A função `brmh()`, retorna um objeto da classe `mcmc` com atributos dessa classe definidos no pacote **coda**. Alem destes, retorna uma matriz com a taxa de aceitação em cada bloco, atributo `bl.acpt()` e uma matriz com `sprop` em cada bloco e atualizado para um bloco posterior, atributo `sprop`.

De forma análoga, o desempenho do algoritmo M-H pode ser melhorado pelo uso de uma *proposal* adaptativa permitindo que os valores de ϵ variem de acordo com o estado da cadeia.

Experimento para avaliação

Para avaliar os algoritmos realizamos o experimento considerando os fatores: tamanho de amostra n, valor de μ e valor de ϕ. Consideramos os níveis: $n = \{30, 100\}$, $\mu = \{0{,}10\}$ e $(\phi)^{-1} = \{0.5{,}1{,}2\}$. As amostras, considerando cada combinação desses fatores, são todas simuladas e armazanadas em uma lista

```
set.seed(1)
am <- list(rnorm(30,0,sqrt(.5)),rnorm(30,0,1),rnorm(30,0,sqrt(2)),
           rnorm(30,10,sqrt(.5)),rnorm(30,10,1),rnorm(30,10,sqrt(2)),
           rnorm(100,0,sqrt(.5)),rnorm(100,0,1),rnorm(100,0,sqrt(2)),
           rnorm(100,10,sqrt(.5)),rnorm(100,10,1),rnorm(100,10,sqrt(2)))
```

Os parâmetros da distribuição a *priori* foram $m0 = 0$ e os demais todos iguais com $n0 = a = b = 0.001$. Simulamos 15.000 amostras da distribuição a *posteriori* em cada cenário. Não vamos descartar as primeiras amostras para fazer uma comparação da convergência dos dois algoritmos. No algoritmo de Metropolis-Hastings, vamos considerar um `atualiza=200`, para que o parâmetro de variância da distribuição proposta seja rapidamente calibrado. Nos comandos a seguir criamos uma lista para armazenar os resultados de cada algoritmo, para cada conjunto de dados e comparamos os tempos dos algoritmos.

```
res <- list('N(0,.5) n=30'=list(), 'N(0,1) n=30'=list(),
            'N(0,2) n=30'=list(), 'N(10,.5) n=30'=list(),
            'N(10,1) n=30'=list(), 'N(10,2) n=30'=list(),
            'N(0,.5) n=100'=list(), 'N(0,1) n=100'=list(),
            'N(0,2) n=100'=list(), 'N(10,.5) n=100'=list(),
            'N(10,1) n=100'=list(), 'N(10,2) n=100'=list())
system.time({
```

Código 5.95: Implementação do burnin e adptação automática da distribuição proposta para algoritmo de Metropolis-Hastings.

```
brmh <- function(fmh, ini, sprop, f.ats=ats.f,
                 nsim=15000, nburn=5000, atualiza=200, ...) {
  ## fmh  : função c/ alg. de Metropolis-Hastings
  ##       com argumentos 'ini', 'sprop' e 'nsim',
  ##       retorna matriz com n_o colunas = n_o de parâmetros
  ##       e atributo 'tx.aceita' com as taxas de aceitação
  ## ini  : valores iniciais
  ## sprop: desvio-padrão ou semi-variancia, pode ser vetor
  ## f.ats: função para atualização de 'sprop'
  ## nsim : número de amostras a obter da posteriori.
  ## nburn: número de amostras iniciais a serem descartadas.
  ## atualiza: número de amostras simuladas a cada atualização
  ##          das variâncias da distribuições propostas.
  ## ... : parametros adicionais passados para a função 'fmh'
  if (nburn>0) {
    res <- brmh(fmh, ini=ini, sprop=sprop,
                nsim=nburn, nburn=0, atualiza=atualiza, ...)
    sprop <- matrix(attr(res, "sprop"), ncol=2)
    return(brmh(fmh=fmh, ini=res[nrow(res), ],
                sprop=sprop[nrow(sprop), ],
                nsim=nsim, nburn=0, atualiza=atualiza, ...))
  }
  if (atualiza<nsim) {
    npars <- length(ini)
    nbl <- length(iref <- seq(1, nsim, atualiza))
    lbl <- diff(iref)
    lbl[length(lbl)+1] <- nsim - sum(lbl)
    m.s <- matrix(sprop, nbl+1, ncol=npars, byrow=TRUE)
    m.acept <- matrix(0, nbl, ncol=npars, byrow=TRUE)
    res <- matrix(NA, nsim, npars)
    for (bl in 1:length(lbl)) {
      r.b <- fmh(ini=ini, sprop=m.s[bl,], nsim=lbl[bl], ...)
      res[1:lbl[bl]+iref[bl]-1, ] <- r.b
      k <- nrow(r.b)
      ini <- r.b[k,]
      b.acc <- attr(r.b, "tx.aceita")
      m.acept[bl, ] <- b.acc
      m.s[bl+1,] <- m.s[bl,]*f.ats(b.acc)
    }
    return(structure(res, mcpar=c(1, nsim, 1), class="mcmc",
                     aceita=colSums(lbl*m.acept)/nsim,
                     bl.aceita=m.acept, sprop=m.s))
  }
  else return(structure(fmh(ini=ini, sprop=sprop, nsim=nsim, ...),
                  class="mcmc", mcpar=c(1, nsim, 1), sprop=sprop))
}
```

```
  for (i in 1:12)
    res[[i]][[1]] <- gibbs.norm(am[[i]], nsim=15000, nburn=0)
})
   user  system elapsed
 11.121   0.020  11.190

system.time({
  for (i in 1:12)
    res[[i]][[2]] <- brmh(mh.norm, c(0,1), c(.1,.1),
                          nburn=0, atualiza=100, y=am[[i]])
})
   user  system elapsed
 93.006   0.060  94.828
```

Os tempos absolutos não são relevantes aqui e vão depender do equipamento utilizado. Estamos mais interessados na relação de tempo entre os algoritmos. O algoritmo de Metropolis-Hastings demora mais para gerar o mesmo número de amostras do que amostrador de Gibbs. Isto é devido à rejeições e também ao maior número de operações efetuadas no M-H a cada iteração. Por exemplo, o valor da densidade da distribuição a *posteriori* precisa ser calculado duas vezes para cada parâmetro a cada iteração.

Resultados

Na Tabela 5.3 observamos as taxas de aceitação do algoritmo Metropolis-Hastings, para μ (primeira coluna) e para $1/\sigma^2$ (segunda coluna). Todas ficaram próximos de 30%, o que indica que a regra adaptativa implementada funcionou adequadamente. Uma observação aqui é que o valor inicial do parâmetro semi-variância da distribuição proposta para $1/\sigma^2$ pode ser inadequado em vários cenários. Com um número de amostras da distribuição a *posteriori* pequeno, o número de passos para a adaptação desse parâmetro deve também ser pequeno, para que sua calibração seja rápida.

Tabela 5.3: Taxas de aceitação (M-H) autocorrelações (M-H e Gibbs)

	txmu	txs2	a.Gmu	a.Gphi	a.MHmu	a.MHphi
N(0,.5) n=30	0.308	0.314	-0.004	-0.003	0.665	0.659
N(0,1) n=30	0.309	0.314	-0.003	-0.017	0.676	0.639
N(0,2) n=30	0.310	0.310	-0.018	-0.004	0.677	0.644
N(10,.5) n=30	0.310	0.315	-0.007	-0.016	0.988	0.698
N(10,1) n=30	0.310	0.314	-0.001	0.004	0.977	0.681
N(10,2) n=30	0.311	0.310	0.000	-0.005	0.957	0.688
N(0,.5) n=100	0.307	0.312	0.001	0.000	0.681	0.648
N(0,1) n=100	0.306	0.311	0.014	0.005	0.674	0.615
N(0,2) n=100	0.308	0.309	0.011	-0.003	0.670	0.606
N(10,.5) n=100	0.307	0.312	-0.007	0.004	0.990	0.739
N(10,1) n=100	0.309	0.310	0.017	-0.016	0.986	0.739
N(10,2) n=100	0.308	0.306	0.004	-0.010	0.981	0.728

Na Tabela 5.3, observamos a correlação entre os valores sucessivos simulados para μ e $1/\sigma^2$. Nas colunas 3 e 4 considerando o algoritmo de Gibbs e nas colunas 5 e 6, considerando o algoritmo de Metropolis-Hastings. Notamos que para este algoritmo, as correlações são próximas de 1, indicando uma grande dependência entre os valores sucessivos. Isso é esperado neste caso devido ao fato de que a taxa de aceitação do valor proposto ser baixa.

Na Figura 5.5 podemos ver os primeiros 200 valores da amostra a *posteriori* de μ pelos dois algoritmos, para cada uma das 12 amostras simuladas. Nessa Figura observamos que a convergência do amostrador de Gibbs é bem mais rápida que o algoritmo de Metropolis-Hastings. Como o valor inicial é 0 (zero), para as amostras simuladas com $\mu = 10$, este algoritmo demorou em torno de 150 a até mais de 200 amostras para chegar ao patamar 10.

A variância proposta para a distribuição proposta para simular de $\mu|\underline{y}$ foi muito pequena, 0.1^2. Assim, as taxas de aceitação no primeiro bloco foram altas. Para os primeiros 3 blocos temos:

```
 t(sapply(res, function(x) attr(x[[2]], 'bl.aceita')[1:3,1]))
```

```
               [,1] [,2] [,3]
N(0,.5) n=30   0.78 0.46 0.37
N(0,1) n=30    0.79 0.52 0.34
N(0,2) n=30    0.90 0.64 0.36
N(10,.5) n=30  0.73 0.76 0.30
N(10,1) n=30   0.71 0.73 0.34
N(10,2) n=30   0.78 0.75 0.43
N(0,.5) n=100  0.71 0.34 0.30
N(0,1) n=100   0.62 0.37 0.38
N(0,2) n=100   0.74 0.65 0.26
N(10,.5) n=100 0.61 0.60 0.22
N(10,1) n=100  0.69 0.61 0.22
N(10,2) n=100  0.61 0.61 0.35
```

Para a terceira amostra, N(0,2) e n=30, a taxa no primeiro bloco foi particularmente alta. Isto porque este um dos cenários de maior variância e com 'n' pequeno, fazendo com que a variância de $\mu|\underline{y}$ seja bem maior que a proposta. Assim, a variância da distribuição proposta neste caso foi aumentada nos primeiros 6 ajustes da distribuição proposta:

```
 head(attr(res[[3]][[2]], 'sprop')[,1], 7)
```

```
[1] 0.100000 0.300000 0.640000 0.768000 1.075200 1.290240 0.903168
```

Para as amostras com $\mu = 10$, no primeiro bloco, mesmo que a variância da proposta estivesse adequada para a distribuição de equilibrio, ela pode ter sido ajustada porque o algoritmo ainda não estava simulando dessa distribuição. Observando o caso para N(10, 0.5) e n=100,

```
 head(attr(res[[10]][[2]], 'sprop'), 10)[,1]
```

```
 [1] 0.1000000 0.2033333 0.4066667 0.3090667 0.2719787 0.2556599 0.3323579
 [8] 0.2625628 0.2468090 0.2961708
```

Notamos que a variância da distribuição proposta foi aumentada duas vezes consecutivas no início e depois diminuida novamente.

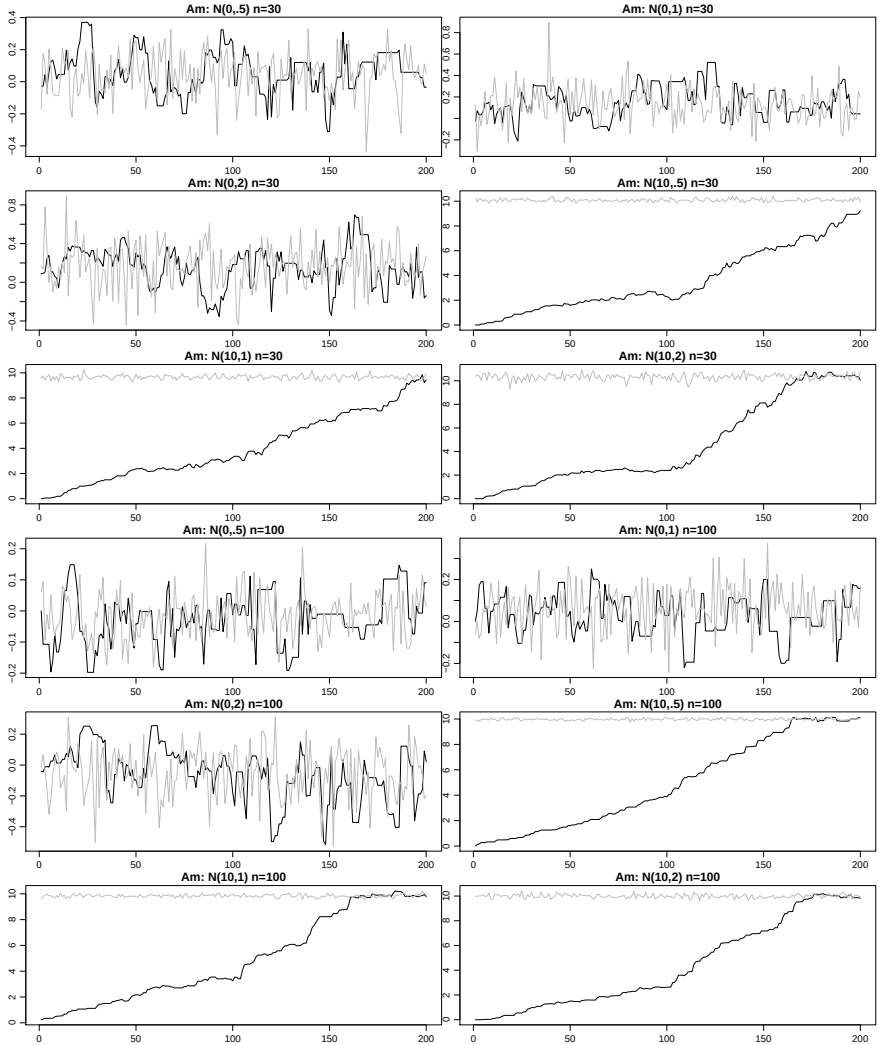

Figura 5.5: Primeiras 200 amostras da *posteriori* de μ, com Metropolis-Hastings (preto) e Gibbs (cinza), para cada um dos 12 conjuntos de dados simulados. Em todos os casos, o valor inicial de μ foi zero.

Na Figura 5.6 podemos ver as primeiras 200 amostras da distribuição a *posteriori* de $1/\sigma^2$. Notamos aqui também que o algoritmo de Metropolis-Hastings demora mais para convergir que o de Gibbs. No caso da amostra N(10, 0.5) e n=30 o esse algoritmo não convergiu nas primeiras 200 amostras simuladas. Neste caso, observamos que nos quatro primeiros ajustes a variância da distribuição proposta foi aumentada e diminuida novamente

```
head(attr(res[[3]][[2]], 'sprop'), 7)[,2]
[1] 0.1000000 0.3000000 0.6000000 0.8400000 1.0640000 0.8086400 0.6145664
```

Isso provavelmente também devido à relativa demora na convergência.

Na Figura 5.7 vemos os valores simulados de μ e $1/\sigma^2$ para o primeiro conjunto de dados simulado. Os gráficos da esquerda referem-se a amostras obtidas pelo algoritmo de Gibbs e os da direita, pelo algoritmo de Metropolis-Hastings. Nos dois gráficos superiores temos as primeiras 5000 amostras da distribuição a *posteriori* de μ e nos gráficos da segunda linha, as primeiras 5000 amostras da distribuição a *posteriori* de $1/\sigma^2$. Em ambos os gráficos de Gibbs há uma menor autocorrelação e portanto uma variação maior entre os valores sucessivos gerando um aspecto visual com uma região mais compacta em preto. No algoritmo de Metropolis-Hastings há repetição de valores sucessivos devido as rejeições, causando pequenos traços horizontais.

A partir das 15.000 amostras da distribuição à *posteriori*, as primeiras 5.000 foram descartadas e as análises foram feitas armazenando uma amostra a cada 10 amostras simuladas. Esta última decisão foi particularmente importante para o algoritmo de Metropolis-Hastings. Pois como visto na Tabela 5.3, as amostras sucessivas simuladas por este algoritmo apresentaram uma forte correlação. Portanto, o número de amostras da distribuição a *posteriori* selecionadas para inferência foi $(15000 - 5000)/10 = 1000$ amostras.

Na terceira para μ e na última linha $1/\sigma^2$ dos gráficos da Figura 5.7, temos os traços das amostras selecionadas de tamanho 300. Notamos que ambos os traços (Gibbs e Metropolis-Hastings), da terceira e ambos da última linhas, são parecidos. Nessas amostras de tamanho 1000 selecionadas, a correlação entre os valores sucessivos é pequena, conforme medidas nas duas últimas colunas da Tabela 5.4. Na Figura 5.8 temos as densidades aproximadas pelas amostras selecionadas da distribuição a *posteriori* para μ, por ambos os algoritmos para cada um dos 12 conjuntos de dados simulados. A densidade marginal a *posteriori* exata está sobreposta às densidades empíricas das amostras dos algoritmos. Notamos que as densidades aproximadas pelos métodos MCMC estão bem próximas da densidade exata.

Similar a Figura 5.8, temos os resultados para $1/\sigma^2$ na Figura 5.9. Também aqui, as densidades aproximadas pelas amostras selecionadas da distribuição a *posteriori* são muito próximas das densidades marginais exatas.

Um resumo comparativo dos métodos pode ser feito considerando os

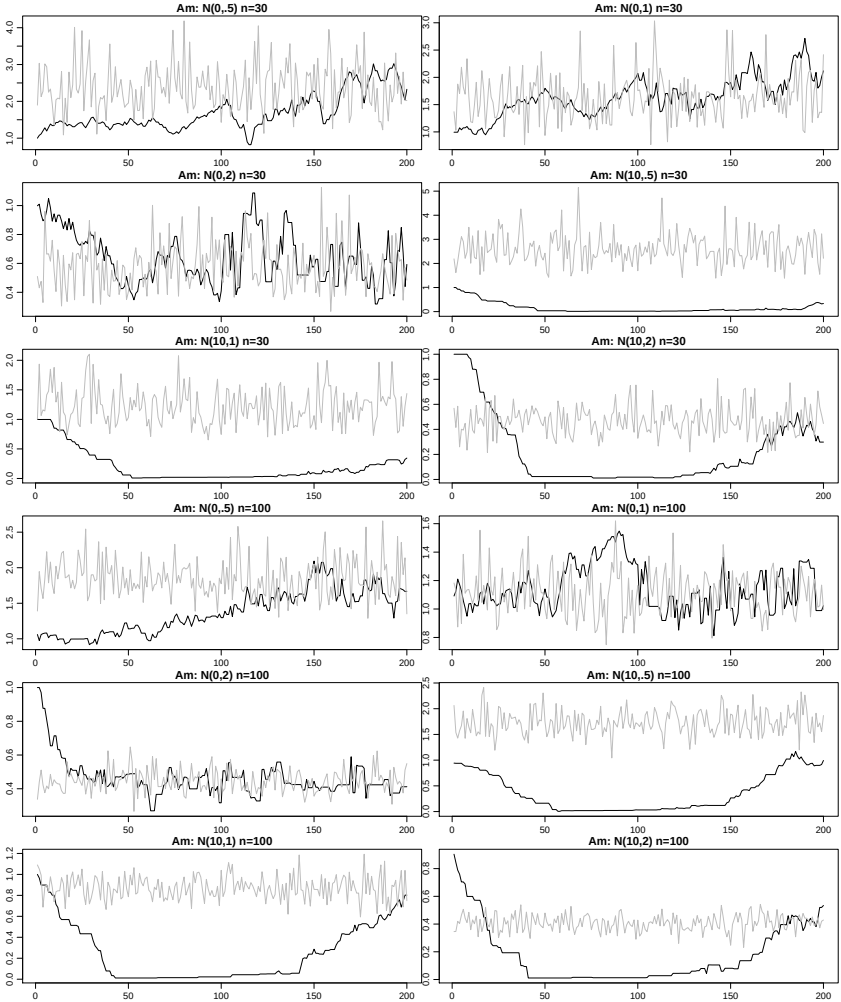

Figura 5.6: Primeiras 200 amostras da *posteriori* de σ^2, com Metropolis-Hastings (preto) e Gibbs (cinza), para cada um dos 12 conjuntos de dados simulados. Em todos os casos, o valor inicial de σ^2 foi 1.

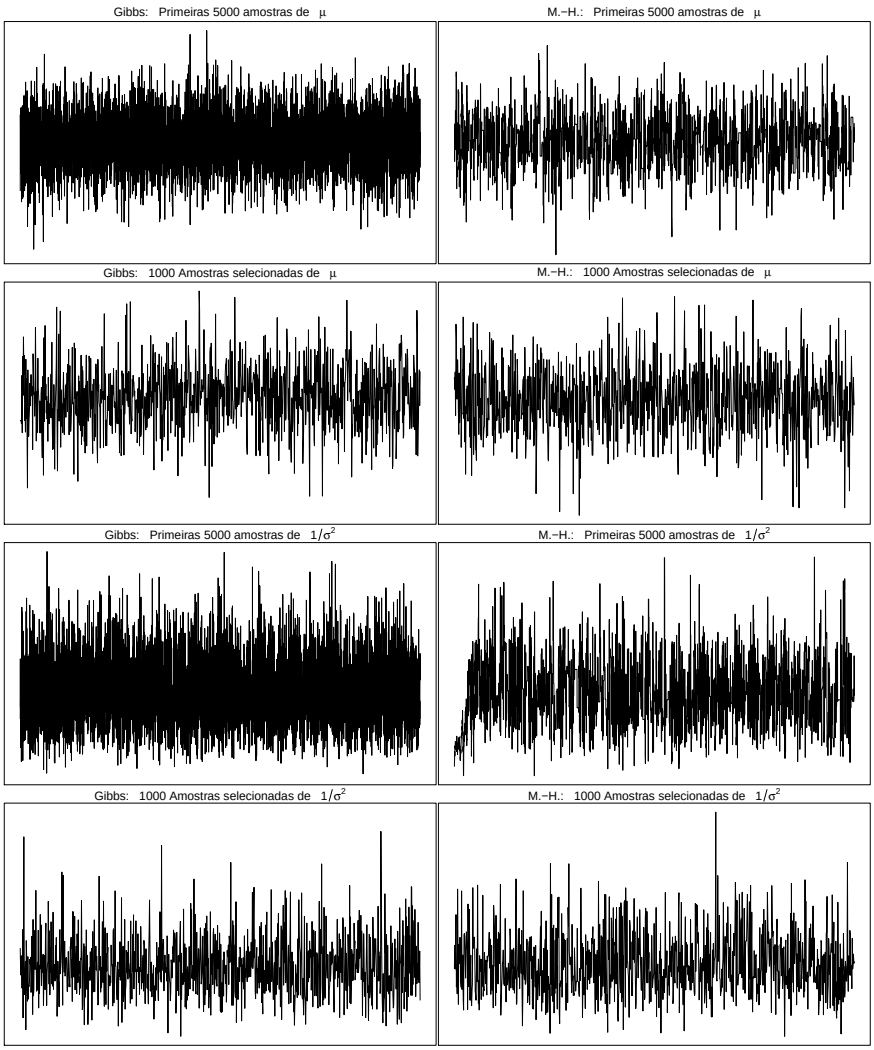

Figura 5.7: Traços das amostras de μ e $1/\sigma^2$ a *posteriori* para o primeiro conjunto de dados. Considerando os algoritmos de Gibbs e M-H, as primeiras 5000 amostras e as 1000 amostras selecionadas.

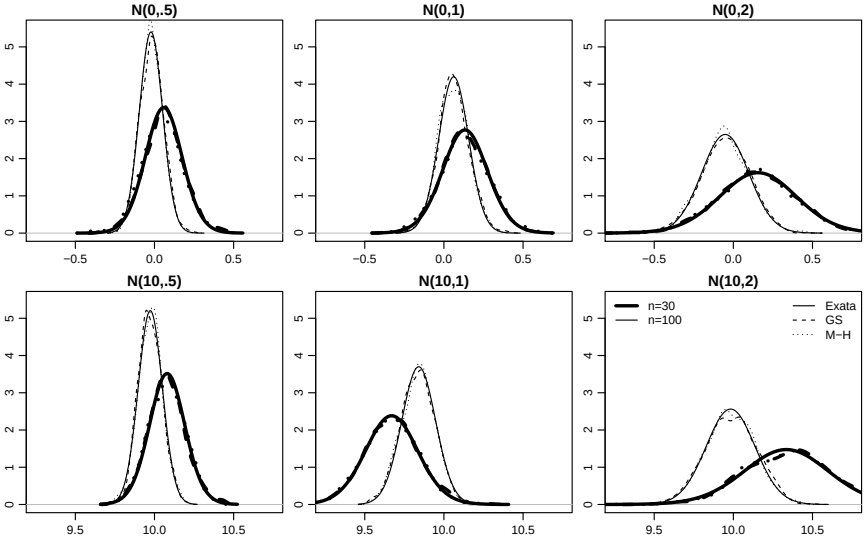

Figura 5.8: Densidades à *posteriori* de μ para os 12 conjuntos de dados.

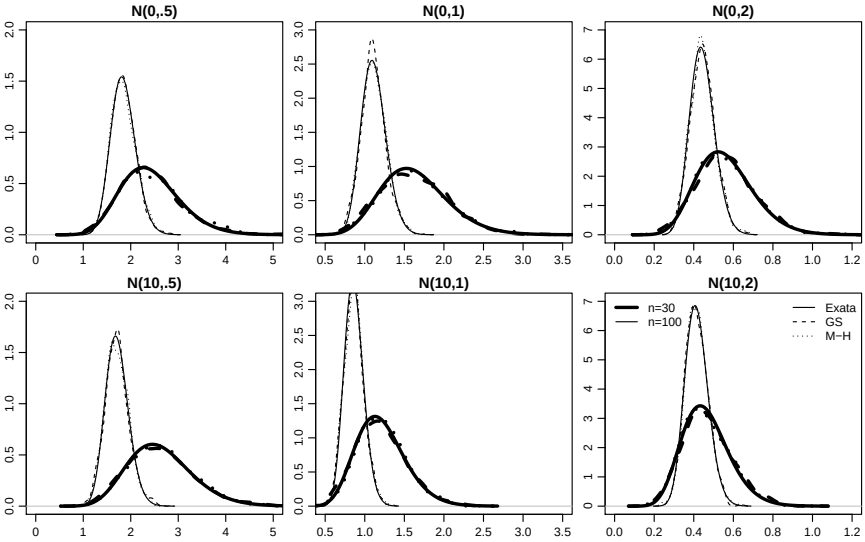

Figura 5.9: Densidades à *posteriori* de $1/\sigma^2$ para os 12 conjuntos de dados.

Tabela 5.4: Correlação entre valores sucessivos das 1000 amostras a extit-posteriori selecionadas.

	Gmu	Gphi	MHmu	MHphi
N(0,.5) n=30	0.0287	-0.0304	0.0261	0.0348
N(0,1) n=30	-0.0080	0.0112	0.0088	-0.0097
N(0,2) n=30	0.0190	-0.0274	-0.0135	0.0764
N(10,.5) n=30	0.0550	0.0486	0.0022	0.0291
N(10,1) n=30	-0.0398	0.0084	0.0350	0.1046
N(10,2) n=30	-0.0101	-0.0104	0.0238	0.0156
N(0,.5) n=100	-0.0181	-0.0046	0.0241	-0.0189
N(0,1) n=100	-0.0365	0.0238	0.0219	-0.0037
N(0,2) n=100	0.0064	-0.0169	0.0228	0.0782
N(10,.5) n=100	-0.0447	0.0504	0.0605	0.0213
N(10,1) n=100	0.0774	0.0251	0.0327	0.0393
N(10,2) n=100	0.0387	-0.0210	0.0371	-0.0481

intervalos HPD. Na Figura 5.10 temos os intervalos HPD (95%) para μ. O gráfico da esquerda é para as amostras com $\mu = 0$ e o da direita, $\mu = 10$. Em cada gráfico, foram plotados 6 grupos de intervalos, para cada uma das 6 amostras com a referida média. Em cada gráfico, os três primeiros grupos de intervalos são para amostras com $n = 30$ e os três últimos para amostras com $n = 100$. Notamos, como se espera, que para amostras $n = 100$ a amplitude dos intervalos é menor. Outro fato esperado é que a medida que a σ^2 aumenta a amplitude do intervalo também aumenta.

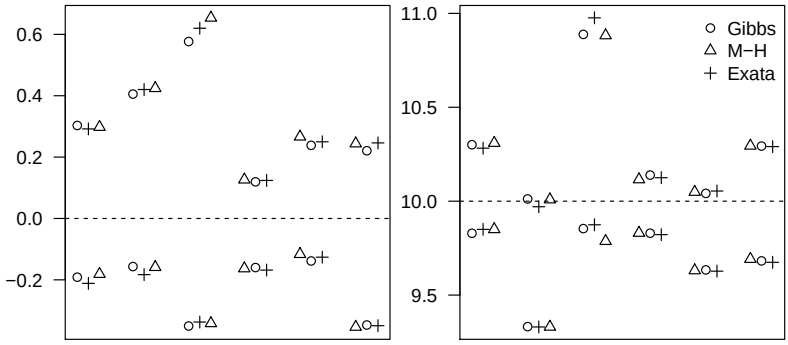

Figura 5.10: Intervalos HPD para $\mu = 0$ (esquerda) e $\mu = 10$ (direita). Em cada gráfico, os três primeiros grupos de IC são para $n = 30$ e os três últimos para $n = 100$

Na Figura 5.11 temos os intervalos HPD (95%) para $1/\sigma^2$. O gráfico da esquerda é para as amostras com $\sigma^2 = 0.5$, o do meio para $\sigma^2 = 1$ e o da direita, $\sigma^2 = 2$. Em cada gráfico, foram plotados 4 grupos de intervalos, para cada uma das 4 amostras com a referida variância. Em cada gráfico, os

dois primeiros grupos de intervalos são para amostras com $n = 30$ e os dois últimos para amostras com $n = 100$. Notamos, como se espera, que para amostras $n = 100$ a amplitude dos intervalos é menor. Também notamos que a média não afeta a amplitude do intervalo.

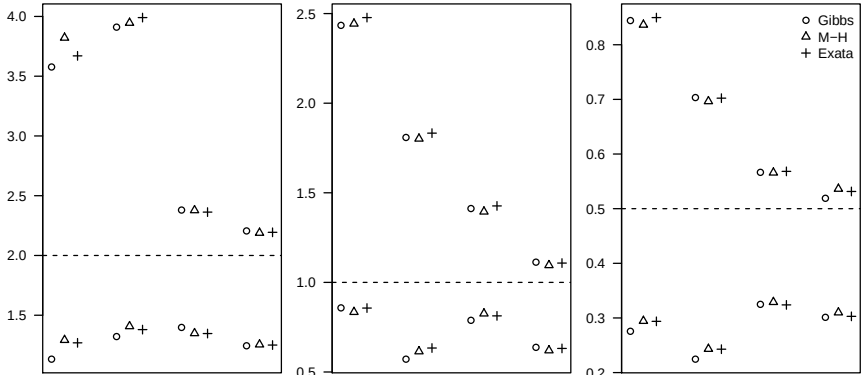

Figura 5.11: Intervalos HPD para $1/\sigma^2$ com $\sigma^2 = 0.5$ (esquerda), $\sigma^2 = 1$ (centro) e $\sigma^2 = 2$ (direita).

5.7 Revisitando exemplos

No Capítulo 2 apresentamos exemplos com estimação por máxima verossimilhança para algumas distribuições destacando alguns aspectos computacionais e formas da superfície de verossimilhança. Vamos agora revisitar alguns destes exemplos sob a perspectiva de inferência bayesiana e usando algoritmos para inferência por simulação ainda que resultados analíticos sejam disponíveis considerando as distribuições a *priori* usadas aqui. Ao invés de escrever funções para implementar algoritmos de MCMC, como na sessão anterior, vamos ilustrar o uso do pacote **MCMCpack**.

O pacote **MCMCpack** implementa algoritmos MCMC para modelos diversos. Além disso, possui uma função, `MCMCmetrop1R()`, que implementa o algoritmo Metropolis *random walk* multivariado de forma genérica. Como descrito na Seção 5.5.1, a idéia básica desse algoritmo é propor um novo valor, $\tilde{\theta}^{(i+1)}$, a partir do valor atual $\theta^{(i)}$, considerando um incremento ϵ.

A função `MCMCmetrop1R()` recebe como primeiro argumento uma função objetivo, que no nosso caso será a distribuição a *posterior* não normalizada, ou seja, simplesmente o produto da verossimilhança pela *priori*. Isto porque, como considerado na Seção 5.5.1, a constante normalizadora se cancela ao calcular a probabilidade de aceitação de $\tilde{\theta}^{(i+1)}$.

A função `MCMCmetrop1R()` adimite θ vetor, de dimensão p. Portanto, ϵ também o será. Neste caso, considera-se uma matriz de covariância, V, para ϵ, a covariância da distribuição proposta para simular θ. A função `MCMCmetrop1R()` permite que a matriz V seja informada. Porém, caso esta matriz não seja informada, será utilizada a função `optim()` para encontrar a moda da função objetivo e obter a hessiana, H ao redor dessa moda. Neste caso $V = tune(-1 * H)^{-1} tune$.

Após a definição de V, a função `MCMCmetrop1R()` usa uma função em C++ que implementa o algoritmo de Metropolis *random walk*. Esta função usa a biblioteca matricial Scythe, Pemstein et al. (2011), escrita em C++ para computação estatística. Inicialmente o algoritmo faz a decomposição de Choleski de V, $V = L'L$ e usa L para simular ϵ a cada iteração, isto é, $\epsilon = Lz$, em que z é um vetor com p amostras de uma distribuição Normal padrão.

O valor proposto $\tilde{\theta}^{(i+1)}$ é avaliado quanto a sua aceitação avaliando em ambiente C a função objetivo (*posteriori* não normalizada) passada pelo usuário no ambiente R. Para melhor eficiência, essa função deve ser definida na escala logarítmica.

Uma limitação desta função é quanto ao espaço paramétrico, pois o algoritmo de Metropolis *random walk* considera $\theta \in \Re^p$. Assim, para alguns modelos com componentes em θ com restrição, por exemplo, $\theta_j \in \Re^+$, usamos simular de $\theta_j^* = log(\theta_j)$ e fazer $\theta_j = e^{\theta_j^*}$.

5.7.1 Distribuição Normal

No exemplo da sessão 2.10 apresentamos a estimação por máxima verossimilhança do modelo gaussiano. Suponha que observamos uma amostra de tamanho n de $Y_i \sim N(\mu, \sigma^2)$, para $i = 1, \ldots, n$. Com esta suposição, temos a verossimilhança do modelo. Suponha que adotamos a *priori* que $\mu \sim N(0, 100)$, ou seja, com este valor elevado da variância a informação a *priori* a respeito de μ é vaga. Para o parâmetro σ^2, vamos atribuir $\sigma \sim G(1, 0.001)$ novamente uma *priori* bastante vaga.

Simulando dados:

```
set.seed(123)
y <- rnorm(100, mean=50, sd=10)
```

O próximo passo é definir uma função com a distribuição *a posteriori* não normalizada e na escala logarítmica

Código 5.96: *posteriori* (não normalizada) do modelo gaussiano.

```
posteriori <- function(par, y, log=TRUE){
  mu <- par[1] ; sigma <- par[2]
  res <- sum(dnorm(y, mean=mu,sd=sigma,log=log)) +
             dnorm(mu, mean=0, sd=10, log=log) +
             dgamma(sigma, 1, 0.001,log=log)
  return(ifelse(log, res, exp(res)))
}
```

Além da função objetivo, a função `MCMCmetrop1R()` tem como argumentos, um vetor de chutes iniciais, o *burnin*, o tamanho da cadeia MCMC, neste caso 20000, e *thin* ou salto que serve para economizar espaço em memória e também diminuir a autocorrelação dentro da cadeia, quando conveniente. A escolha do argumento *tune* é feita por tentativa e erro, ou seja, colocando um valor e observando uma taxa de aceitação muito alta, diminui-se esse valor. Mas, conforme ilustramos anteriormente, em algumas implementações de algoritmos, poderia-se implementar o ajuste automático com verificações à medida que a cadeia vai sendo gerada.

Na chamada a seguir aumentamos em cerca de 80% a variância obtida pela inversa de $-H$ e verificamos que a taxa de aceitação está entre 20% e 40%.

```
 require(MCMCpack)
 bayesN <- MCMCmetrop1R(posteriori, theta.init=c(20,5), burnin=500,
                        mcmc=20000, thin=10, y=y, tune=1.8)
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate was 0.34122
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
 summary(bayesN)
```

```
Iterations = 501:20491
Thinning interval = 10
Number of chains = 1
Sample size per chain = 2000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

       Mean     SD Naive SE Time-series SE
[1,] 50.473 0.8956  0.02003        0.02521
[2,]  9.243 0.6686  0.01495        0.01906

2. Quantiles for each variable:

       2.5%    25%    50%    75% 97.5%
var1 48.700 49.915 50.489 51.080 52.20
var2  8.029  8.777  9.205  9.682 10.68
```

A saída é um objeto da classe `mcmc` já adequado para diagnósticos de convergência com o pacote **coda**. Assim as médias a *posteriori* e intervalos de credibilidade baseados em quantis, são facilmente obtidas. Podemos

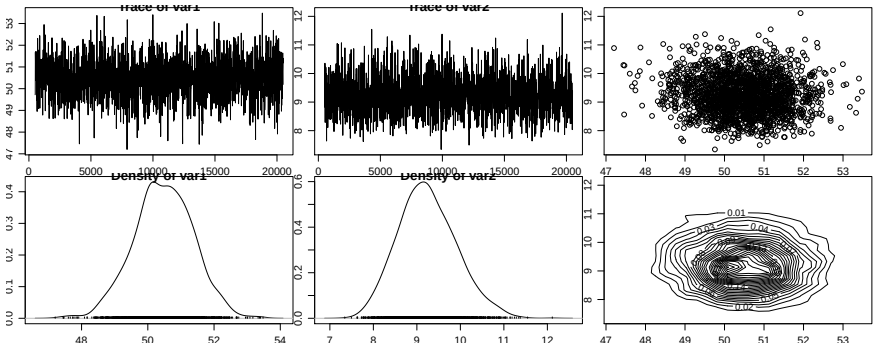

Figura 5.12: Traço das cadeias e *posteriori*'s suavizadas para μ (esquerda) e σ (centro), diagrama de dispersão das amostras e contornos da densidade conjunta (direita) - modelo gaussiano.

visualizar o traço das cadeias e o gráfico das densidades suavizadas a *posteriori* com as cadeias geradas. De forma geral, quando nenhum padrão é claro no traço das cadeias significa que o algoritmo convergiu mas testes der convergência mais cuidadosos devem ser efetuados.

A Figura 5.12 resume os resultados e o diagrama de dispersão e densidade do contornos da conjunta refletem a ortogonalidade na verossimilhança.

5.7.2 Distribuição Gama

Outro modelo analisado por verossimilhança foi o modelo Gama com parâmetros a e s, Sessão 2.11 . Vamos ilustrar a inferência para o mesmo modelo pelo método bayesiano. Para isto, seja $Y_i \sim G(a,s)$ com $i = 1, \ldots, n$. Adotamos as seguintes *priori*'s $a \sim G(1,0.1)$ e $s \sim G(1,0.1)$.

A distribuição a *posteriori* não normalizada é especificada no Código 5.97. Para assegurar valores no espaço paramétrico escrevemos a função utilizando o logaritmo dos parâmetros. Ou seja, o algoritmo de Metropolis irá trabalhar no logaritmo de ambos os parâmetros. Assim, ambos os parâmetros serão simulados na escala logarítmica. Em inferência por simulação este artifício não traz dificuldades adicionais uma vez que se temos uma amostra de θ e queremos uma amostra de $g(\theta)$, basta aplicar $g(\cdot)$ aos elementos da amostra de θ.

Código 5.97: (log) Densidade a *posteriori* para o modelo Gamma.

```
post.gama <- function(par,y){
  a <- exp(par[1]) ; s <- exp(par[2])
  return(sum(dgamma(y, shape=a, scale=s,log=TRUE)) +
         dgamma(a, shape=1, scale=100, log=TRUE) +
         dgamma(s, shape=1, scale=100,log=TRUE))
}
```

De forma análoga ao exemplo anterior, simulamos dados e uma amostra dos parâmetros deste modelo via MCMC pelo código a seguir.

```
set.seed(123)
y100 <- rgamma(100, shape=10, scale=5)
bayesG <- exp(MCMCmetrop1R(post.gama,theta.init=c(1,1), burnin=5000,
                           mcmc=20000, thin=10, y=y100, tune=1.8))
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate was 0.33516
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Note que exponenciamos o resultado da função `MCMCmetrop1r()`,

```
summary(bayesG)
```

```
Iterations = 5001:24991
Thinning interval = 10
Number of chains = 1
Sample size per chain = 2000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

       Mean     SD Naive SE Time-series SE
[1,] 13.373 1.8123  0.04052        0.03955
[2,]  3.733 0.5334  0.01193        0.01147

2. Quantiles for each variable:

       2.5%    25%   50%    75%  97.5%
var1 10.067 12.088 13.31 14.562 17.206
var2  2.846  3.349  3.67  4.056  4.957
```

Na Figura 5.13 podemos novamente ver as distribuições a *posteriori* marginais e conjunta o traço das cadeias. Os contornos da densidade conjunta novamente refletem a forma da verossimilhança sobre esta parametrização.

Existem vantagens computacionais e de convergência em parametrizações adequadas. A ortogonalidade permite uma varredura mais eficiente do espaço paramétrico. Nesta situação, poderia-se até considerar V diagonal. Além disso, se a ordem de magnitude dos valores dos parâmetros deve ser próxima, o tamanho dos "passos"em cada dimensão também ficam próximos. Esta condição adicional permitiria a diagonal de V constante. Essas duas considerações permite considerar simplificações no Metropolis *random walk*.

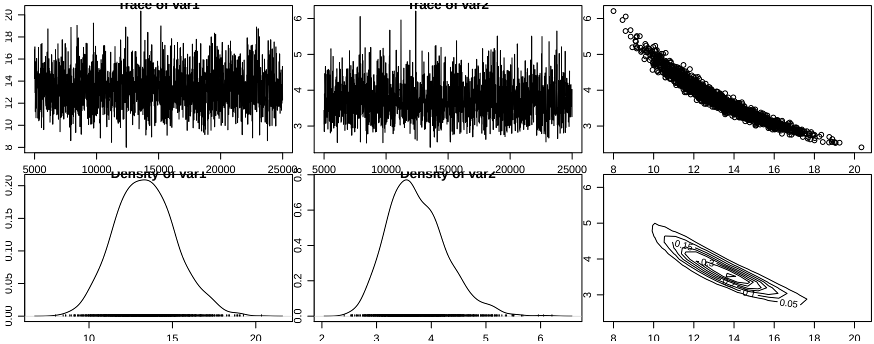

Figura 5.13: Traço das cadeias *posteriori*'s marginais suavizadas para a (esquerda) e s (centro), *posteriori* conjunta (direita) - modelo Gama.

Vamos reimplementar o exemplo considerando a reparametrização 3, apresentada no exemplo em 2.11 que tem a propriedade de ortogonalidade considerando ainda a reparametrização utilizando o logaritmo. Podemos usar a mesma estrutura da função anterior, simplesmente recebendo os parâmetros na nova parametrização e transformando internamente para avaliar a *posteriori*. que fica definida da forma a seguir. Note entretanto que há a necessidade de se redefinir a *priori* em uma escala compatível para os novos parâmetros. Aqui como $mu = a * s$, a *priori* para μ seria o produto de duas densidades Gama.

Código 5.98: Densidade a *posteriori* - modelo Gama na parametrização 3.

```
post.gama.3 <- function(par, y) {
    a <- exp(par[1])
    mu <- exp(par[2])
    return(sum(dgamma(y, shape = a, scale = mu/a, log = TRUE)) +
        dgamma(a, shape = 1, scale = 100, log = TRUE) + dgamma(a,
        shape = 1, scale = 100, log = TRUE) + dgamma(mu/a,
        shape = 1, scale = 100, log = TRUE))
}
```

Procedemos o ajuste via MCMC como no exemplo anterior.

```
bayesG.3 <- MCMCmetrop1R(post.gama.3, theta.init = c(1, 1),
    burnin = 5000, mcmc = 20000, thin = 10, y = y100, tune = 1.8)
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate was 0.33864
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Na Figura 5.14 mostramos os traços das cadeias e os gráficos das posteriori's marginais e conjunta.

Caso deseje-se as *posteriori*'s na parametrização original, transformar os valores simulados e obter resultados da forma usual com comandos como

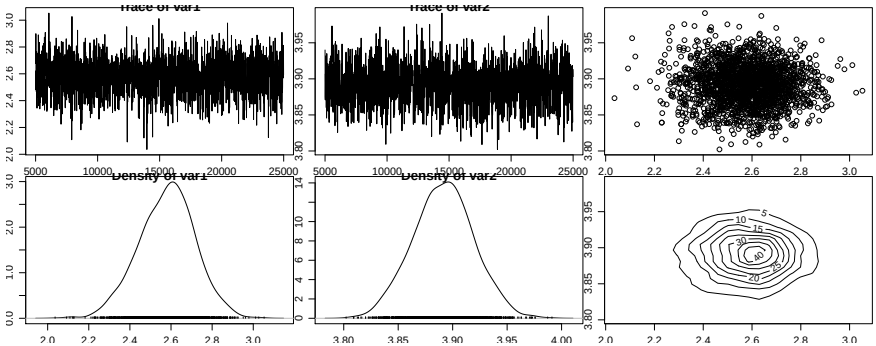

Figura 5.14: Traço das cadeias e *posteriori*'s suavizadas para $\log(a)$ (esquerda) e $\log(\mu)$ (centro), e *posteriori* conjunta (direita) - modelo Gama reparametrizado.

a seguir.

```
 post.as <- as.mcmc(cbind(a=exp(bayesG.3[,1]),
                          s= exp(bayesG.3[,2]-bayesG.3[,1])))
 summary(post.as)
```

```
Iterations = 1:2000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 2000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

   Mean     SD Naive SE Time-series SE
a 13.31 1.8028  0.04031        0.04728
s  3.75 0.5346  0.01195        0.01398

2. Quantiles for each variable:

   2.5%    25%    50%    75%  97.5%
a 10.02 12.056 13.288 14.479 17.113
s  2.85  3.374  3.693  4.073  4.917
```

Comparando os intervalos com ambas parametrizações

```
 HPDinterval(bayesG)
```

```
       lower     upper
var1 9.777995 16.861214
var2 2.777156  4.789711
attr(,"Probability")
[1] 0.95
```

```
 HPDinterval(post.as)
```

```
     lower     upper
a 9.834988 16.877757
s 2.743589  4.789399
attr(,"Probability")
[1] 0.95
```

notamos que são similares.

Como fica claro nesta Seção, sob o enfoque de bayesiano e inferência por simulação não nos preocupamos com obtenção de derivadas, em saber se aproximações são razoáveis, ou outros procedimentos que discutimos na análise de verossimilhança Em inferência bayesiana via simulação como os exemplos apresentados aqui, assimetrias são naturalmente capturadas nas distribuições a *posteriori*. Além disso, reparametrização é trivial, não requer nada além de a função de reparametrização nas amostras da *posteriori* para obter a *posteriori* sob a reparametrização desejada. Uma vez, obtida as amostras da *posteriori* a inferência se resume a extrair medidas descritivas nas amostras. Isso justifica a grande atratividade e flexibilidade da abordagem bayesiana e de inferência por simulação. A parte subjetiva é a escolha das *priori*'s que nos nossos exemplos foram escolhidas sem nenhum cuidado adicional, embora o tópico de elicitação de *priori*'s seja fundamental em inferência bayesiana. É claro, que apresentamos exemplos muito simples, em modelos mais complicados a escolha das *priori*'s pode ser decisiva para o sucesso dos algoritmos MCMC. Da mesma forma algoritmos podem se beneficiar de parametrizações mais adequadas. Assim como na análise de verossimilhança, a ortogonalidade provê melhores propriedades para os algoritmos. Resultados analíticos como gradientes podem ser usados em algoritmos MCMC de melhor performance, não discutidos aqui.

Deixamos para o leitor implementar os demais modelos e exemplo apresentados no Capítulo 2.

5.8 Inferência Bayesiana em modelos de regressão

No Capítulo 3 apresentamos os modelos de regressão e entre eles implementamos dois exemplos, regressão de Poisson e regressão Simplex. Revisitamos agora estes modelos com análises sob o enfoque bayesiano, com inferência por simulação. Usualmente em modelos de regressão, dividimos o vetor de parâmetros em uma parte referente à média do modelo e uma segunda parte referente aos parâmetros de variabilidade/dispersão. Uma vez definido o modelo a verossimilhança já esta implícita, para as priori's em geral para a parte de média é uma comum atribuir priori's gaussianas com média zero e variância grande. Para parâmetros de dispersão é comum usar distribuições Gama pouco informativas. Começamos por um modelo que só tem parâmetros de média. Assim como na sessão anterior utilizamos o pacote **MCMCpack**.

5.8.1 Regressão Poisson

Considere $Y_i \sim P(\lambda_i)$ com $i = 1, \ldots, n$. Suponha ainda que $\lambda_i = \exp(\beta_0 + \beta_1 x_i)$ com $beta_0$ e β_1 parâmetros de regressão sobre os quais deseja-se fazer inferência e x_i uma covariável conhecida. Isto define um modelo de *regressão Poisson*. Para completar a especificação bayesiana deste modelo suponha prioris independentes $[\beta_i] \sim N(0, 100)$. Com isso, temos especificados todos elementos do modelo bayesiano, definimos uma função com a distribuição a posteriori não normalizada.

Código 5.99: Densidade à *posteriori* para o modelo de regressão de Poisson.

```
postPoisReg <- function(par, formula, dados){
  mf <- model.frame(formula, dados)
  y <- model.response(mf)
  X <- model.matrix(formula, mf)
  lambda <- exp(drop(X%*%par))
  return(sum(dpois(y, lambda=lambda, log=TRUE)) +
         sum(sapply(par, dnorm, mean=0, sd=10,log=TRUE)))
}
```

Usando a mesma amostra simulada na sessão 3.1 podemos proceder com a estimação, usando o pacote **MCMCpack**.

```
PoisReg <- MCMCmetrop1R(postPoisReg, theta.init=c(1,1), burnin=5000,
                mcmc=25000, thin=10, dados=dados10, form=y~cov, tune=1.8)
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate was 0.32927
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
summary(PoisReg)
```

```
Iterations = 5001:29991
Thinning interval = 10
Number of chains = 1
Sample size per chain = 2500

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

       Mean      SD  Naive SE Time-series SE
[1,] 2.2198 0.14366 0.0028731      0.0028512
[2,] 0.4294 0.03822 0.0007644      0.0007612

2. Quantiles for each variable:

       2.5%    25%    50%    75%  97.5%
var1 1.9260 2.1242 2.2244 2.3169 2.4871
var2 0.3561 0.4032 0.4282 0.4547 0.5061
```

Na Figura 5.15 são mostrados os traços das cadeias e as distribuições a posteriori para cada parâmetro bem como o diagrama de dispersão e a

densidade suavizada da conjunta que mostram a forte relação (não ortogonalidade entre os parâmetros).

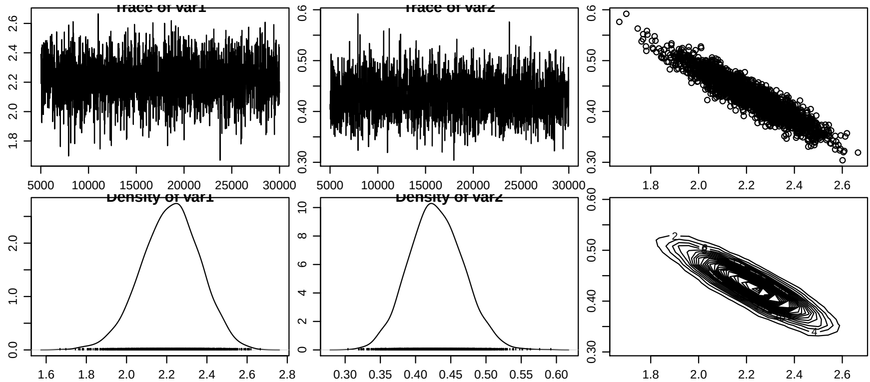

Figura 5.15: Traço das cadeias e distribuições a posteriori suavizadas para β_0 (esquerda), β_1 (centro) e amostras e densidade conjunta empírica (direita) - Regressão Poisson.

Podemos ver na Figura 5.15 que os parâmetros são não ortogonais. Fazendo uma pequena modificação na forma de declaração do modelo, podemos fazer com que a relação entre estes parâmetros seja um pouco mais próxima da ortogonalidade. Esta reparametrização consiste em apenas centrar ou padronizar a covariável. Veja a seguir como fica o ajuste. Verifica-se na Figura 5.16 que a posteriori conjunta mostra um comportamento mais próximo da ortogonalidade, amenizando a correlação observada na Figura 5.15.

```
PoisReg2 <- MCMCmetrop1R(postPoisReg, theta.init=c(1,1), burnin=500,
                         mcmc=20000, thin=10, dados=dados10,
                         form= y~scale(cov), tune=1.8)
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate was 0.33234
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
summary(PoisReg2)
```

```
Iterations = 501:20491
Thinning interval = 10
Number of chains = 1
Sample size per chain = 2000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

       Mean      SD Naive SE Time-series SE
[1,] 3.2948 0.06609 0.001478       0.001820
[2,] 0.7207 0.06567 0.001469       0.001788

2. Quantiles for each variable:
```

```
      2.5%    25%    50%    75%  97.5%
var1 3.169 3.2491 3.2965 3.3407 3.4232
var2 0.594 0.6771 0.7191 0.7638 0.8507
```

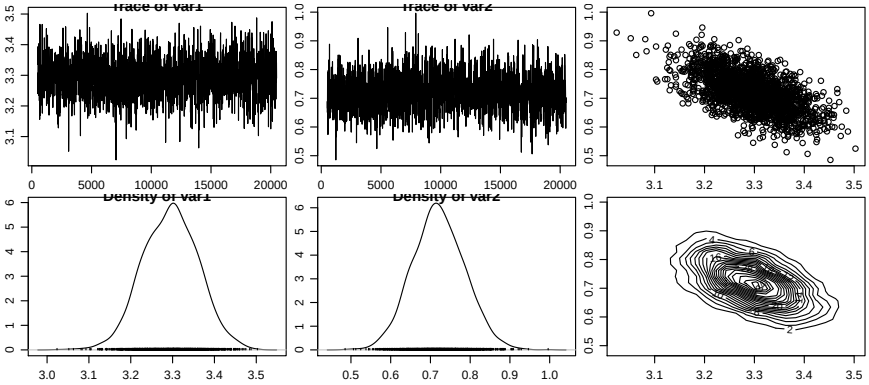

Figura 5.16: Traço das cadeias e distribuições a posteriori suavizadas para β_0 (esquerda), β_1 (centro) e amostras e densidade conjunta empírica (direita) com covariável centrada - Regressão Poisson.

5.8.2 Regressão Simplex

Na Seção 3.2 vimos o modelo de regressão Simplex como um possível modelo para variáveis resposta restritas ao intervalo unitário. Como exemplo de aplicação utilizamos os dados do Índice de Qualidade de Vida de Curitiba (IQVC), e o interesse principal é verificar a relação entre o IQVC e a renda média do bairro. O modelo supõe que $Y_i \sim S(\mu_i, \sigma^2)$, para $i = 1, \ldots, n$. Além disso, $\text{logit}(\mu_i) = \beta_0 + \beta_1 renda_i$. Este modelo têm três parâmetros, ou seja, $\underline{\theta} = (\beta_0, \beta_1, \sigma^2)$. Para completar a especificação bayesiana, designamos que $[\beta_i] \sim N(0{,}100)$, independentes. Lembramos que neste modelo é possível obter o parâmetro σ^2 como uma função dos parâmetros de média, o que permitia escrever a verossimilhança concentrada. No caso de algorítmos MCMC podemos usar este e diminuir a dimensão do problema, ou seja, podemos rodar a cadeia somente em função dos parâmetros de média, e após obter a amostra final (depois de *burn-in* e *thinning*) usamos os valores para obter amostras de σ^2. Isto reduz substancialmente o custo computacional do algoritmo MCMC. Com isso, definimos uma função com a distribuição a posteriori não escalonada de interesse como se segue.

Usando os dados do Índice de Qualidade de Vida de Curitiba, podemos seguir com a estimação Bayesiana dos parâmetros do modelo de regressão Simplex. Novamente usamos o pacote **MCMCpack**.

Código 5.100: Densidade à *posteriori* para o modelo de regressão de Simplex.

```
postSimplex <- function(par, formula, data){
  mf <- model.frame(formula, data)
  y <- model.response(mf)
  X <- model.matrix(as.formula(formula), mf)
  mu <- inv.logit(X%*%par)
  d0 = (y - mu)^2 / (y * (1-y) * mu^2 * (1-mu)^2)
  sigma <- sum(d0)/length(y)
  post <- sum(dsimplex(y, mu=mu,sigma=sigma)) +
          sum(sapply(par, dnorm, mean= 0, sd=10, log=TRUE))
  return(post)
}
```

```
dados <- read.table("simplex.txt", header=TRUE)
bayesSimplex <- MCMCmetrop1R(postSimplex, theta.init=c(0,0),
                     burnin=500, mcmc=20000, thin=10,
                     formula=y~MEDIA, data=dados, tune=1.8)
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate was 0.34171
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
summary(bayesSimplex)
```

```
Iterations = 501:20491
Thinning interval = 10
Number of chains = 1
Sample size per chain = 2000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

        Mean      SD Naive SE Time-series SE
[1,] -0.4090 0.11476 0.002566       0.003377
[2,]  0.6851 0.07087 0.001585       0.002043

2. Quantiles for each variable:

        2.5%     25%     50%     75%   97.5%
var1 -0.6287 -0.4870 -0.4087 -0.3334 -0.1808
var2  0.5446  0.6374  0.6875  0.7333  0.8175
```

Novamente plotando os traços das cadeias e as distribuições a posteriori.

Obtemos a posteriori para o parâmetro σ^2 usando as amostras simuladas de β_0 e β_1. Basta calcular σ^2 para cada dupla de β's, veja o código abaixo:

```
X <- model.matrix(~MEDIA, data=dados)
mu <- inv.logit(X %*% t(bayesSimplex))
d0 = with(dados, (y - mu)^2 / (y*(1-y)*mu^2 * (1-mu)^2))
sigma <- mcmc(data=colSums(d0)/length(dados$y))
```

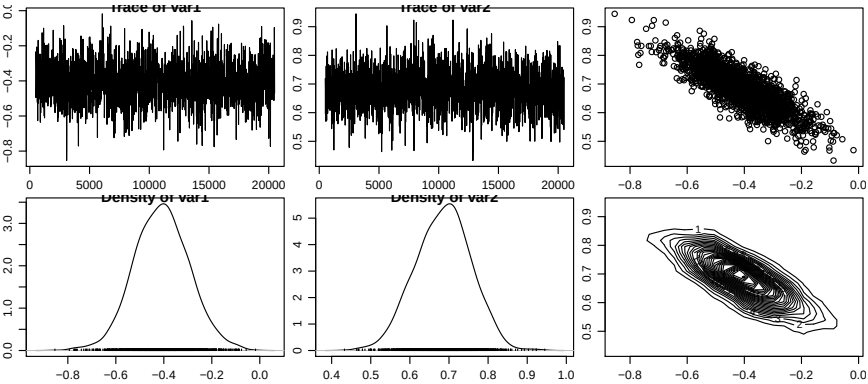

Figura 5.17: Traço das cadeias e distribuições a posteriori suavizadas para β_0 (esquerda), β_1 (centro) e amostras e densidade conjunta empírica (direita) - Regressão Simplex.

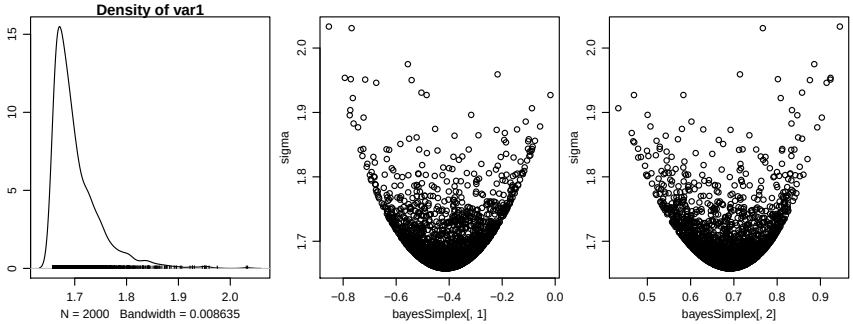

Figura 5.18: Distribuições a posteriori suavizadas σ^2 (esquerda) e diagrams de dispersão contra demais parâmetros (centro e direita) - Regressão Simplex.

Visualizando a cadeia simulada e a distribuição a posteriori de σ^2.

A posteriori mostra uma forte assimetria, o que caso estivéssemos estimando via verossimilhança implicaria que os intervalos assintóticos seriam inadequados. Os diagramas de dispersão contra os demais parâmetros mostram a severa não ortogonalidade, o que, neste caso reforça a vantagem em se proceder a marginalização implementando o algorítimo MCMC apenas nos parâmetros de média.

Para obter o perfil de verossimilhança deste parâmetro é necessário um esforço computacional bastante elevado. Em comparação, via inferência bayesiana apenas aplicamos uma conta simples e obtemos um intervalo de credibilidade bastante assimétrico e consequentemente mais real para este parâmetro.

5.9 Inferência Bayesiana para modelos de regressão com efeitos aleatórios

No Capítulo refcap:ranef descrevemos a forma geral de modelos de regressão com efeitos aleatórios. Nesta classe de modelos surgem limitações no uso de inferência via verossimilhança à medida que aumenta o número de parâmetros, em particular para distribuições não gaussianas. A principal limitação está no custo computacional e complexidade em resolver as integrais para obter a verossimilhança marginal que, em geral, deve ser maximizada numericamente. Além disso, surgem dificuldades para a construção de intervalos de confiança para os parâmetros de variância que estão presentes em maior quantidade nesta classe de modelos. As aproximações assintóticas são de forma geral inadequadas por impor simetria, o que em geral não é razoável para os parâmetros associados os efeitos aleatórios enquanto que métodos computacionais podem se tornar proibitivos.

Nesta seção vamos ver que no contexto de inferência bayesiana é possível contornar mais facilmente a complexidade em trabalhar com efeitos aleatórios e a inferência continua sendo factível mesmo em situações bastante complexas. Vamos ilustrar os elementos da implementação bayesiana revisitando os modelos de Poisson com intercepto aleatório e efeito aninhado apresentados anteriormente, e também analisando o modelo Beta longitudinal. Para estes modelos optamos por fazer a implementação utilizando o programa `JAGS` já que este programa é bem mais geral e eficiente do que nossos códigos ilustrativos, permitindo análises de modelos mais complexos. Embora seja um aplicativo isolado, o `JAGS` pode ser usado a partir de uma sessão do R. Para isto pode-se usar funções básicas do R como `sink()`, `write()`, `system()` e `scan()` para escrever, acionar o `JAGS` e importar os resultados. Entretanto, há pacotes no R que facilitam a interação com o `JAGS`. Dois deles são o **rjags** e o pacote **dclone**. O primeiro é muito recomendado e criado pelo mesmo autor do `JAGS` (Martin Plummer). Entretanto vamos utilizar o segundo já que retornaremos a utilizá-lo no Capítulo 6.

5.9.1 Poisson com intercepto aleatório

Na Seção 4.4 apresentamos análises de verossimilhança para o modelo de Poisson com intercepto aleatório definido em 4.8 como uma forma de tratar contagens que apresentam superdispersão. Este modelo tem apenas dois parâmetros $\underline{\theta} = (\beta_0, \tau^2)$. Para completar a especificação bayesiana adicionamos à especificação do modelo as *priori*'s $[\beta_0] \sim N(0{,}100)$ e $[\tau^2] \sim G(1/2{,}1/2)$.

Para processar as análises devemos começar definindo o modelo segundo a linguagem `JAGS` dentro uma função do R como no código 5.101.

Com esta função e os objetos de dados e as definições para a cadeia utilizamos a função `jags.fit()` do pacote **dclone** que passa todas as informações para uma chamada do `JAGS` no sistema. As cadeias geradas retornam diretamente para o R em um objeto da classe `mcmc`, já adequado para a análise de convergência das *posteriori*'s usando recursos do pacote **coda**. Na definição do modelo é importante notar também que o `JAGS` parametriza a distribuição gaussiana com parâmetro de precisão (inverso do desvio padrão). Por exemplo, no código 5.101 a declaração `beta0 ~dnorm(0, 0.001)` está definindo uma *priori* de média zero e desvio padrão $1000 = 1/0{,}0001$.

Código 5.101: Função em linguagem `JAGS` da densidade à *posteriori* para o modelo de regressão Poisson com efeito aleatório de intercepto.

```
mod.poisson <- function(){
  for(j in 1:n.ua){
        for(i in 1:n.rep){
                Y[i,j] ~ dpois(lambda[i,j])
                log(lambda[i,j]) <- beta0 + b[j]
        }
        b[j] ~ dnorm(0, tau2)
  }
  beta0 ~ dnorm(0, 0.001)
  tau2 ~ dgamma(0.5, 0.5)
}
```

Note que escrevemos o modelo em termos de uma variável resposta Y, além disso, o `for()` é especificado sobre `n.ua` que é a quantidade de unidades amostrais, e `n.rep` que é a quantidade de repetições dentro de cada unidade amostral. Estes valores devem ser informados para a função `jags.fit()`. Estas informações são passadas por uma **lista nomeada**, como ilustrado na chamada de código a seguir. Outro fato importante é a forma matricial em que o modelo é definido, o que requer que os dados sejam armazenados como uma matriz. Os demais argumentos da função são um vetor com os nomes dos parâmetros a serem monitorados (para os quais as cadeias serão retornadas), a função com o modelo, e os parâmetros que controlam a cadeia. No exemplo definimos em `n.chains` que cinco cadeias independentes devem ser produzidas.

```
require(dclone)
y.mat <- matrix(dados$y, 10, 10)
dados.list <- list(Y = y.mat, n.ua = 10, n.rep = 10)
bayesPois <- jags.fit(dados.list,c("beta0","tau2"),mod.poisson,
                      n.adapt = 1000, n.update = 1000, thin = 5,
                      n.iter=10000, n.chains=5)

summary(bayesPois)
```

```
Iterations = 2005:12000
Thinning interval = 5
Number of chains = 5
Sample size per chain = 2000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

       Mean    SD Naive SE Time-series SE
beta0 1.991 0.140  0.00140       0.004573
tau2  6.530 3.023  0.03023       0.037248

2. Quantiles for each variable:

       2.5%   25%   50%   75%  97.5%
beta0 1.706 1.902 1.993 2.080  2.265
tau2  2.031 4.312 6.066 8.233 13.715
```

A saída da função `jags.fit()` é um objeto da classe `mcmc`, portanto todas as funcionalidades (métodos) desta classe estão disponíveis para este objeto, de forma similar à saída da função `MCMCmetrop1R()` que usamos anteriormente. Na Figura 5.19 mostramos o traço das cadeias e a distribuição a *posteriori* suavizada. Note a forte assimetria para o parâmetro de precisão, muito comum nesta classe de modelos.

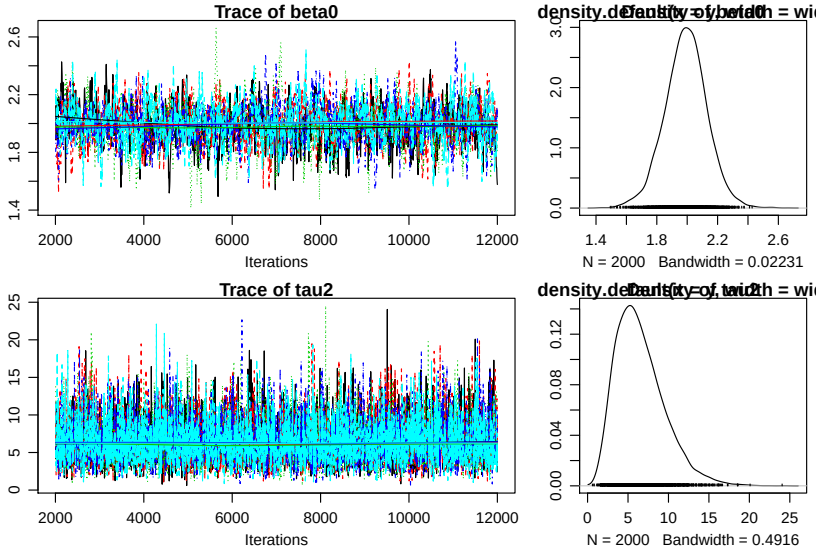

Figura 5.19: Traço das cadeias e distribuições a *posteriori* suavizadas - Poisson com intercepto aleatório.

5.9.2 Poisson com efeito aninhado

O modelo Poisson com efeito aninhado discutido na sessão 4.5, já apresentava dificuldades para estimação por máxima verossimilhança, uma vez que a integral contida na verossimilhança tem dimensão 6. O modelo segundo a notação que utilizaremos aqui define que $Y_{ij} \sim P(\lambda_{ij})$ com $log(\lambda_{ij}) = \beta_0 + b_i + b_{i:j}$. Supomos para os efeitos aleatórios $b_i \sim N(0, 1/\sigma^2)$ e $b_{i:j} \sim N(0,1/\tau^2)$. Este modelo possui três parâmetros, $\underline{\theta} = (\beta_0, \sigma^2, \tau^2)$. Para completar a especificação bayesiana, atribuímos as *priori*'s $\beta_0 \sim N(0, 100^2)$, $\sigma^2, \tau^2 \sim G(1/2, 1/2)$. A função do R que define o modelo em sintaxe `JAGS` é definida no código 5.102.

Código 5.102: Função em linguagem `JAGS` da densidade à *posteriori* para o modelo de regressão Poisson com efeito aleatório aninhado de intercepto.

```
poisson.aninhado <- function(){
  for(j in 1:n.bloco){
      for(i in 1:n.rep){
        Y[j,i] ~ dpois(lambda[j,i])
        log(lambda[j,i]) <- beta0 + Z.int[j,i]*b0[j] +
             Z.rep1[j,i]*b1[j] + Z.rep2[j,i]*b2[j] +
             Z.rep3[j,i]*b3[j] + Z.rep4[j,i]*b4[j] +
             Z.rep5[j,i]*b5[j]
      }
      b0[j] ~ dnorm(0,sigma2)
      b1[j] ~ dnorm(0,tau2)
      b2[j] ~ dnorm(0,tau2)
      b3[j] ~ dnorm(0,tau2)
      b4[j] ~ dnorm(0,tau2)
      b5[j] ~ dnorm(0,tau2)
  }
  beta0 ~ dnorm(0, 0.01)
  sigma2 ~ dgamma(1/2, 1/2)
  tau2 ~ dgamma(1/2, 1/2)
}
```

No código a seguir preparamos o conjunto de dados para ser passado ao `JAGS`, o que deve ser feito com atenção à forma adequada. No exemplo utilizamos o mesmo conjunto de dados utilizado na sessão 4.5.

```
dados$UM <- 1
dados.id <- split(dados, dados$ID)
Z <- model.matrix(~ UM + bloco - 1, data=dados.id[[1]])
Y.mat <- matrix(NA, ncol=4, nrow=20)
for(i in 1:4){
  Y.mat[,i] <- dados.id[[i]]$y
}
Z.int  <- t(matrix(Z[,1], ncol=4, nrow=20))
```

```
Z.rep1 <- t(matrix(Z[,2], ncol=4, nrow=20))
Z.rep2 <- t(matrix(Z[,3], ncol=4, nrow=20))
Z.rep3 <- t(matrix(Z[,4], ncol=4, nrow=20))
Z.rep4 <- t(matrix(Z[,5], ncol=4, nrow=20))
Z.rep5 <- t(matrix(Z[,6], ncol=4, nrow=20))
dat.jags <- list(Y=t(Y.mat), Z.int=Z.int, Z.rep1=Z.rep1, Z.rep2=Z.rep2,
       Z.rep3=Z.rep3, Z.rep4=Z.rep4, Z.rep5=Z.rep5, n.bloco=4,n.rep=20)
```

Para fazer o ajuste usando a função `jags.fit()` com a chamada a seguir. Mostramos ainda um resumo do ajuste retornado por `summary()` e na Figura 5.20 podemos visualizar o traço das cadeias simuladas e as distribuições a *posteriori* suavizadas.

```
poisson.ani <- jags.fit(dat.jags,c("beta0","sigma2","tau2"), poisson.aninhado,
             n.adapt=1000, n.update=250, thin=10, n.iter=15000, n.chains=3)
```

```
summary(poisson.ani)
```

```
Iterations = 1260:16250
Thinning interval = 10
Number of chains = 3
Sample size per chain = 1500

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

        Mean     SD Naive SE Time-series SE
beta0  2.846 0.7983  0.01190        0.05881
sigma2 2.004 1.7322  0.02582        0.07432
tau2   3.383 1.2092  0.01803        0.02837

2. Quantiles for each variable:

         2.5%    25%   50%   75% 97.5%
beta0  0.8899 2.6147 3.084 3.352 3.912
sigma2 0.1229 0.7107 1.564 2.761 6.317
tau2   1.4812 2.4937 3.234 4.114 6.132
```

5.9.3 Modelo Beta longitudinal

Revisitamos agora o exemplo na sessão 4.6 sob o enfoque bayesiano. Utilizamos este modelo para analisar uma variável resposta restrita ao intervalo $(0,1)$ que é observada ao longo de tempo. O modelo Beta longitudinal adotado utiliza efeitos aleatórios para induzir estruturas de dependência para observações ao longo do tempo tomadas nas mesmas unidades. Consideramos intercepto e inclinação como efeitos aleatórias. O modelo tem a seguinte forma:

$$
\begin{aligned}
Y_{it} &\sim B(\mu_{it}, \phi) \\
g(\mu_{it}) &= (\beta_0 + b_i) + (\beta_1 + b_1)t \\
\begin{bmatrix} b_0 \\ b_1 \end{bmatrix} &\sim NM_2\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_I^2 & \rho \\ \rho & \sigma_S^2 \end{bmatrix} \right)
\end{aligned}
$$

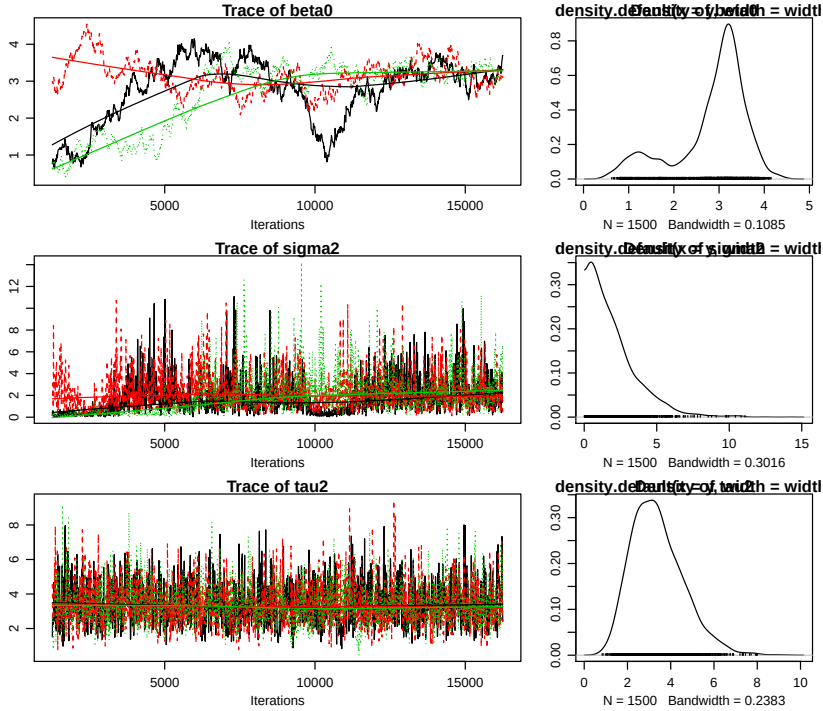

Figura 5.20: Traço das cadeias e distribuições a *posteriori* suavizadas - Poisson com efeito aninhado.

Neste modelo temos como vetor de parâmetros $\underline{\theta} = (\beta_0, \beta_1, \sigma_I^2, \sigma_S^2, \rho, \phi)$. A principal diferença deste modelo, para os dois anteriores é que neste caso a distribuição do efeito aleatório é uma gaussiana bivariada com parâmetro de correlação entre o intercepto e a inclinação aleatória, o que implica que não podemos utilizar o produto de gaussianas univariadas independentes. Este modelo é especificado para o `JAGS` no código 5.103.

Para o ajuste deste modelo precisamos passar uma lista com as matrizes de delineamentos `X.int`, `X.slope` da parte fixa, e `Z.int`, `Z.slope` da aleatória. É também necessário indicar do número de unidades amostrais `n.ua` e o número de repetições `n.rep`. Montamos uma lista nomeada com todas esta informações no código abaixo, vamos usar o mesmo conjunto de dados simulado na sessão 4.6 na qual o modelo foi ajustado por máxima verossimilhança.

```
dados <- read.table("dadosBeta.txt")
n.ua <- length(unique(dados$ID))
n.rep <- length(unique(dados$cov))
## Arrumando os dados para o JAGS
y.mat <- t(matrix(dados$y, n.rep, n.ua))
```

Código 5.103: Função em linguagem JAGS da densidade à *posteriori* para o modelo de regressão Beta longitudinal.

```
beta.longitudinal <- function(){
  for(j in 1:n.ua){
        for(i in 1:n.rep){
      Y[j,i] ~ dbeta(mu[j,i]*phi,(1-mu[j,i])*phi)
      logit(mu[j,i]) <- X.int[j,i]*beta0 + X.slope[j,i]*beta1 +
                        Z.int[j,i]*b0[j] + Z.slope[j,i]*b1[j]
    }
    b0[j] <- inter.slope[j,1]
    b1[j] <- inter.slope[j,2]
    inter.slope[j,1:2] ~ dmnorm(mu.theta[j,1:2], Pr.theta[1:2,1:2])
    mu.theta[j,1] <- 0
    mu.theta[j,2] <- 0
  }
  Sigma[1,1] <- tau.I^2
  Sigma[2,2] <- tau.S^2
  Sigma[1,2] <- rho*(tau.I*tau.S)
  Sigma[2,1] <- rho*(tau.I*tau.S)
  Pr.theta <- inverse(Sigma)
  beta0 ~ dnorm(0, 0.01)
  beta1 ~ dnorm(0, 0.01)
  phi   ~ dgamma(1, 0.01)
  tau.I ~ dgamma(1,0.01)
  tau.S ~ dgamma(1,0.01)
  rho   ~ dunif(-1,1)
}
```

```
## Separando os dados por id
dados.id <- split(dados,dados$ID)
## Matriz da parte fixa
X.int <- model.matrix(~1, data=dados.id[[1]])
X.slope <- model.matrix(~cov-1, data=dados.id[[1]])
X.int <- matrix(rep(X.int,n.ua),n.rep,n.ua)
X.slope <- matrix(rep(X.slope,n.ua),n.rep,n.ua)
## Matriz da parte aleatoria
Z.int <- X.int
Z.slope <- X.slope
## Montando a lista para o JAGS
dat.jags <- list(Y = y.mat, X.int = t(X.int), X.slope = t(X.slope),
  Z.int = t(Z.int), Z.slope = t(Z.slope), n.ua = n.ua, n.rep = n.rep)
```

Para o ajuste, usamos a função `jags.fit()` e obter um resumo do ajuste, via a função `summary()`. As distribuições a *posteriori* suavizadas podem ser vistas na Figura 5.21

```
beta.model <- jags.fit(dat.jags,
               c("beta0","beta1", "tau.I", "tau.S", "phi", "rho"),
               beta.longitudinal, n.adapt = 2000, n.update = 1000,
               thin = 5, n.iter= 10000, n.chains=3)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 1755

Initializing model
```

```
summary(beta.model)
```

```
Iterations = 3005:13000
Thinning interval = 5
Number of chains = 3
Sample size per chain = 2000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

         Mean     SD Naive SE Time-series SE
beta0  0.7236 0.3091 0.003991       0.024002
beta1  0.9772 0.2399 0.003098       0.014644
phi   34.9419 4.2756 0.055198       0.072013
rho    0.3792 0.3646 0.004706       0.013934
tau.I  0.7930 0.2647 0.003417       0.010123
tau.S  0.6405 0.2595 0.003350       0.009827

2. Quantiles for each variable:

          2.5%     25%     50%     75%   97.5%
beta0 -0.04907  0.5798  0.7347  0.9039  1.3135
beta1  0.48472  0.8347  0.9745  1.1291  1.4552
phi   27.24007 31.9274 34.8135 37.7524 43.7490
rho   -0.39075  0.1300  0.4193  0.6633  0.9581
tau.I  0.44447  0.6121  0.7385  0.9073  1.4676
tau.S  0.27393  0.4629  0.5941  0.7618  1.2917
```

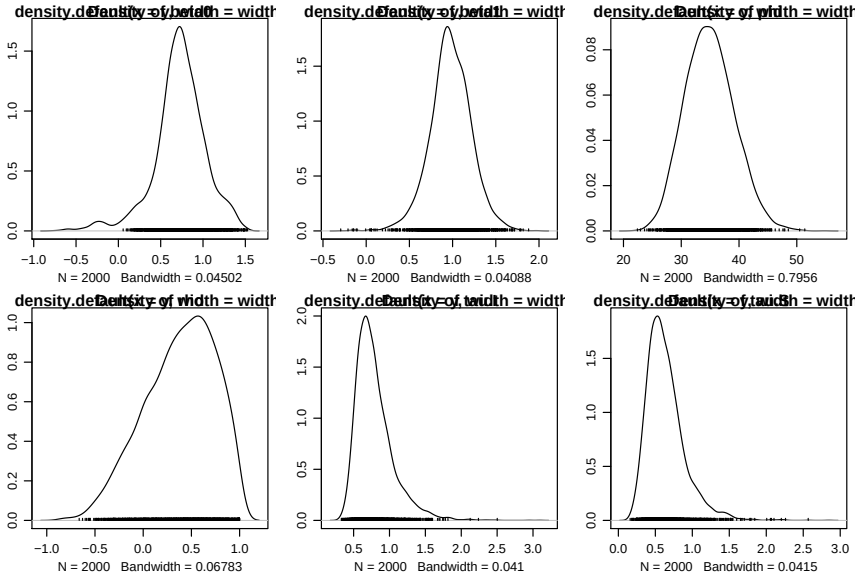

Figura 5.21: Distribuições a *posteriori* suavizadas - Beta longitudinal.

5.10 Regressão dinâmica via MCMC

Nesta seção nós voltamos aos modelos dinâmicos introduzidos no final da primeira parte do livro, na seção 4.8. A abordagem apresentada nessa seção apresenta uma limitação na estimação de θ_t. Essa limitação é devida ao fato de que θ_t é estimado considerando sua distribuição condicionada nos valores estimados de ψ.

A inferência bayesiana para os modelos dinâmicos é geralmente feita usando MCMC. Neste caso, um algoritmo amostrador de Gibbs pode ser usado para amostrar de $[\theta,\psi|\underline{y}]$, a partir das condicionais completas $[\psi|\theta,\underline{y}]$ e $[\theta|\psi,\underline{y}]$. $[\psi|\theta,\underline{y}]$ é específico de cada modelo. Para amostrar de $[\theta|\psi,\underline{y}]$, foi proposto o algoritmo `FFBS`, ***F**orward **F**iltering **B**ackward **S**ampling*, Carter & Kohn (1994), Früwirth-Schnatter (1994) e Shephard (1994). Este algoritmo é uma pequena modificação do passo de suavização de Kalman. Em lugar de estimar a média e variância de θ_t e retornar essas estimativas, é feita a simulação de θ_t a partir dessas estimativas de médias e variâncias.

Na suavização do algoritmo de Kalman, é considerado $[\theta_t|\underline{y}]$, porém, para simular θ_t, considera-se $[\theta_t|\theta_{t+1},\underline{y}]$, Petris et al. (2009). Assim, o algoritmo FFBS segue os passos:

1. execute o passo de filtragem
2. simule θ_n de $\theta_n|\underline{y}$

3. para $i = n-1, n-2, ..., 0$ simule θ_i de $\theta_i|\theta_{i+1},\underline{y}$.

Assim, podemos combinar a função que faz a filtragem com uma função para fazer o passo BS (*Backward Sampling*). Inicialmente vamos implementar uma função para simular da distribuição Normal multivariada que será utilizada para amostrar da distribuição do vetor de estados. Esta função irá simular apenas uma amostra. A função para o passo de *backward sampling* é implementada no código 5.104.

```
rmvnorm1 <- function(m, S)
  m + crossprod(chol(S), rnorm(ncol(as.matrix(S))))
```

Código 5.104: Algoritmo Backward Sampling para o modelo de regressão dinâmica.

```
kbs1r <- function(dlm) {
  ## dlm: lista com componentes do modelo dinâmico de regressão
  ##      suavizada com a funcao kfilter1r().
  m <- dlm$x.f[,dlm$n+1]
  S <- dlm$P.f[,,dlm$n+1]
  x.bs <- dlm$x.f
  x.bs[,dlm$n+1] <- rmvnorm1(drop(m), S)
  for (i in dlm$n:1) {
    aux <- chol2inv(chol(dlm$P.p[,,i+1]))
    aux <- crossprod(dlm$G,aux)
    J <- dlm$P.f[,,i]%*%aux
    m <- dlm$x.f[,i] + J%*%(x.bs[,i+1] - dlm$x.p[,i+1])
    S <- dlm$P.f[,,i]  - J%*%dlm$G%*%dlm$P.f[,,i]
    x.bs[,i] <- rmvnorm1(drop(m), S)
  }
  ## retorna uma matrix n x k dos estados simulados
  return(x.bs)
}
```

Como em geral ψ é desconhecido, considera-se o seguinte amostrador de Gibbs, em blocos, para simular de $\psi,\theta|\underline{y}$:

1. considere valores iniciais de ψ, ψ^0
2. para $i = 1, 2, ..., N$
   (a) simule $\theta^{(i)}$ de $[\theta|\psi^{(i-1)},\underline{y}]$ usando o algoritmo FFBS
   (b) simule $\psi^{(i)}$ de $[\psi|\psi^{(i-1)},\theta^{(i)},\underline{y}]$

e, para simular ψ, considera-se um algoritmo de Gibbs, se a distribuição condicional completa de ψ é conhecida ou, se não, o algoritmo de Metropolis ou Metropolis-Hastings. No exemplo de modelo de regressão dinâmica considerado, as distribuições condicionais completas possuem forma conhecida. Portanto, vamos simular destas distribuições condicionai.

Nós vamos considerar as funções `ddlm1r()` e `kfilter1r()`, definidas na seção 4.8.

Nós vamos considerar dois blocos: $\psi_1 = diag\{G\}$ e $\psi_2 = \{diag\{W\},V\}$. Vamos considerar prioris Normais(0,1), independentes, para cada elemento de ψ_1,

$$\psi_1[i] \sim N(m.g_i, s2.g_i)$$

e as distribuições condicionais completas, Petris et al. (2009), são

$$\psi_1[i] \sim N(mm.g_i, ss2.g_i)$$

em que

$$ss2.g_i = (\frac{1}{s2.g_i} + \psi_2[i] \sum_{t=1}^{n} \theta_t^2[i])^{-1}$$

$$mm.g_i = ss2.g_i(\frac{m.g_i}{ss2.g_i} + \psi_2[i] \sum_{t=1}^{n} \theta_t[i]\theta_{t-1}[i]).$$

A distribuição a priori de cada elemento de ψ_2, é uma Gama-Inversa, ou Gamma cada elemento de $1/\psi_2$. Assim, sendo os primeiros k elementos de ψ_2, a diagonal de W, $\psi_2[1], \psi_2[2], ..., \psi_2[k]$ com priori Gamma(a_i,b_i), temos que a posteriori

$$[1/\psi_2[i]|\theta,\underline{y}] = Gamma(a_i + \frac{n}{2}, b_i + \frac{1}{2}\sum_{t=1}^{n}(\theta_t[i] - G_{(i,i)}\theta_{t-1}[i])).$$

Para o elemento $k+1$ de ψ_2, $\psi_2[k+1]$, temos a posteriori

$$[1/\psi_2[k+1]|\theta,\underline{y}] = Gamma(a_{k+1} + \frac{n}{2}, b_i + \frac{1}{2}\sum_{t=1}^{n}(y_t - F_t\theta_t)).$$

Assim, o algoritmo MCMC para o modelo de regressão dinâmica, pode ser implementado com a função a seguir.

Vamos aplicar esse algoritmo na mesma serie de dados simuladas na seção 4.8. Vamos considerar dois valores iniciais para ψ e simular duas cadeias, para avaliar (visualmente) a convergência. Em ambos os casos, vamos considerar prioris $N(0.5, 0.1)$ para os elementos de $diag\{G\}$ e $G(0.5, 0.3)$ para os elementos de $diag\{W\}$ e para V.

```
### criando o modelo
m1r <- ddlm1r(y, x, diag(3), 1, diag(3), V0=diag(3)*10)
### criando dois conjuntos de valores iniciais
inis <- list(c(rep(1,3), rep(100,4)),
             c(rep(-1,3), rep(1/100,4)))
### usando mclapply() do pacote multicore
require(multicore)
## simula duas cadeias em paralelo
```

Código 5.105: Algoritmo FFBS com amostrador de Gibbs para o modelo de regressão dinâmica.

```
gffbs1r <- function(dlm, ini=NULL, a.g=rep(0,dlm$k),
                    b.g=rep(1,dlm$k), a.wv=rep(0.001, dlm$k+1),
                    b.wv=rep(0.001, dlm$k+1), m.g=0,
                    s2.g=1, nsim=10000, nburn=1000) {
  ## dlm: lista com componentes do modelo  de regressão dinamica.
  ## ini: valores iniciais de G, W e V. G e W diagonais e V escalar
  ##      Default=NULL, sendo tomados os valores G, W e V em 'dlm'
  ## a.g e b.g: parametros da distribuicao a priori beta para
  ##            os elementos da diagonal de G.
  ## a.wv e b.wv: parametros da distribuicao a priori Gamma para
  ##              o inverso da diagonal de W e inverso de V
  ## m.g e s2.g: parametros da distribuicao Normal, a
  ##             distribuicao a priori da diagonal de G
  ## nsim: número de amostras a serem simuladas
  ##       da distribuicão à posteriori. Valor default é 10000
  ## nburn: núm. amostras iniciais descartadas. Default nburn=1000.
  if (nburn>0) {
    res <- gffbs1r(dlm, ini=ini, a.g=a.g, b.g=b.g,
                   a.wv=a.wv, b.wv=b.wv, m.g=m.g, s2.g=s2.g,
                   nsim=nburn, nburn=0)
    return(gffbs1r(dlm, ini=as.double(res[nburn, 1:(dlm$k*2+1)]),
                   a.g=a.g, b.g=b.g, a.wv=a.wv, b.wv=b.wv,
                   m.g=m.g, s2.g=s2.g, nsim=nsim, nburn=0))
  }
  if (!is.null(ini)) {
    dlm$G <- diag(ini[1:dlm$k])
    dlm$W <- diag(ini[1:dlm$k + dlm$k])
    dlm$V <- ini[2*dlm$k+1]
  }
  res <- matrix(NA, nsim, dlm$k*2+1 + dlm$k*(dlm$n+1))
  aa.wv <- a.wv + dlm$n/2
  for (i in 1:nsim) {
    ## update theta
    dlm <- kfilter1r(dlm)
    theta.at <- kbs1r(dlm)
    res[i, 2*k+1+1:((dlm$n+1)*k)] <- as.vector(t(theta.at))
    ## update W and V
    ss.tt <- rowSums((theta.at[,-1] -
                      dlm$G%*%theta.at[,1:dlm$n])^2)
    ss.yft <- sum((dlm$y-colSums(dlm$F*theta.at[,-1]))^2)
    bb.wv <- b.wv + c(ss.tt, ss.yft)/2
    res[i, 1+k+0:k] <- 1/rgamma(k+1, aa.wv, bb.wv)
    dlm$W <- diag(res[i, k+1:k])
    dlm$V <- res[i, 2*k+1]
    ## update G
    aux <- rowSums(theta.at[,1:dlm$n]^2)/res[i, k+1:k]
    ss2.g <- 1/(1/s2.g + aux)
    aux <- rowSums(theta.at[,-1]*theta.at[,1:dlm$n])
    mm.g <- ss2.g*(aux/res[i,k+1:k] + m.g/s2.g)
    res[i, 1:k] <- rnorm(k, mm.g, sqrt(ss2.g))
    dlm$G <- diag(res[i, 1:k])
  }
  return(structure(res, class='mcmc', mcpar=c(1,nsim,1)))
}
```

```
##set.seed(123)
##system.time(res2 <- mclapply(inis, function(ii) {
  ##gffbs1r(m1r, ini=ii, a.g=0.5, b.g=0.3,
    ##      a.wv=.5, b.wv=.1, nsim=13000, nburn=0)}))
## guarda resultado em disco
##dump("res2", "res2")
## carrega resultado do disco
source("res2")
```

Na Figura 5.22 vemos as primeiras 50 amostras da distribuição a posteriori de ψ e dos primeiros valores em θ_1 para as duas cadeias. Observa-se a convergência das duas cadeias para um único patamar.

```
n0 <- 50
par(mfrow=c(5,2), mar=c(2,2,2,.5), mgp=c(2,1,0))
for (i in 1:10) {
  ylm <- range(sapply(res2, function(x)
                      as.vector(x[1:n0, i])))
  plot.ts(res2[[1]][1:n0,i], ylim=ylm,
          xlab="", ylab="")
  lines(res2[[2]][1:n0,i], lty=2)
}
```

Na Figura 5.23 vemos o traço de amostras da distribuição a posteriori dos parâmetros $diag\{G\}$, $diag\{W\}$, V e os primeiros três valores de θ_1. Estas amostras foram tomadas das duas cadeias simuladas de 13000 amostras, desconsiderando-se as primeiras 3000 amostras e tomando amostras a cada 10 amostras, totalizando 2000 amostras.

```
ind <- seq(3*nrow(res2[[1]])/13+10, nrow(res2[[1]]), 10)
amsel <- Reduce("rbind", lapply(res2, function(x) x[ind,]))
par(mfrow=c(5,2), mar=c(2,2,2,.5), mgp=c(2,1,0))
for (i in 1:10) {
  vp <- c(diag(G), diag(W), V, theta[1,1:3])[i]
  ylm <- range(amsel[,i], vp)
  plot.ts(amsel[,i], ylim=ylm, ylab="", xlab="")
  abline(h=vp, col="gray", lwd=3)
}
```

Na Figura 6.7, na Seção 6.4 comparamos resultados de MCMC e de uma alternativa utilizando o algoritmo INLA que será apresentado mais adiante. Na Figura são mostradas a densidade a posteriori de $1/V$, $diag\{G\}$, $1/diag\{W\}$, considerando as 2000 amostras simuladas por MCMC e a densidade a posteriori desses parâmetros calculadas via INLA.

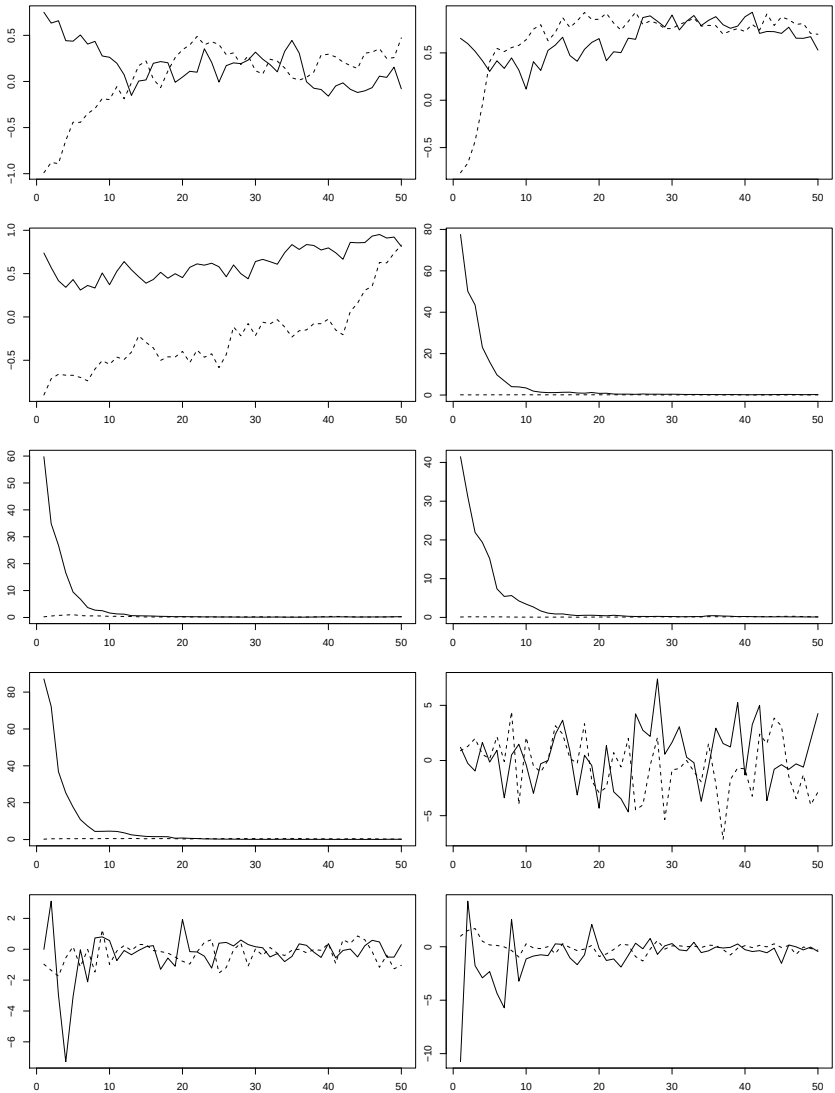

Figura 5.22: Primeiras 50 amostras da distribuição a posteriori dos parâmetros do modelo de regressão dinâmica, considerando dois pontos iniciais diferentes.

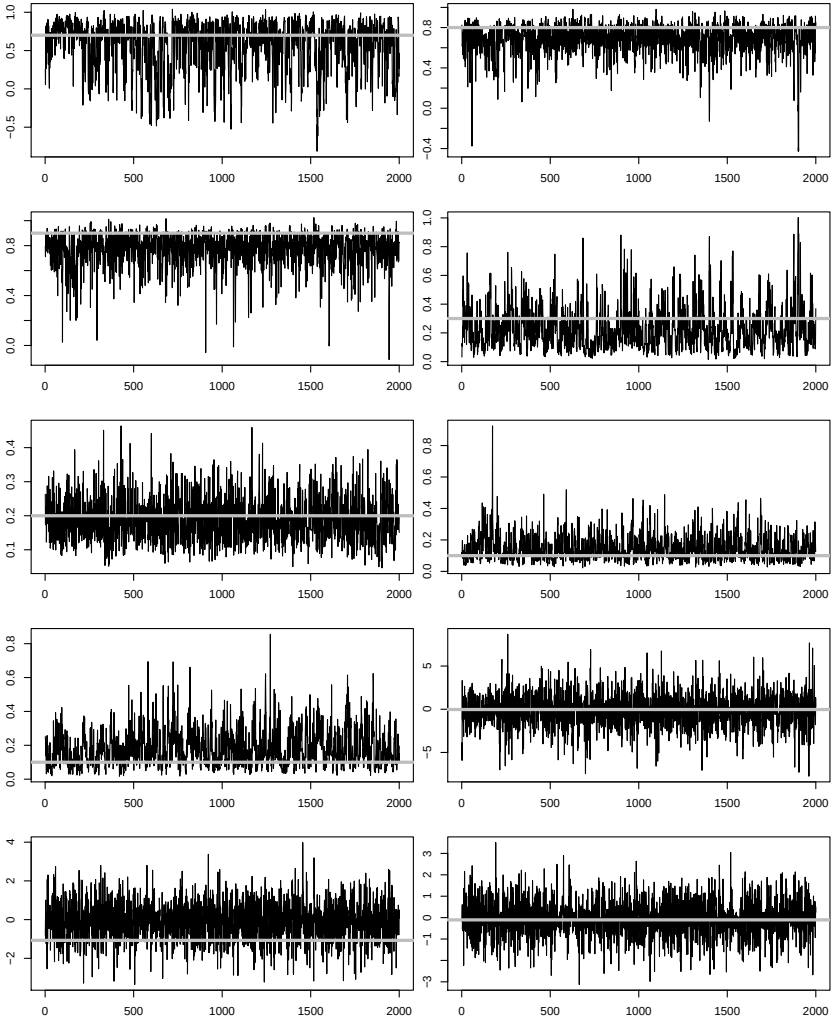

Figura 5.23: Traço de amostras da distribuição a posteriori de alguns parâmetros do modelo de regressão dinâmica.

Capítulo 6

Tópicos Adicionais

Neste Capítulo serão introduzidos duas estratégias alternativas às já apresentadas para inferência por verossimilhança e bayesiana. O algoritmo *data clone*, que usa os algoritmos *Markov Chain Monte Carlo - MCMC*, para fazer inferência baseada em verossimilhança para modelos complexos. No Capítulo 4 vimos a limitação das técnicas de integração numérica em trabalhar com integrais de grande dimensão em modelos de efeitos aleatórios. O algoritmo *data clone* é uma tentativa de superar essas limitações mais ainda sob paradigma de inferência baseada na verossimilhança. Um fato importante desta abordagem é que dela deriva uma forma bastante simples de estudar a estimabilidade de modelos complexos o que é uma discussão relativamente nova a um antigo problema. O algoritmo utiliza inferência bayesiana como um mecanismo de computação da verossimilhança e não como um paradigma de inferência. A intuição é a seguinte: em muitos modelos a medida que o número de dados cresce, a verossimilhança domina o problema e a *priori* é cada vez menos importante. A proposta é então aumentar a quantidade de dados simplesmente replicando os observados de forma que as estimativas são as mesmas que seria obtidas para o conjunto não replicado. Desta forma a clonagem de dados busca obter estimação e inferência baseada na verossimilhança.

Como vimos também nas Seções referentes a inferência bayesiana os algoritmos MCMC podem ser lentos e de difícil diagnóstico de convergência, entre outros problemas. Neste ponto, o método INLA - *Integrated Nested Laplace Approximation*, oferece uma alternativa para contornar estas limitações do MCMC retomando as ideias de integração numérica, porém com uma nova roupagem e no contexto de inferência bayesiana, onde apenas aproximações determinísticas são usadas para obter as distribuições a *posteriori*. A grande vantagem do INLA é o expressivo ganho em tempo computacional

que permite o ajuste e comparação de vários modelos para um problema.

6.1 Clonagem de dados

O algoritmo de clonagem de dados (*data cloning*) é implementando replicando os dados de forma adequada e fazendo inferência bayesiana, tipicamente via MCMC, nos dados replicados. Considere que as observações y_{ij} com $i = 1, \ldots, N$ blocos e $j = 1, \ldots, n_i$ repetições em cada bloco, são clonadas $K - vezes$ por blocos, ou seja, N blocos passam a ser $N \times K$ blocos. Denote os dados clonados por y_{ij}^K e a verossimilhança clonada resultante por $L^K(\boldsymbol{\beta}, \Sigma, \phi)$. A verossimilhança dos dados clonados possui duas características relevantes para o procedimento. Primeiro, a localização do ponto de máximo desta função é exatamente o mesmo que o da função $L(\boldsymbol{\beta}, \Sigma, \phi)$. Segundo, a matriz de informação de Fisher da verossimilhança clonada é K vezes a da verossimilhança original.

Usando a estrutura hierárquica do modelo misto podemos formular um modelo completo bayesiano designando *priori*'s para todos os elementos do vetor de parâmetros. Denote $\pi(\boldsymbol{\beta})$, $\pi(\Sigma)$ e $\pi(\phi)$ as distribuições a *priori* de cada componente do vetor de parâmetros do modelo. Combinando com a verossimilhança clonada temos a seguinte distribuição a *posteriori* de interesse,

$$\pi^K(\boldsymbol{\beta}, \Sigma, \phi | y_{ij}) = \frac{[\int f_i(\mathbf{y}_i|\boldsymbol{\beta}, \Sigma, \phi) f(\mathbf{b}_i|\Sigma) d\mathbf{b}_i]^K \pi(\boldsymbol{\beta})\pi(\Sigma)\pi(\phi)}{C(K; y_{ij})} \quad (6.1)$$

em que

$$C(K; y_{ij}) = \int [\int f_i(\mathbf{y}_i|\boldsymbol{\beta}, \Sigma, \phi) f(\mathbf{b}_i|\Sigma) d\mathbf{b}_i]^K \pi(\boldsymbol{\beta})\pi(\Sigma)\pi(\phi) d\boldsymbol{\beta} d\Sigma d\phi \quad (6.2)$$

é a constante normalizadora. Amostrar desta distribuição a *posteriori* é possível usando algoritmos de MCMC. Lele (2010a) mostram que quando K aumenta, a média desta distribuição a *posteriori* converge para o estimador de máxima verossimilhança e K vezes a variância a *posteriori* converge para a correspondente variância assintótica do MLE. Importante destacar que apesar da atribuição de distribuições a *priori* no algoritmo MCMC, a inferência resultante torna-se independente das *priori*'s escolhidas uma vez a verossimilhança passa a dominar a *posteriori* com o aumento do número de clones dos dados e a influência da *priori* torna-se desprezível na prática.

Ao propor um modelo hierárquico deve-se ter o cuidado para formular modelos que tenham compatibilidade com a estrutura dos dados e que possam ter todos os seus parâmetros identificáveis. Em muitas aplicações

práticas os modelos são complexos, tornando provas analíticas de identificabilidade extremamente difíceis e raramente são ou podem ser feitas na prática. De forma geral, a análise é feita assumindo-se que os parâmetros são de fato identificáveis (Lele, 2010b) em particular em análises bayesianas pois a não identificabilidade na verossimilhança pode ser resolvida (ou mascarada) pela *priori*.

A clonagem de dados é uma proposta de simples implementação para o estudo da identificabilidade de modelos. Lele (2010a) demonstra que se existem parâmetros são não identificáveis, ao aumentar-se o número de clones, a distribuição a *posteriori* converge para a distribuição a *priori* truncada no espaço de não identificabilidade. Consequentemente, o maior autovalor da matriz de variância-covariância não converge para zero. Este resultado pode ser usado para estudar a falta de identificabilidade de parâmetros em modelos hierárquicos de forma geral. Mais especificamente, se a variância da distribuição a *posteriori* de um parâmetro de interesse converge para zero com o aumento do número de clones, este parâmetro é identificável. Desta forma, *data cloning* alerta para o problema de identificabilidade e também ajuda o analista a decidir se um determinado parâmetro é ou não importante para o modelo.

Para exemplificar o uso e resultados fornecidos pelo método, vamos considerar dois modelos um sem e outro com problemas de estimabilidade. Nestas análises utilizaremos o pacote **dclone** que deve ser instalado (com suas dependências) e carregado.

```
require(dclone)
```

6.1.1 Modelo Poisson com intercepto aleatório

O modelo Poisson com intercepto aleatório definido em 4.8, já foi estimado anteriormente via verossimilhança (4.4) e inferência bayesiana usando MCMC (5.8.1) Agora vamos rapidamente mostrar como usar o algoritmo data clone para fazer inferência baseado em verossimilhança e estudar a estimabilidade dos parâmetros deste modelo.

O pacote **dclone** utiliza o **JAGS** para MCMC do algoritmo. A chamada é muito parecida com a da função `jags.fit()`. Lembre-se que o modelo escrito em `JAGS` tem a seguinte forma:

Código 6.106: Modelo Poisson com intercepto aleatório em JAGS.

```
mod.poisson <- function(){
  for(j in 1:n.ua){
        for(i in 1:n.rep){
                Y[j,i] ~ dpois(lambda[j,i])
        log(lambda[j,i]) <- beta0 + b[j]
    }
    b[j] ~ dnorm(0, tau2)
  }
  beta0 ~ dnorm(0, 0.001)
  tau2  ~ dgamma(0.5, 0.5)
}
```

Vamos utilizar aqui os mesmos dados simulados anteriormente supondo 10 unidades com 10 observações em cada uma delas. Preparando a base de dados, note que para usar o **dclone**, precisamos passar os índices i e j do modelo invertidos e transpor a matriz com a variável resposta.

```
y.mat <- matrix(dados$y, 10, 10)
dados.list <- list(Y = t(y.mat), n.ua = 10, n.rep = 10)
```

A chamada da função `dc.fit()`, tem todos os argumentos da função `jags.fit()` acrescida de alguns argumentos adicionais. O argumento `multiply` indica sobre quais unidades será feita a clonagem, neste caso de indivíduos. O argumento `unchanged` indica o que não será clonado, neste caso as repetições. O argumento `n.clones` indica quantos clones serão usados pelo algoritmo, neste caso vamos clonar $1, 5, 10, 20, 30, 40$ e até 50 vezes os dados. Podemos visualizar os resultados do ajuste com a função `summary()`.

```
k <- c(1,5,10,20,30,40,50)
clone <- dc.fit(data = dados.list, params= c("beta0","tau2"),
          model = mod.poisson, n.clones=k, n.iter= 10000, n.adapt = 500,
          n.update = 500, thin =5, multiply="n.ua",unchanged = "n.rep")
summary(clone)
```

A análise de estimabilidade é feita inspecionando os gráficos das amostras a *posteriori* de acordo com o número de clones e das variâncias em escala logarítmica. Sob identificabilidade, as variâncias devem diminuir com o aumento da clonagem segundo a taxa esperada. Este comportamento é verificado nos gráficos Figura 6.1, ilustrando o diagnóstico de identificabilidade para o modelo Poisson com intercepto aleatório que possui o comportamento esperado.

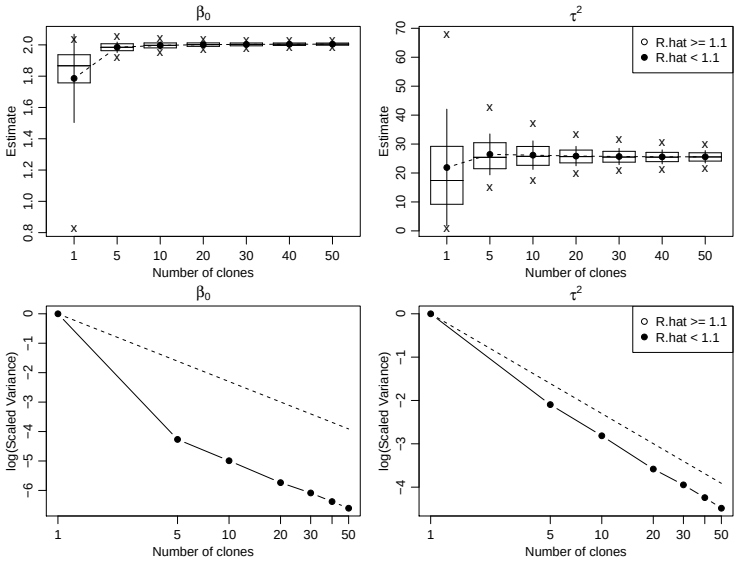

Figura 6.1: Boxplot's das amostras e logaritmo da variância a *posteriori* para diferentes números de clones no modelo Poisson com intercepto aleatório.

6.1.2 Modelo gaussiano com efeito aleatório

Considere agora o modelo com duas fontes de variação (componentes de variância) especificado como a seguir.

$$
\begin{aligned}
Y_i &\sim N(\mu_i, 1/\sigma^2) \\
\mu_i &= \beta_0 + b_i \\
b_i &\sim N(0, 1/\tau^2).
\end{aligned}
$$

Este modelo é claramente não identificável pois tem duas fontes de variação que não podem ser diferenciadas. Seria necessário que obtivéssemos repetições de Y para o mesmo μ_i para tornar o modelo identificável. Não havendo replicações um dos parâmetros de variância é redundante, porém a soma deles é estimável. Este é um modelo muito simples, a ideia é usá-lo para mostrar qual o comportamento deste modelo quando fazemos o diagnóstico de estimabilidade. O que espera-se acontecer é que as cadeias para β_0 e $\phi = 1/\sigma^2 + 1/\tau^2$ tenham um bom comportamento, porém as cadeias de σ^2 e τ^2 tenham um comportamento anômalo refletindo o problema de identificabilidade. Vamos primeiro definir uma função para simular observações sem replicação.

Código 6.107: Simulação do modelo gaussiano com efeito aleatório.

```
simula.normal <- function(b0, sigma, tau, n){
  bi <-  rnorm(n, 0, sd= 1/tau)
  mui <- b0 + bi
  return(rnorm(n, mean=mui, sd=1/sigma))
}
```

Simulando deste modelo com $\beta_0 = 5$, $\sigma = 1$, $\tau = 1$ e portanto $\phi = 2$ e $n = 100$ obtemos um conjunto de dados que armazenamos em uma lista adequada para uso do JAGS.

```
set.seed(123)
Y <- simula.normal(b0 =5, sigma = 1, tau = 1, n =100)
dat.gauss <- list(Y = Y, n = 100)
```

O passo seguinte é escrever o modelo em notação JAGS como fizemos em exemplos do Capítulo anterior. No exemplo definimos *priori*s $N(0,1000)$ para o parâmetro de média e $Gamma(1/2,1/2)$ para ambos parâmetros de variância.

Código 6.108: Modelo Gaussiano com intercepto aleatório em JAGS.

```
model.normal <- function(){
  for(i in 1:n){
    Y[i] ~ dnorm(mu[i], sigma2)
    mu[i] <- b0 + b[i]
    b[i] ~ dnorm(0, tau2)
  }
  b0 ~ dnorm(0, 0.001)
  sigma2 ~ dgamma(0.5, 0.5)
  tau2 ~ dgamma(0.5, 0.5)
  phi <- (1/sigma2) + (1/tau2)
}
```

Ajustamos o modelo via **dclone** com até 50 clones. Em uma chamada extraímos os parâmetros originais e em outra apenas a soma dos parâmetros de variância. Podemos ver o comportamento das cadeias e das distribuições a *posteriori* na Figura 6.2.

```
k <- c(1,5,10,20,30,40,50)
Gclone1 <- dc.fit(data = dat.gauss, params= c("b0","tau2","sigma2"),
         model = model.normal, n.clones=k, n.iter= 5000,
         n.adapt = 1000, n.update = 100, thin = 5, multiply="n")
Gclone2 <- dc.fit(data = dat.gauss, params= c("phi"),
         model = model.normal, n.clones=k, n.iter= 5000, n.adapt = 1000,
         n.update = 100, thin = 5, multiply="n")
```

Os gráficos da Figura 6.2 mostram claramente que as cadeias para β_0 e $\phi = 1/\tau^2 + 1/\sigma^2$ têm o comportamento esperado. Já as cadeias de σ^2 e

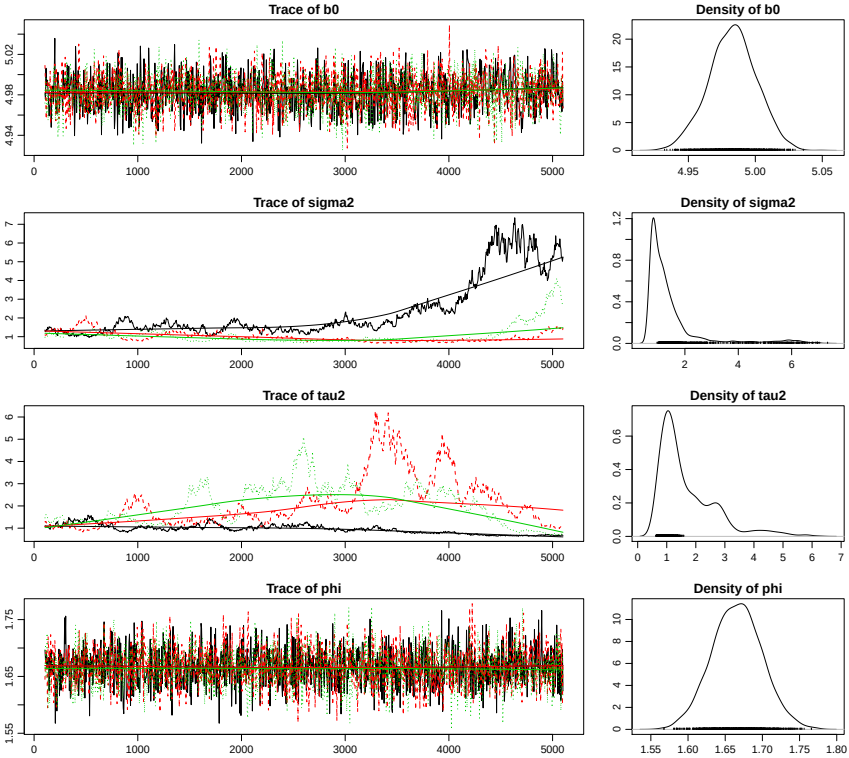

Figura 6.2: Traço das cadeias e distribuições a *posteriori*.

τ^2 mostram claras evidências de não convergência. O diagnóstico de estimabilidade, apresenta *boxplot*'s das amostras da *posteriori* de acordo com o número de clones utilizadas para cada um dos parâmetros e um gráfico do logaritmo da variância escalonada que, sob identificabilidade deve decrescer seguindo a linha pontilhada.

Pode-se ver claramente pelos gráficos que os parâmetros σ^2 e τ^2 são não identificáveis e prejudicam muito a convergência das cadeias. Olhando para a soma, temos uma situação bem comportada já que este é um modelo muito simples, estimar o parâmetro de média e variância da distribuição Normal é um trabalho bastante simples computacionalmente.

Dados com replicações

Uma situação diferente da anterior situação é quando temos mais que uma amostra dentro da mesma unidade de observação. Pode-se pensar em simples replicações ou mesmo em um modelo de medidas repetidas ou mesmo dados longitudinais. Na situação de distribuição gaussiana, esta-

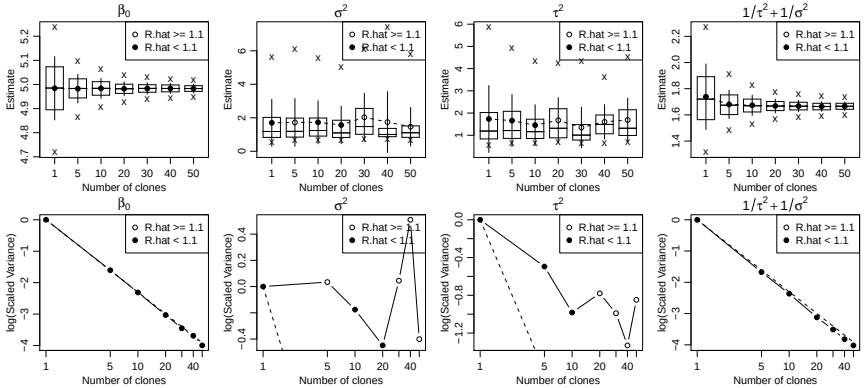

Figura 6.3: Boxplot's amostras e logaritmo da variância da *posteriori*.

mos na classe de modelos lineares mistos. O modelo é o mesmo do anterior, somente os dados são indexados para refletir as replicações.

$$
\begin{aligned}
Y_{ij}|b_i &\sim N(\mu_i, 1/\sigma^2) \\
\mu_i &= \beta_0 + b_i \\
b_i &\sim N(0, 1/\tau^2).
\end{aligned}
$$

Denotamos com $i = 1, \ldots, N$ as unidades amostrais e repetições $j = 1, \ldots, n_i$ em cada uma das unidade. Note que não é necessário que todas as unidades amostrais tenham a mesma quantidade de repetições. Vamos generalizar a função anterior para simular deste modelo. A função é escrita em função de parâmetros de precisão (inverso da variância).

Código 6.109: Simulando modelo gaussiano com intercepto aleatório com replicações.

```
simula.gauss <- function(X, Z, beta.fixo, prec.pars){
  n.bloco <- ncol(Z)
  n.rep <- nrow(Z)/n.bloco
  bi <- rnorm(n.bloco, 0, sd=1/prec.pars[1])
  mu <- cbind(X,Z)%*%c(beta.fixo,bi)
  y <- rnorm(length(mu),mu, sd = 1/prec.pars[2])
  return(data.frame(y=y,X=X,ID=rep(1:n.bloco,each=n.rep)))
}
```

Usando a função, podemos simular um conjunto de dados com $N = 8$ unidades amostrais e dentro de cada unidade $n_i = 10$ repetições. Como parâmetros tem-se $\beta_0 = 5$, $\tau = 1$ e $\sigma = 1$. Ao final, os dados são colocados na estrutura requerida para utilizar o JAGS.

```
ID <- as.factor(rep(1:8,each=10))
X <- rep(1,80)
Z <- model.matrix(~-1 + ID)
set.seed(123)
dados <- simula.gauss(X=X,Z=Z,beta.fixo = 5, prec.pars=c(1,1))
y.mat <- matrix(dados$y,10,8)
dados.list <- list(Y = t(y.mat), n.ua = 8, n.rep = 10)
```

Podemos escrever este modelo em **JAGS** adaptando o código 6.108 para permitir as replicações (blocos de observações por unidade).

Código 6.110: Modelo Gaussiano com intercepto aleatório por blocos em JAGS.

```
mod.gauss.rep <- function(){
  for(j in 1:n.ua){
        for(i in 1:n.rep){
                Y[j,i] ~ dnorm(mu[j,i], sigma2)
                mu[j,i] <- beta0 + b[j]
     }
     b[j] ~ dnorm(0,tau2)
  }
  beta0  ~ dnorm(0, 0.001)
  tau2   ~ dgamma(0.5, 0.5)
  sigma2 ~ dgamma(0.5, 0.5)
}
```

Chamando novamente a função `dc.fit()` temos o ajuste a seguir. Diferentemente do caso anterior, as cadeias (Figura 6.4) com as densidade a *posteriori* agora tem um comportamento adequado. Os gráficos na Figura 6.5 confirmam a estimabilidade deste modelo para a estrutura de dados disponível. O resumo apresenta os resultados para 50 clonagens.

```
k <- c(1,5,10,20,30,40,50)
GcloneR <- dc.fit(data=dados.list, params= c("beta0","tau2", "sigma2"),
       model = mod.gauss.rep, n.clones=k, n.iter= 5000, n.adapt = 1000,
       n.update = 100, thin = 5, multiply="n.ua",unchanged = "n.rep")
```

```
summary(GcloneR)
```

```
Iterations = 105:5100
Thinning interval = 5
Number of chains = 3
Sample size per chain = 1000
Number of clones = 50

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

        Mean      SD  DC SD  Naive SE Time-series SE  R hat
beta0  5.222 0.29226 2.0666 0.0053358      0.0223480 1.0007
sigma2 1.261 0.02958 0.2092 0.0005401      0.0004612 1.0009
tau2   1.098 0.14190 1.0034 0.0025906      0.0092132 0.9997
```

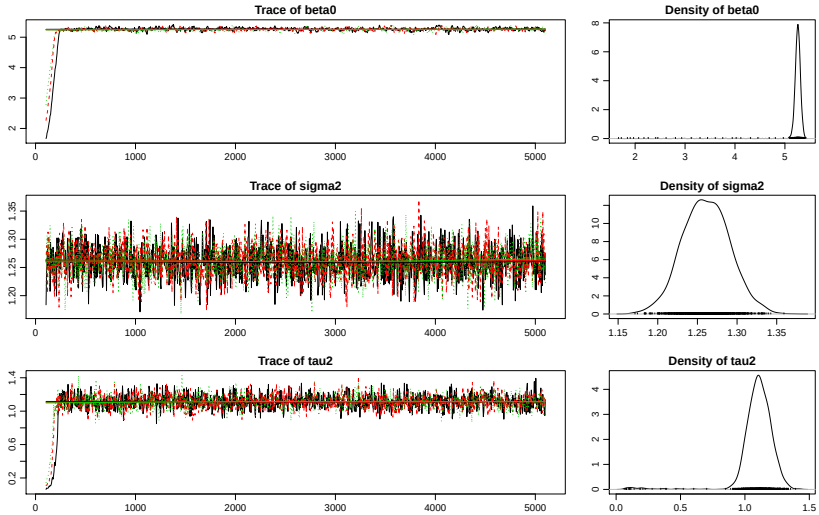

Figura 6.4: Traço das cadeias e densidade a *posteriori* - Modelo Gaussiano com blocos.

```
2. Quantiles for each variable:

         2.5%   25%   50%   75% 97.5%
beta0  5.1115 5.223 5.258 5.291 5.356
sigma2 1.2029 1.241 1.260 1.281 1.320
tau2   0.9118 1.051 1.109 1.169 1.280
```

6.2 INLA - Aproximação de Laplace aninhada integrada

O texto desta seção está fortemente baseado no artigo de Rue et al. (2009a). Seguimos muito perto o original, enfatizando alguns pontos considerados importantes para o entendimento da nova metodologia.

Para simplificar a discussão seguinte, denote genericamente $\pi(.|.)$ como a densidade condicional de seus argumentos, e faça $\underline{x}$ ser todas as n variáveis gaussianas η_i, α, $f^{(j)}$, β_k. A densidade $\pi(\underline{x}|\theta_1)$ é gaussiana com média zero (assumida), matriz de precisão $Q(\theta_1)$ com hiperparâmetro θ_1. Denote por $N(\underline{x};\mu,\Sigma)$ a $N(\mu,\Sigma)$ densidade gaussiana com média μ e covariância (inverso da precisão) Σ na configuração $\underline{x}$. Note que foi incluído η_i ao invés de ϵ_i em $\underline{x}$, isto simplificará a notação no seguir do texto.

A distribuição para as n_d variáveis observadas $\underline{y} = y_i : i \in I$ é denotada por $\pi(\underline{y}|\underline{x},\theta_2)$ e assume-se que $y_i : i \in I$ são condicionalmente inde-

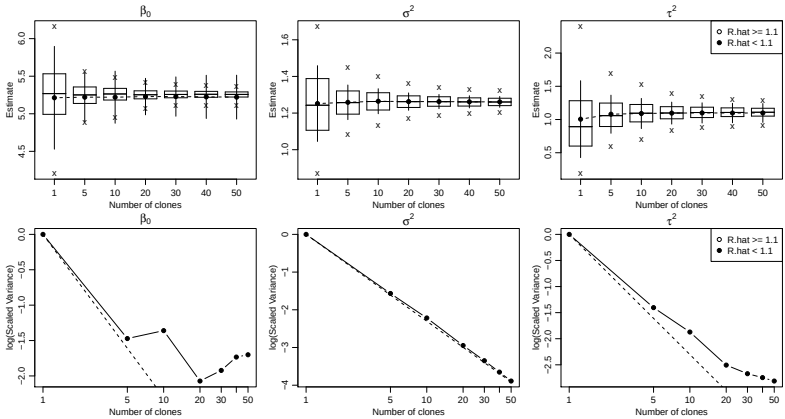

Figura 6.5: Boxplot's das amostras e logaritmo da variância da *posteriori*.

pendentes dado $\underline{x}$ e θ_2. Este ponto é muito importante, pois ao assumir independência condicional a $\pi(\underline{y}|\underline{x},\theta_2)$ se resume a um produto da distribuição assumida para a variável resposta, ou seja, a verossimillhança. Para simplificar, denote por $\underline{\theta} = (\theta_1^T, \theta_2^T)^T$ com $dim(\underline{\theta}) = m$. A *posteriori* para uma matriz $Q(\underline{\theta})$ não singular, fica dada por

$$\begin{aligned}\pi(\underline{x},\underline{\theta}|\underline{y}) &\propto \pi(\underline{\theta})\pi(\underline{x}|\underline{\theta})\prod_{i\in I}\pi(y_i|x_i,\underline{\theta}) \\ &\propto \pi(\underline{\theta})\|Q(\underline{\theta})\|^{n/2}\exp\left(-\frac{1}{2}\underline{x}^T Q(\underline{\theta})\underline{x} + \sum_{i\in I}\log\pi(y_i|x_i,\underline{\theta})\right).\end{aligned}$$

A imposição de restrições lineares (se necessário) são denotadas por $A\underline{x} = e$ para uma matriz A $k \times n$ de rank k. O objetivo principal é aproximar as marginais a *posteriori* $\pi(x_i|\underline{y})$, $\pi(\underline{\theta}|\underline{y})$ e $\pi(\theta_j|\underline{y})$. Muitos, mas não todos os modelos Gaussianos Latentes na literatura satisfazem duas propriedades básicas que serão assumidas aqui. A primeira é que o campo latente $\underline{x}$, que usualmente é de grande dimensão, admite propriedades de independência condicional. Assim, o campo latente é um Campo Aleatório Markoviano Gaussiano (CAMG) com uma matriz de precisão $Q(\underline{\theta})$ (Rue & Held, 2005). Isto significa que pode-se usar métodos numéricos para matrizes esparsas, que em geral são muito mais rápidos que os métodos gerais para matrizes densas (Rue & Held, 2005). A segunda propriedade é que o número de hiperparâmetros m, é pequeno, $m \leq 6$. Ambas propriedades são usualmente necessárias para produzir inferência rápida, embora existam excessões (Eidsvik et al., 2009).

A abordagem INLA trabalha usando o fato que a marginal a *posteriori*

de interesse pode ser escrita como

$$\pi(x_i|\underline{y}) = \int \pi(x_i|\underline{\theta},\underline{y})\pi(\underline{\theta}|\underline{y})\partial\underline{\theta} \quad \text{e} \quad \pi(\theta_j|\underline{y}) = \int \pi(\underline{\theta}|\underline{y})\partial\underline{\theta}_{-j}. \tag{6.3}$$

O fato chave da abordagem é usar esta forma para construir aproximações aninhadas,

$$\widetilde{\pi}(x_i|\underline{y}) = \int \widetilde{\pi}(x_i|\underline{\theta},\underline{y})\widetilde{\pi}(\underline{\theta}|\underline{y})\partial\underline{\theta} \quad \text{e} \quad \widetilde{\pi}(\theta_j|\underline{y}) = \int \widetilde{\pi}(\underline{\theta}|\underline{y})\partial\underline{\theta}_{-j}. \tag{6.4}$$

Aqui, $\widetilde{\pi}(.|.)$ é a densidade (condicional) aproximada de seus argumentos. Aproximações para $\pi(x_i|\underline{y})$ são calculadas aproximando $\pi(\underline{\theta}|\underline{y})$ e $\pi(x_i|\underline{\theta},\underline{y})$, e usando integração numérica (soma finita) para integrar fora $\underline{\theta}$. A integração é possível quando a dimensão de $\underline{\theta}$ é pequena, em geral menor ou igual a 6. Como vai ficar claro no decorrer do texto. A abordagem aninhada torna a aproximação de Laplace muito acurada quando aplicada para modelos Gaussianos latentes. A aproximação de $\pi(\theta_j|\underline{y})$ é calculada integrando fora θ_{-j} vindo de $\widetilde{\pi}(\underline{\theta}|\underline{y})$; este ponto será retomado na sequência para detalhes práticos.

A abordagem INLA é baseada na seguinte aproximação $\widetilde{\pi}(\underline{\theta}|y)$ para a marginal *posteriori* de $\underline{\theta}$.

$$\widetilde{\pi}(\underline{\theta}|y) \propto \left.\frac{\pi(\underline{x},\underline{\theta},\underline{y})}{\widetilde{\pi}_G(\underline{x}|\underline{\theta},\underline{y})}\right|_{\underline{x}=\underline{x}^*(\underline{\theta})} \tag{6.5}$$

onde, $\widetilde{\pi}_G(\underline{x}|\underline{\theta},\underline{y})$ é a aproximação gaussiana para a condicional completa de $\underline{x}$ o que caracteriza a aproximação como de Laplace, e $\underline{x}^*(\underline{\theta})$ é a moda para a condicional completa de $\underline{x}$, para um dado $\underline{\theta}$. Note que a expressão 6.5 só é válida em um ponto, e portanto, para obter a aproximação da distribuição completa esta expressão precisa ser avaliada para um dado conjunto de $\underline{\theta}$. A proporcionalidade em 6.5 segue do fato que a constante normalizadora para $\pi(\underline{x},\underline{\theta}|\underline{y})$ é desconhecida. Note que $\widetilde{\pi}(\underline{\theta}|\underline{y})$ tende a ser bastante diferente da gaussiana. Isto sugere que a aproximação gaussiana direta para $\pi(\underline{\theta}|\underline{y})$ não é acurada o bastante. Um aspecto crítico da abordagem INLA é explorar e manipular $\widetilde{\pi}(\underline{\theta}|\underline{y})$ e $\widetilde{\pi}(x_i|\underline{y})$ de uma forma não paramétrica. (Rue & Martino, 2007) usaram 6.5 para aproximar marginais *posteriori* para vários modelos Gaussianos latentes. Suas conclusões foram que $\widetilde{\pi}(\underline{\theta}|\underline{y})$ é particularmente acurada, mesmo rodando um longo MCMC não puderam detectar nenhum erro nesta aproximação. Para as marginais *posteriori* do campo latente, eles propõem começar pela $\widetilde{\pi}_G(\underline{x}|\underline{\theta},\underline{y})$, ou seja,

$$\widetilde{\pi}(x_i|\underline{\theta},\underline{y}) = N(x_i;\mu_i(\underline{\theta}),\sigma_i^2(\underline{\theta})). \tag{6.6}$$

Aqui, $\mu(\underline{\theta})$ é a média (vetor) para a aproximação gaussiana, considerando que $\sigma^2(\underline{\theta})$ é o vetor correspondente de variâncias marginais. Esta aproximação pode ser integrada numericamente com respeito a $\underline{\theta}$, ver 6.4,para obter aproximações para as marginais de interesse do campo latente,

$$\widetilde{\pi}(x_i|\underline{y}) = \sum_k \widetilde{\pi}(x_i|\underline{\theta_k}, \underline{y}) \times \widetilde{\pi}(\underline{\theta_k}|\underline{y}) \times \Delta_k. \tag{6.7}$$

A soma é sobre os valores de $\underline{\theta}$ com pesos Δ_k. Rue & Martino (2007) mostraram que a marginal *posteriori* para $\underline{\theta}$ foi acurada, enquanto o erro na aproximação gaussiana 6.6 foi grande. Em particular, 6.6 pode apresentar erro na locação e/ou falta de assimetria (*skewness*). A maior dificuldade relatada em Rue & Martino (2007) foi detectar os $x_i's$ nas quais a aproximação era menos acurada e a falta de habilidade para melhorar a aproximação nestas localizações. Além disso, eles não conseguiam controlar o erro da aproximação e escolher os pontos de integração θ_k de uma forma adaptativa e automática. Rue et al. (2009a) contornam as dificuldades encontradas em Rue & Martino (2007) e apresentam uma abordagem completa para inferência aproximada em modelos gaussianos latentes que nomearam de *Integrated Nested Laplace Approximations* (INLA). A principal mudança é aplicar mais uma vez a aproximação de Laplace, desta vez para $\pi(x_i|\underline{y},\underline{\theta})$. Eles também apresentam uma alternativa rápida que corrige a aproximação gaussiana 6.6 para o erro de locação e falta de assimetria a um custo extra moderado. Estas correções são obtidas por uma expansão em séries de Laplace. Esta alternativa é uma escolha inicial natural por ser de baixo custo computacional e alta acurácia. Os autores demonstram como derivar ferramentas para testar a aproximação, para aproximar marginais a *posteriori* para um subconjunto de $\underline{x}$, e para calcular várias medidas de interesse, tais como, verossimilhança marginal, Critério de Informação da *Deviance* (DIC) e outras várias medidas preditivas bayesianas.

Antes de apresentar a proposta completa de Rue et al. (2009a) é interessante apresentar de forma rápida a aproximação gaussiana. A abordagem INLA é baseada em aproximações gaussianas para densidades da forma:

$$\pi(\underline{x}) \propto \exp\left(-\frac{1}{2}\underline{x}^T Q \underline{x} + \sum_{i\in I} g_i(x_i)\right) \tag{6.8}$$

onde $g_i(x_i)$ é $\log \pi(y_i|x_i, \underline{\theta})$. A aproximação gaussiana $\widetilde{\pi}_g(\underline{x})$ é obtida encontrando a moda e a curvatura na moda. A moda é calculada iterativamente usando o método de Newton Raphson, também conhecido como o algoritmo escore e sua variante o escore de Fisher Fahrmeir & Tutz (2001). Seja $\underline{\mu}^{(0)}$ um valor inicial, expande-se $g_i(x_i)$ em torno de $\mu_i^{(0)}$ até o termo de segunda ordem

$$g_i(x_i) \approx g_i(\mu_i^{(0)}) + b_i x_i - \frac{1}{2} c_i x_i^2 \tag{6.9}$$

onde b_i e c_i depende de $\mu^{(0)}$. A aproximação gaussiana é obtida, com matriz de precisão $Q + diag(c)$ e moda dada pela solução de $(Q + diag(c))\underline{\mu}^{(1)} = \underline{b}$. Este processo é repetido até a convergência para a distribuição gaussiana com, média $\underline{x}^*$ e matriz de precisão $Q^* = Q + diag(c^*)$. Como o termo não quadrático em 6.9 é somente função de x_i e não uma função de x_i e x_j, então a matriz de precisão para a aproximação gaussiana é da forma $Q + diag(c)$. Isto é computacionalmente conveniente, porque as propriedades Markovianas do CAMG são preservadas. Existem outras sugestões na literatura de como construir aproximações gaussianas, ver Rue (2001), Rue & Held (2005) e Kuss & Rasmussen (2005).

Nesta seção será apresentada a abordagem INLA para aproximar marginais a *posteriori* para campos latentes Gaussianos, $\pi(x_i|\underline{y})$, $i = 1, \ldots, n$. A aproximação é feita em três passos. O primeiro passo aproxima a marginal a *posteriori* de $\underline{\theta}$ usando a aproximação de Laplace 6.5. O segundo passo calcula a aproximação de Laplace, ou a aproximação de Laplace simplificada, para $\pi(x_i|\underline{y}, \underline{\theta})$, para valores selecionados de $\underline{\theta}$, a fim de melhorar a aproximação gaussiana 6.6. O terceiro passo combina os dois anteriores usando integração numérica 6.7.

Cabe ressaltar aqui que toda a seção foi escrita usando o artigo de Rue et al. (2009a), sendo uma revisão deste onde alguns pontos considerados importantes, são explicitados para facilitar a leitura e entendimento. Ainda, antes de descrever propriamente a abordagem INLA, será apresentado a metodologia como um todo através de um simples algoritmo, passo-a-passo. Considere,

1. Selecione um conjunto de $\Theta = (\underline{\theta}_1, \ldots, \underline{\theta}_k)$
2. Para $k = 1$ até K faça
3. Calcule $\widetilde{\pi}(\underline{\theta}_k|\underline{y})$
4. Calcule $\widetilde{\pi}(x_i|\underline{\theta}_k, \underline{y})$ como uma função de x_i
5. Fim para
6. Calcule $\widetilde{\pi}(x_i|\underline{y}) = \sum_k \widetilde{\pi}(x_i|\underline{\theta}_k, \underline{y})\widetilde{\pi}(\underline{\theta}_k|\underline{y})\Delta_k$

O algoritmo começa selecionando um conjunto possívelmente pequeno de $\underline{\theta}'s$. O procedimento para selecionar este conjunto é descrito na seção 3.5.1, e é feito explorando a distribuição $\widetilde{\pi}(\underline{\theta}|\underline{y})$.

Após ter os vetores duas aproximações são calculadas, a primeira é exatamente a distribuição $\widetilde{\pi}(\underline{\theta}|\underline{y})$ que é calculada como na equação 3.17. Para a segunda aproximação existem três possibilidades.

A primeira e mais barata computacionalmente é a gaussiana que é explicada na seção 3.4 e retomada na seção 3.5.3, a segunda e mais acurada é

a aproximação de Laplace que é explicada na seção 3.5.3. Como ficará claro no decorrer do texto o último passo que consiste na integração numérica é bastante facilitado pelo procedimento explicado na seção 3.5.1 induzir que todos os pesos Δ_k sejam iguais, o que torna o último passo simplesmente uma soma com todos os pesos iguais. Esta é uma descrição bastante geral da abordagem INLA, e os detalhes seguem nas próximas seções.

Para o cálculo da aproximação $\widetilde{\pi}(x_i|\underline{\theta}_k, \underline{y})$ (Rue et al., 2009a) propõem também uma terceira possibilidade denominada de *aproximação de Laplace simplicada*, que através de uma aproximação oferece substanciais ganhos computacionais. Esta abordagem não será considerada neste texto.

6.2.1 Explorando $\widetilde{\pi}(\theta|y)$

O primeiro passo da abordagem INLA é calcular uma aproximação para a *posteriori* marginal de $\underline{\theta}$ como em 6.5. O denominador em 6.5 é a aproximação gaussiana para a condicional completa de $\underline{x}$, e é calculado segundo a equação 6.8. O principal uso de $\widetilde{\pi}(\underline{\theta}|\underline{y})$ é incorporar a incerteza com respeito a $\underline{\theta}$ quando aproximando a marginal a *posteriori* de x_i, ver 6.7. Para fazer isto, não é necessário representar $\widetilde{\pi}(\underline{\theta}|\underline{y})$ parametricamente, mas basta explorá-la suficientemente bem para encontrar bons pontos para o cálculo da integração numérica. Até o fim desta seção, será discutido como as marginais a *posteriori* $\pi(\theta_j|\underline{y})$ podem ser aproximadas. Assuma por simplicidade que $\underline{\theta} = (\theta_1, \ldots, \theta_m) \in \Re^m$, que pode sempre ser obtido usando uma reparametrização.

- **Passo 1** Localize a moda de $\widetilde{\pi}(\underline{\theta}|\underline{y})$, otimizando $\log \widetilde{\pi}(\underline{\theta}|\underline{y})$ com respeito a $\underline{\theta}$. Isto pode ser feito usando algum método quasi-Newton que vai usar alguma aproximação da derivada de segunda ordem de $\log \widetilde{\pi}(\underline{\theta}|\underline{y})$ usando sucessivas diferenças entre vetores gradientes. O gradiente é aproximado usando diferenças finitas. Denote por $\underline{\theta}^*$ como sendo a configuração modal.

- **Passo 2** Na configuração modal $\underline{\theta}^*$ calcule a matriz Hessiana $H > 0$, usando diferença finita. Seja $\Sigma = H^{-1}$, que vai ser a matriz de covariância para $\underline{\theta}$ se a densidade fosse gaussiana. Para facilitar a exploração, use variáveis padronizadas z ao invés de $\underline{\theta}$: Seja $\Sigma = V\Lambda V^T$ a decomposição em autovalores e autovetores de Σ, e defina θ via z, como segue

$$\theta(z) = \underline{\theta}^* + V\Lambda^{1/2}z. \tag{6.10}$$

  Se $\widetilde{\pi}(\underline{\theta}|\underline{y})$ é a densidade gaussiana, então z é $N(0,I)$. Esta reparametrização corrige a escala e rotação, e simplifica a integração numérica, ver Smith et al. (1987).

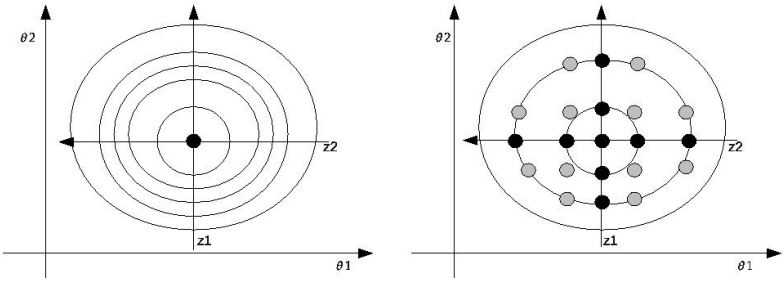

Figura 6.6: Ilustração da exploração da marginal posteriori para θ.

- **Passo 3** Explore $\log \widetilde{\pi}(\underline{\theta}|\underline{y})$ usando a z-parametrização. A figura 6.6 ilustra o procedimento quando $\log \widetilde{\pi}(\underline{\theta}|\underline{y})$ é unimodal. O painel (a) mostra um gráfico de contorno para $\log \widetilde{\pi}(\underline{\theta}|\underline{y})$ para $m = 2$, a localização da moda e o novo eixo de coordenada para z. Deve-se explorar $\log \widetilde{\pi}(\underline{\theta}|\underline{y})$ com o objetivo de encontrar pontos com a maior massa de probabilidade. O resultado deste processo é mostrado no painel (b). Cada sinal é um ponto onde $\log \widetilde{\pi}(\underline{\theta}|\underline{y})$ é considerado como significante, e que será usado na integração numérica 6.7. Detalhes são como segue. Começa-se vindo da moda $(z = 0)$, e caminha-se na direção positiva de z_1 com passos de tamanho δ_z por exemplo $\delta_z = 1$, contanto que

$$\log \widetilde{\pi}(\underline{\theta}(0)|\underline{y}) - \log \widetilde{\pi}(\underline{\theta}(z)|\underline{y}) < \delta_\pi \tag{6.11}$$

  onde, por exemplo $\delta_\pi = 2.5$. Então troca-se a direção e procede-se similarmente. As outras coordenadas são tratadas da mesma forma. Isto produz os sinais pretos. Pode-se agora preencher todos os valores intermediários tomando-se todas as diferentes combinações do sinais pretos. Estes novos sinais (mostrados em cinza) são incluídos se 6.11 for satisfeita. Como o *layout* dos pontos $\underline{\theta}_k$ é uma grade regular, pode-se tomar todos os pesos Δ_k em 6.7 como sendo iguais.

Em (a) a moda é localizada, a hessiana e o sistema de coordenadas z são calculados. Em (b) cada coordenada é explorada (pontos pretos) até um certo limite da log-densidade. Finalmente os pontos cinza são explorados.

6.2.2 Aproximando $\pi(\theta_j|y)$.

Marginais a posteriori para θ_j podem ser obtidas diretamente vindo de $\widetilde{\pi}(\underline{\theta}|\underline{y})$ usando integração numérica. Entretanto, isto é computacionalmente

caro, porque é necessário calcular $\widetilde{\pi}(\underline{\theta}|\underline{y})$ para um grande número de configurações. Uma abordagem mais factível é usar os pontos obtidos durantes os passos 1-3 para construir um interpolante para $\log \widetilde{\pi}(\underline{\theta}|\underline{y})$, e calcular marginais usando integração numérica vinda deste interpolante. Se uma alta acurácia é necessária, na prática basta obter uma configuração mais densa, por exemplo $\delta_z = 1/2$ ou $1/4$.

6.2.3 Aproximando $\pi(x_i|\theta, y)$

Até aqui tem-se um conjunto de pontos ponderados $\underline{\theta}_k$ para ser usado na integração 6.7. O próximo passo é providenciar aproximações acuradas para a posteriori marginal dos x_i's condicionado nos valores selecionados de $\underline{\theta}$. Nesta seção são discutidas, duas aproximações para $\widetilde{\pi}(x_i|\underline{y}, \theta_k)$, a gaussiana e a de Laplace.

Usando aproximação gaussiana

Uma aproximação simples e barata computacionalmente para $\pi(x_i|\underline{\theta},\underline{y})$ é a aproximação gaussiana $\widetilde{\pi}_G(x_i|\underline{\theta},\underline{y})$, onde a média $\mu_i(\underline{\theta})$ e a variância marginal $\sigma_i^2(\underline{\theta})$ são derivadas usando propriedades de Campos Aleatórios Markovianos Gaussianos, e algoritmos para matrizes esparsas Rue & Martino (2007). Durante a exploração de $\widetilde{\pi}(\underline{\theta}|\underline{y})$, ver seção 3.4, já foi calculada $\widetilde{\pi}(x|\underline{\theta},\underline{y})$, assim só variâncias marginais precisam ser calculadas. A aproximação gaussiana apresenta resultados razoáveis, mas pode apresentar erros na locação e/ou erros devido a falta de assimetria Rue & Martino (2007).

Usando aproximação de Laplace

A forma natural de melhorar a aproximação gaussiana é calcular a aproximação de Laplace

$$\widetilde{\pi}_{LA}(x_i|\underline{\theta},\underline{y}) \propto \left.\frac{\pi(x,\underline{\theta},\underline{y})}{\widetilde{\pi}_{GG}(\underline{x}_{-i}|x_i,\underline{\theta},\underline{y})}\right|_{\underline{x}_{-i}=\underline{x}^*_{-i}(x_i,\underline{\theta})} \tag{6.12}$$

Aqui, $\widetilde{\pi}_{GG}$ é a aproximação gaussiana para $x_{-i}|x_i,\theta,\underline{y}$, e x^*_{-i} é a configuração modal. Note que $\widetilde{\pi}_{GG}$ é diferente da densidade condicional correspondente a $\widetilde{\pi}_G(x|\underline{\theta},\underline{y})$. Infelizmente, 6.12 implica que $\widetilde{\pi}_{GG}$ precisa ser recalculada para cada valor de x_i e $\underline{\theta}$, uma vez que sua matriz de precisão depende de x_i e $\underline{\theta}$. Isto é muito caro, porque requer n fatorizações da matriz de precisão completa. (Rue et al., 2009a) propõem duas modificações em 6.12 que tornam isto factível computacionalmente. A primeira modificação consiste

em evitar o passo de otimização ao calcular $\widetilde{\pi}_{GG}(x_{-i}|x_i,\underline{\theta},\underline{y})$ aproximando a configuração modal,

$$\underline{x}^*_{-i}(x_i,\underline{\theta}) \approx E_{\widetilde{\pi}_G}(\underline{x}_{-i}|x_i). \tag{6.13}$$

O lado direito é calculado sob a densidade condicional derivada da aproximação gaussiana $\widetilde{\pi}_G(x|\underline{\theta},\underline{y})$. O benefício computacional é imediato. Primeiro, a média condicional pode ser calculada por uma atualização *rank-one* vindo da média incondicional. Outro fato positivo é que a média condicional é contínua com respeito a x_i, que não é o caso quando otimização numérica é usada para calcular $x^*_i(x_i,\underline{\theta})$. A segundo modificação materializa a seguinte intuição: somente aqueles x_j que estão 'próximos' de x_i vão ter impacto na marginal de x_i. Se a dependência entre x_j e x_i decai com o aumento da distância entre i e j, somente aqueles x_j's em uma 'região de interesse' acerca de i, $R_i(\theta)$, determinam a marginal de x_i. A esperança condicional em 6.13 implica que

$$\frac{E_{\widetilde{\pi}_G}(x_j|x_i)-\mu_j(\underline{\theta})}{\sigma_j(\underline{\theta})} = a_{ij}(\underline{\theta})\frac{x_i-\mu_i(\underline{\theta})}{\sigma_i(\underline{\theta})} \tag{6.14}$$

para alguma $a_{ij}(\underline{\theta})$ quando $i \neq j$. Assim, uma simples regra para construir o conjunto $R_i(\underline{\theta})$ é

$$R_i(\underline{\theta}) = j : |a_{ij}(\underline{\theta})| > 0.001. \tag{6.15}$$

A importância computacional de usar $R_i(\underline{\theta})$ segue do cálculo do denominador de 6.12, onde agora só é necessário fatorar uma $|R_i(\underline{\theta})| \times |R_i(\underline{\theta})|$ matriz esparsa. A expressão 6.12, simplificada como explicado acima, precisa ser calculada para diferentes valores de x_i para encontrar a densidade. Para selecionar estes pontos, usa-se a média e variância da aproximação gaussiana 6.6, e combina-se, diferentes valores para as variáveis padronizadas

$$x_i^{(s)} = \frac{x_i-\mu_i(\underline{\theta})}{\sigma_i(\underline{\theta})} \tag{6.16}$$

de acordo com as correspondentes escolhas de abscissas dadas pela regra de quadratura de Gauss-Hermite. Para representar a densidade $\widetilde{\pi}_{LA}(x_i|\underline{\theta},\underline{y})$, usa-se

$$\widetilde{\pi}_{LA}(x_i|\underline{\theta},\underline{y}) \propto N(x_i;\mu_i(\underline{\theta}),\sigma_i^2(\underline{\theta})) \times \exp\{\text{cubic spline}(x_i)\} \tag{6.17}$$

O spline cúbico é ajustado para a diferença entre a log-densidade $\widetilde{\pi}_{LA}(x_i|\underline{\theta},\underline{y})$ e $\widetilde{\pi}_G(x_i|\underline{\theta},\underline{y})$ nas abscisas selecionadas, e então a densidade é normalizada usando integração por quadratura.

A verossimilhança marginal $\pi(\underline{y})$ é uma quantidade útil para comparar modelos, como o fator de Bayes é definido pela razão de verossimilhanças marginais entre dois modelos competidores.

$$\widetilde{\pi}(\underline{y}) = \int \frac{\pi(\underline{\theta}, \underline{x}, \underline{y})}{\widetilde{\pi}_G(\underline{x}|\underline{\theta}, \underline{y})} \Bigg\|_{\underline{x}=\underline{x}^*(\underline{\theta})} d\underline{\theta}. \tag{6.18}$$

onde $\pi(\underline{\theta}, \underline{x}, \underline{y}) = \pi(\underline{\theta})\pi(\underline{x}|\underline{\theta})\pi(\underline{y}|\underline{x}, \underline{\theta})$. Uma alternativa, mais simples para estimar a verossimilhança marginal é obtida por assumir que $\underline{\theta}|\underline{y}$ é gaussiana; então 6.18 torna-se uma quantidade conhecida vezes $|H|^{-1/2}$, onde H é a matriz hessiana. Este método pode falhar no caso em que a marginal posteriori $\pi(\underline{\theta}|\underline{y})$ é multi-modal, mas isto não é específico do cálculo de verossimilhança marginal mas se aplica a abordagem geral. Felizmente, modelos gaussianos latentes geram distribuições *posteriori*'s unimodais em muitos casos.

O critério de informação da *Deviance* Spiegelhalter et al. (2001) é um critério de informação popular para modelos hierárquicos, e (em muitos casos) é bem definido para *priori*'s impróprias. Sua principal aplicação é a seleção de modelos bayesianos, mas ele também dá uma noção do número efetivo de parâmetros no modelo. Neste contexto, a *deviance* é

$$D(\underline{x}, \underline{\theta}) = -2 \sum_{i \in I} \log \pi(y_i|x_i, \underline{\theta}) + const \tag{6.19}$$

DIC é definido como duas vezes a média da *deviance* menos a *deviance* para a média. O número efetivo de parâmetros é dado a *deviance* da média da média da *deviance*.

6.3 Exemplos computacionais com INLA

Nesta seção vamos apresentar um resumo dos códigos INLA para o ajuste de alguns dos modelos apresentados do livro. Utilizaremos o pacote **INLA** destacando que este pacote não está disponível nos repositórios oficiais do R e deve ser obtido na página do projeto[1]. É importante que sejam verificadas e instaladas as dependências deste pacote. Vamos supor aqui que o leitor instalou e carregou o **INLA** para efetuar as análises.

```
require(INLA)
```

»= Começamos com o modelos de regressão Poisson. Pedimos ao leitor cuidado com as comparações que o INLA usa a distribuição Normal, parametrizada com precisão ao invés de desvio padrão.

[1] http://www.r-inla.org

```
require(INLA)
## Modelo Poisson
dados.poisson <- read.table("POISSON.txt",header=TRUE)
mod.poisson <- inla(y ~ cov, family="poisson", data=dados.poisson)
```

O INLA não tem uma verossimilhança Simplex, porém recentemente foi implementada a verossimilhança Beta, vamos exemplificar o seu uso com os dados do IQVC.

```
## Modelo Beta
dados.beta <- read.table("simplex.txt")
mod.beta <- inla(y ~ MEDIA, family="beta", data=dados.beta)
```

Na parte dos modelos com efeitos aleatórios o modelo mais simples implementado foi o Poisson com intercepto aleatório, em INLA o ajuste fica da seguinte forma.

```
## Poisson com intercepto aleatório
dados.int <- read.table("InterceptoPoisson.txt", header=TRUE)
mod.int <- inla(y~f(ID,model="iid"), family="poisson", data=dados.int)
```

Seguindo para o modelo Poisson com efeito aninhado.

```
## Poisson Aninhado
dados.ani <- read.table("PoissonAni.txt",header=TRUE)
mod.Poisson.ani <- inla(y ~ f(ID,model="iid") + f(Bloco.A, model="iid"),
                        family="poisson", data=dados.ani)
```

Finalizamos com o modelo Beta longitudinal.

```
## Beta longitudinal
dados.beta.long <- read.table("dadosBeta.txt",header=TRUE)
i <- 1:150
j <- 151:300
model.beta.long <- inla(y ~ cov + f(i, model="iid2d", n = 300) +
              f(j,cov,copy="i"), family="beta", data=dados.beta.long)
```

6.4 Regressão dinâmica via INLA

A inferência bayesiana ganhou um enorme salto de usabilidade a partir da facilidade de recursos computacionais disponibilizados para implementar MCMC. Porém, em modelos mais complexos, usar MCMC pode ser proibitivo do ponto de vista computacional. Isto porque crescendo a complexidade do modelo, geralmente cresce o número de parâmetros, crescendo também o número de distribuições a posteriori para buscar. Isso torna a convergência lenta e necessário muitas amostras da posteriori ou algoritmos específicos. Há casos em que demoram-se dias para obter uma amostra satisfatória de MCMC em modelos com alguma complexidade, e em especial os que estruturas de dependências cuja dimensionalidade das operações cresce com as observações tais como modelos espaciais e/ou temporais.

Diante dessas dificuldades, uma alternativa atrativa para uma grande classe de modelos é o uso de aproximações analíticas da distribuição a posteriori. O uso das aproximações de Laplace aninhadas e integradas, *Integrated Nested Laplace Aproximations* - INLA, proposto por Rue et al. (2009c), tem se provado útil, relativamente geral e operacional em uma diversidade de contextos. Essa difusão está se dando de forma rápida, motivada principalmente pela disponibilidade de software, na forma do pacote **INLA** (Rue et al., 2009b) do R, para aplicação dessa técnica a uma grande variedade de modelos. No restante desta sessão vamos considerar que o pacote **INLA** está instalado[2] e carregado.

```
require(INLA)
```

A inferência via INLA pode ser extendida para modelos dinâmicos como demostrado em Ruiz-Cárdenas et al. (2012) e nos exemplos que acompanham o artigo. Estimar modelos de regressão dinâmica usando o INLA em R pode ser feito de forma bastante direta a partir das funções disponíveis no pacote **INLA** com um arranjo adequado da estrutura de dados. Outros modelos dinâmicos, tais como modelos de crescimento, multivariados e espaço-temporais, também podem ser estimados via INLA (Ruiz-Cárdenas et al., 2012).

A modelagem usando o pacote **INLA** envolve a especificação de modelos na classe de modelos lineares generalizados e de sobrevida com efeitos mistos. Os modelos podem conter efeitos fixos e efeitos aleatórios gaussianos. Além disso, pode-se especificar efeitos aleatórios gaussianos para os coeficientes das covariáveis de tal forma que temos os coeficientes variando. É exatamente esta possibilidade que permite a estimação modelos de regressão dinâmica usando o **INLA** .

Dentre os modelos gaussianos para efeitos aleatórios estruturados no tempo, há o autoregressivo de ordem 1, o passeio aleatório de ordem 1 e o passeio aleatório de ordem 2. Assim, basta especificar um destes para os coeficientes de regressão de um modelo de regressão para séries temporais e teremos um modelo de regressão dinâmica.

Um efeito aleatório é declarado no **INLA** usando a função `f()` deste pacote. Nesta função é necessário informar um vetor de índices com valores variando na dimensão do efeito aleatório. Para os dados da seção anterior, vamos declarar um vetor de índices para o intercepto e para cada covariável. As covariáveis entram como pesos multiplicando as estimativas do efeito aleatório. Esse efeito aleatório pode ou não ter restrição de soma zero. No nosso caso não vamos ter essa restrição.

A seguir simulamos um conjunto de dados ($n = 100$) segundo um modelo dinâmico como definido em 4.8.

[2]disponível para *download* em www.r-inla.org

Para fazer comparação com os resultados através do algorítmo de MCMC, da Seção 5.10, vamos considerar as mesmas distribuições a priori para ψ. Para utilizar o INLA definimos a estrutura de dados em um `data-frame` que inclui variáveis `ID` identificadoras das unidades às quais serão atribuídos efeitos aleatórios. Defimos também uma lista com informações sobre os parâmetros e finalmente utilizamos a função `inla()` declarando o modelo desejado por uma `formula` em Rna qual os termos considerados aleatórios são definidos dentro de `f()` sempre utilizando a variável indicadora das unidades.

```
dad <- data.frame(y=y, i0=1:n, i1=1:n,
                  i2=1:n, x1=x[2,], x2=x[2,])
hyp <- list(theta1=list(param=c(.5,.1), initial=0),
            theta2=list(param=c(.5,.3)))
mod <- inla(y ~ 0 + f(i0, model="ar1", constr=FALSE, hyper=hyp) +
         f(i1, x1, model="ar1", constr=FALSE, hyper=hyp) +
         f(i2, x2, model="ar1", constr=FALSE, hyper=hyp), data=dad,
         control.data=list(hyper=list(theta=list(initial=0,
                                          param=c(.5,.1)))))
```

O objeto `mod` guarda um sumário e a distribuição marginal a posteriori de cada efeito aleatório, θ_t, e dos parâmetros ψ.

Na Figura 6.7 nós visualizamos a densidade a posteriori de $1/V$, $diag\{G\}$, $1/diag\{W\}$, considerando as 2000 amostras simuladas por MCMC (curvas em linhas contínuas) e as densidades marginais a posteriori obtidas via INLA (curvas tracejadas). A linha pontilhada na vertical corresponde ao valor verdadeiro usado para simular os dados. Também, visualizamos a região de 95% credibilidade para cada valor de θ_t considerando as 2000 amostras simuladas por MCMC (área em cinza) e o intervalo de 95% de credibilidade obtido via INLA (linhas tracejadas). A linha vertical é o valor verdadeiro de θ.

Observamos na Figura 6.7 que os resultados são satisfatórios embora apresentem diferenças para alguns parâmetros. Por exemplo, para $diag\{W\}[1]$ temos que a distribuição a posteriori obtida via INLA é mais concentrada que a obtida via MCMC. No caso de G, também são observadas algumas diferenças. Neste caso, nós consideramos um algoritmo MCMC com possibilidade de simular valores menores que -1 e maiores que 1. Já considerando a abordagem via INLA, o modelo considerado a priori para θ é autoregressivo de ordem 1, fazendo com que os elementos de $diag\{G\}$ sejam restritos ao intervalo $(0, 1)$. Portanto neste caso a diferença parece estar no modelo e não no algoritmo de inferência.

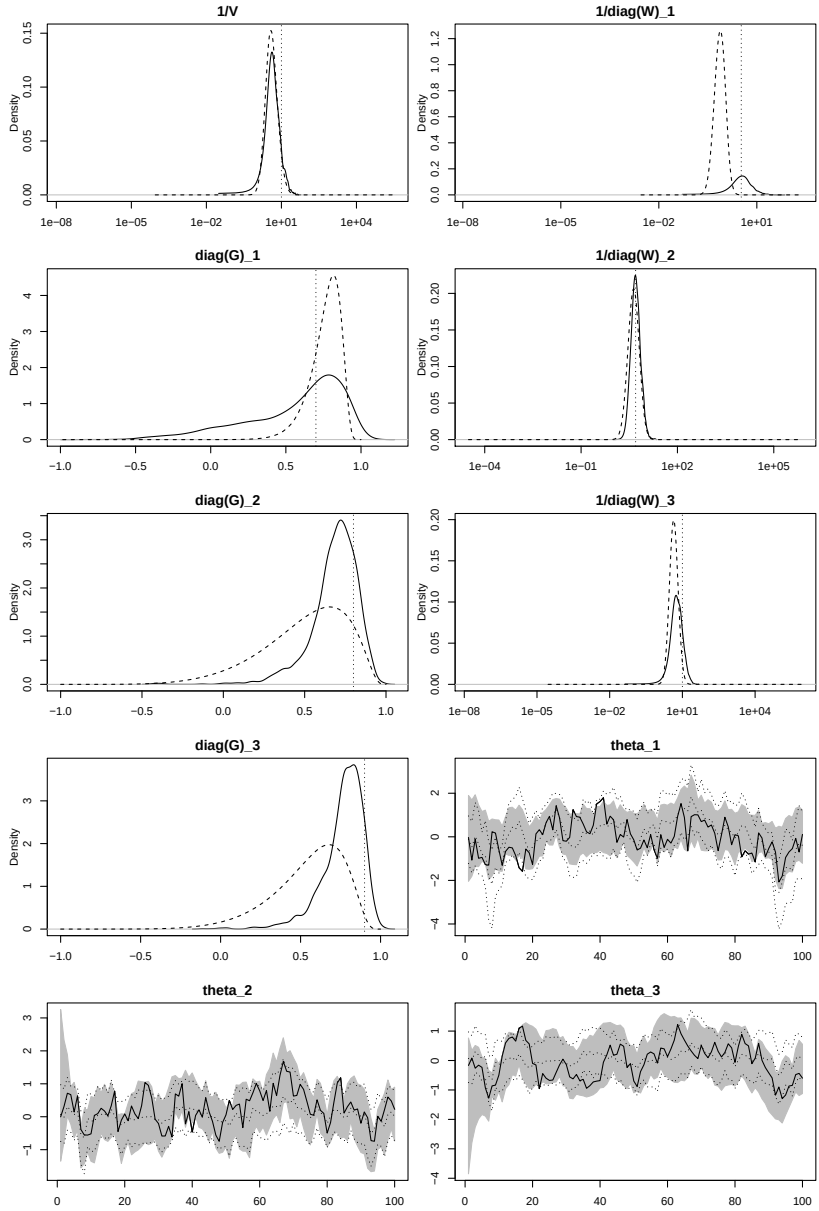

Figura 6.7: Gráficos comparativos dos resultados via MCMC e via INLA para o modelo de regressão dinâmica.

6.5 Modelo dinâmico espaço temporal via INLA

Nesta seção vamos considerar que temos dados observados em diferentes localizações geográficas e em diferentes momentos no tempo, ou seja, temos um dado espaço-temporal. Na dimensão temporal é mais comum se ter dados em variação temporal discreta, por exemplo, dados diários, mensais, anuais, etc. Na dimensão espacial podemos ter dados em alguns pontos de uma região geográfica ou dados de áreas geográficas (setores censitários, municípios, estados, etc). Nesta última situação, as regiões são fixas, assim como os tempos. Na situação de dados observados em alguns pontos, pode-se ter os pontos fixados, por exemplo observações climáticas feitas em estações meteorológicas, ou ter a localização dos pontos aleatória, por exemplo a localização da residência de pessoas acometidas de uma certa doença. Para cada uma dessas três situações, os modelos para os termos espaciais são diferentes. Assunção et al. (2010) apresentam e discutem os fundamentos e resultados centrais da especificação condicional de modelos espaciais.

6.5.1 Um modelo para mortalidade infantil

Nesta seção, vamos considerar um formato de dados bastante comum na área de mapeamento de doenças. Nessa área, é comum a análise de dados do número de casos de uma doença agrupados em região geográfica, como, por exemplo, um município. Desta forma, vamos considerar um modelo de variação espacial discreta. Além disso, vamos considerar, neste caso, uma distribuição de Poisson para a resposta.

Vamos analisar dados de mortalidade infantil nos 37 municípios da mesoregião de Curitiba. A partir do site do DATASUS, conseguimos dados de nascidos vivos de 1994 a 2009 e número de óbitos infantis de 1996 a 2009. Também consideramos três possíveis covariáveis: cobertura de imunização total, cobertura de imunização oral contra tuberculose (BCG) e cobertura de imunização contra poliomelite. Para estras três covariáveis, há dados de cada município para o período de 1995 a 2011. Assim, vamos considerar os dados no período de 1995 a 2009. Consideramos os óbitos em 1995 e 2009 como dados faltantes e podemos estimá-los utilizando o modelo. Para o ano de 2009 vamos comparar o números estimados com os observados.

Seja $o_{i,t}$ e $n_{i,t}$ os números de óbitos infantis e de nascidos vivos, respectivamente, no município i no ano t. Temos $i = 1,2,\ldots,N = 37$ e $t = 1,2,\ldots,T = 15$. Sob a suposição de que a taxa de mortalidade infantil é igual para todos o municípios em todo o período, essa taxa única é

estimada por

$$R = \frac{\sum_{t=1}^{T}\sum_{i=1}^{N} o_{i,t}}{\sum_{t=1}^{T}\sum_{i=1}^{N} n_{i,t}}.$$

Nessa hipótese, o número esperado de óbitos infantis é dado por $E_{i,t} = R * n_{i,t}$. Com isso vamos considerar o que

$$o_{i,t} \sim Poisson(\psi_{i,t} E_{i,t})$$

em que $\psi_{i,t}$ é o risco relativo do município i no ano t. Esse risco mede o quanto o $o_{i,t}$ está acima ou abaixo do número esperado $E_{i,t}$.

Também podemos considerar que a taxa em cada ano muda na mesma proporção para todos os municípios. Assim, podemos considerar

$$R_t = \frac{\sum_{i=1}^{N} o_{i,t}}{\sum_{i=1}^{N} n_{i,t}}.$$

Desta forma elimina-se qualquer tendência global existente ao longo do período.

Vamos considerar o seguinte modelo para $\psi_{i,t}$

$$\log(\psi_{i,t}) = \eta_{i,t} = \alpha_{i,t} + \beta_{i,t} X_{i,t}$$

em que $\eta_{i,t}$ é o preditor linear; $X_{i,t}$ é uma variável observada e varia em cada município e em cada ano; $\beta_{i,t}$ mede o efeito da covariável e varia para cada município e para cada ano, $\alpha_{i,t}$ é um intercepto aleatório variando para cada município e no tempo e captura a variação em $\psi_{i,t}$ não explicada pela covariável. Se $o_{i,t}$ é próximo de $E_{i,t}$, ou seja, o observado não difere muito do esperado, $\psi_{i,t} = 1$ e $\eta_i = 0$.

Considerando a classe de modelos proposta em Vivar & Ferreira (2009) e Vivar (2007), Temos que

$$\alpha_{i,t} = \phi_\alpha \alpha_{i,t-1} + w1_{i,t} \beta_{i,t} = \phi_\beta \beta_{i,t-1} + w2_{i,t}$$

em que $w1_{,t}$ e $w_{,t}$ são erros gaussianos multivariados com média zero e matriz de covariância com estrutura espacial. Para estimar esses modelos podemos considerar a abordagem proposta em Ruiz-Cárdenas et al. (2012).

Inicialmente vamos considerar um caso particular, quando $w1_{,t}$ e $w2_{,t}$ são ICAR, autoregressivo intrínsico, a priori (Besag & Kooperberg, 1995). Neste caso,

$$w1_{i,t} | w1_{-i,t} \sim N(\sum_{j \sim i} w1_{j,t}/d_i, \sigma^2_{w1}/d_i)$$

em que $w1_{-i,t}$ indica o vetor $w1_{,t}$ sem a área i, $j \sim i$ é o conjunto de áreas vizinhas da área i, d_i é o número de áreas vizinhas da área i e $\sigma^2_{w1} = \tau^{-1}_{w1}$ é

o parâmetro de variância de α. Portanto, a distribuição condicional de $w1_{i,t}$ é Normal com média sendo a média de $w1$ nas áreas vizinhas e a variância inversamente proporcional ao número de áreas vizinhas.

A estimação de alguns modelos dinâmicos usando o pacote **INLA**, pode ser simplificada para alguns casos com recentes opções de sintaxe no pacote para definir efeitos aleatórios por grupos. Podemos definir um modelo espacial para cada tempo e considerar cada tempo como um grupo e propor um modelo autoregressivo de ordem 1 para grupos. Cameletti et al. (2011) desenvolve uma aplicação desta abordagem em modelagem espaço-temporal. No nosso exemplo, suponha uma matriz Q_S de precisão para a componente espacial de $\alpha_{i,t}$, $w1_{,t}$. Considerando Q_T a matriz de estrutura da precisão de um processo univariado temporal autoregressivo de ordem 1, temos que a matriz de precisão de $w1$ é $Q = Q_S \otimes Q_T$. Na situação em que Q_S é a matriz de precisão resultante do modelo ICAR, podemos usar essa estratégia no pacote **INLA**.

Modelos sem covariáveis

Na modelagem com efeitos aleatórios espaciais para dados de áreas, precisamos definir a matriz (ou lista) de vizinhança entre as áreas. Assim, vamos inicialmente usar um mapa do Paraná dividido em municípios, obtido do site do IBGE. Utilizamos o pacote **spdep** (Bivand et al., 2012) e suas dependência para importar e representar os mapas no R.

```
require(spdep)
pr.m <- readShapePoly("~/mapas/41mu2500gc")
```

Após importar o mapa para para o R, selecionamos a parte do mapa com apenas os municípios da mesoregião de Curitiba, criamos a lista de vizinhança e salvamos em arquivo para uso pelo **INLA**:

```
cwbm <- pr.m[pr.m$MESOREGIAO=="METROPOLITANA DE CURITIBA",]
nbm <- poly2nb(cwbm)
nb2INLA(file="dados/mesoc.g", nbm)
```

Para a análise usando o pacote **INLA**, empilhamos os dados de cada ano, isto é, as N primeiras linhas são dados do primeiro ano. Os dados possuem as seguinte colunas: `ano`, `código do município`, `número de nascidos vivos`, `número de óbitos infantis`, `imuno`, `bcg`, `polio`, `i` (um índice de município variando de 1 a 37) e `t` (um índice de tempo variando de 1 a 15). Os dados (atributos dos municípios) são lidos e verificados com os comandos a seguir.

```
micwb <- read.csv2("dados/micwb.csv", fileEnc="latin1")
head(micwb, 2) ## primeiras linhas
```

```
  ano    mun nasc obi imuno   bcg  polio i t
1 1995 411320  445  NA 17.65 39.11  39.00 1 1
2 1995 412010   58  NA 45.23 95.29 102.35 2 1
```

```
tail(micwb, 2) ## ultimas linhas
```

```
     ano    mun nasc obi  imuno    bcg  polio  i  t
554 2009 410950  104   1  92.19 109.01 117.12 36 15
555 2009 411995  278   3 100.72 124.58 125.00 37 15
```

Precisamos calcular o número esperado de óbitos. Vamos considerar uma taxa para cada ano. Assim, eliminamos a tendência temporal comum a todos os municípios e focamos a modelagem nas particularidades de cada município. Alguns municípios pequenos não tiveram nascidos vivos em alguns dos anos. Para calcular o número esperado, consideramos $n_{ij} = 0.5$ nesses casos. Para o ano de 1994, consideramos taxa de 1995.

```
ob.a <- tapply(micwb$obi, micwb$t, sum, na.rm=TRUE)
nc.a <- tapply(micwb$nasc, micwb$t, sum, na.rm=TRUE)
tx.a <- ob.a/nc.a
tx.a[1] <- tx.a[2]
micwb$nasc[micwb$nasc==0] <- 0.5
micwb$esperado <- micwb$nasc * rep(tx.a, each=length(nbm))
```

A SMR (da sigla em inglês para razão de mortalidade padronizada) observada pode ser calculada por

```
smro <- micwb$obi/micwb$esperado
```

Para avaliação, vamos definir uma coluna adicional nos dados com os óbitos até 2008 e colocando NA's para o ano 2009. Esta será a variável resposta na modelagem.

```
micwb$y <- micwb$obi
micwb$y[micwb$ano==2009] <- NA
```

Vamos considerar um conjunto de modelos de complexidade crescente para $\eta_{i,t}$:

$m_0:$	α_0
$m_1:$	$\alpha_0 + \alpha_{0,t}$
$m_2:$	$\alpha_0 + \alpha_{i,0}$
$m_3:$	$\alpha_0 + \alpha_{0,t} + \alpha_{i,0}$
$m_4:$	$\alpha_0 + \alpha_{i,t}$

em que:

- m_0: não temos modelagem do risco relativo;
- m_1: $\alpha_{0,t}$ é um processo autoregressivo no tempo, comum a todas as áreas;
- m_2: $\alpha_{i,0}$ é um processo autoregressivo no espaço, comum a todos os tempos;
- m_3: temos a soma de ambos os anteriores
- m_4: temos o modelo dinâmico para $\alpha_{i,t}$

Vamos criar uma lista de cinco fórmulas para especificar o conjunto de cinco modelos.

```
forms <- list(m0=y ~ 1,
              m1=y ~ 1 + f(i, model="ar1"),
              m2=y ~ 1 + f(i, model="besag", graph="dados/mesoc.g"),
              m3=y ~ 1 + f(t, model="ar1") +
              f(i, model="besag", graph="dados/mesoc.g"),
              m4=y ~ 1 + f(i, model="besag", graph="dados/mesoc.g",
                group=t, control.group=list(model="ar1",
                                hyper=list(theta=list(param=c(0,1))))))
```

e usar a função `inla()` para aproximar as distribuições a posteriori e o cálculo é feito em poucos segundos nos computadores atuais.

```
require(INLA)
mods <- lapply(forms, inla, family="Poisson",
               data=micwb, E=micwb$esperado,
               control.predictor=list(compute=TRUE),
               control.compute=list(dic=TRUE))
sapply(mods, function(x) x$cpu.used)
```

```
                    m0        m1        m2        m3         m4
Pre-processing  0.5307660 0.4286566 0.4650400 0.5396781  0.4900405
Running inla    1.3110819 2.7041285 1.8977787 6.9944375 24.4648252
Post-processing 0.3522794 0.4051874 0.4052265 0.4419107  0.9067664
Total           2.1941273 3.5379725 2.7680452 7.9760263 25.8616321
```

Podemos ver o valor do DIC para cada modelo e verificamos que o modelo dinâmico é o que apresentou o menor DIC.

```
sapply(mods, function(x) x$dic[[1]])
```

```
      m0       m1       m2       m3       m4
2951.815 2499.458 2495.070 2496.099 2470.631
```

É recomendável verificar o ajuste do modelo aos dados. Um diagnóstico simples é visualizar os dados observados e o valor estimado pelo modelo dado por $E_{i,t}\mathrm{e}\{\psi_{i,t}\}$. Vamos inicialmente organizar o risco relativo estimado pelo modelo 4 e o número de óbitos estimado.

```
smr4 <- matrix(exp(mods$m4$summary.linear.pred$m), 37)
est <- micwb$esperado*smr4
```

Também podemos comparar o comportamento da SMR observada e a SMR estimada pelo modelo em relação ao número de óbitos.

Na Figura 6.8 temos o diagrama de dispersão entre o número de casos e a SMR calculada pela razão entre o número observado de casos e o número esperado de casos, isto é, a SMR observada. Observamos neste gráfico uma grande variação na SMR quando o número de óbitos é pequeno. Esta variação na SMR decai a medida que o número de óbitos aumenta. Esse fato expressa bem a elevada variação em taxas de municípios pequenos, com poucos nascidos vivos e, consequentemente, poucos óbitos. No gráfico do meio, temos a dispersão entre o número observado de casos e a SMR estimada pelo modelo. Neste caso, notamos padrão parecido, porém, agora a SMR estimada não apresenta valores muito extremos como a observada. No gráfico da direita temos a dispersão entre o número de casos estimado pelo modelo e o número observado de casos. Neste gráfico, observamos que há um bom ajuste do modelo.

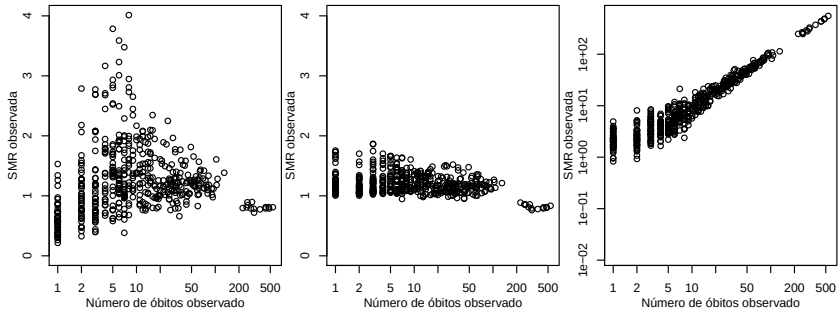

Figura 6.8: SMR observada pelo número de óbitos (esquerda), SMR estimada pelo número de óbitos (meio) e número estimado pelo número de óbitos observado.

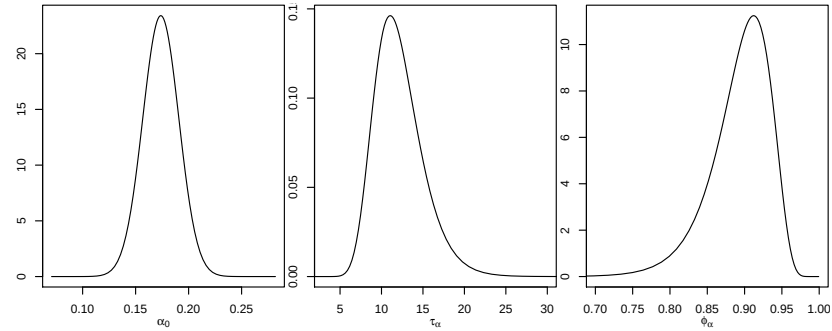

Figura 6.9: Densidade dos hiperparâmetros a posteriori.

Na Figura 6.9 temos a densidade à posteriori para α_0, τ_{w1} e para ϕ_α, considerando o modelo m_4. A seguir, temos a média, desvio padrão e intervalo de 95% de credibilidade.

```
 mods$m4$summary.fix[ ,c(1,2,3,5)]
```

```
      mean         sd 0.025quant 0.975quant
0.17372634 0.01705562 0.14022568 0.20713809
```

```
 mods$m4$summary.hyp[ ,c(1,2,3,5)]
```

```
                      mean         sd 0.025quant 0.975quant
Precision for i 12.1344197 2.94144908  7.4410901 18.9158045
GroupRho for i   0.8934278 0.04029005  0.7968966  0.9525941
```

Os valores de α_0 estão em torno de 0.14 a 0.21. A evolução de α no tempo é bastante suave, com ϕ_α a posteriori em torno de 0.80 a 0.95. Com τ_α em torno de 7.4 a 18.9, $w1$ tem variância em torno de 0.05 a 0.2.

Temos a distribuição marginal a posteriori de cada $\alpha_{i,t}$. Podemos visualizar as séries para alguns municípios ou visualizar o mapa para alguns

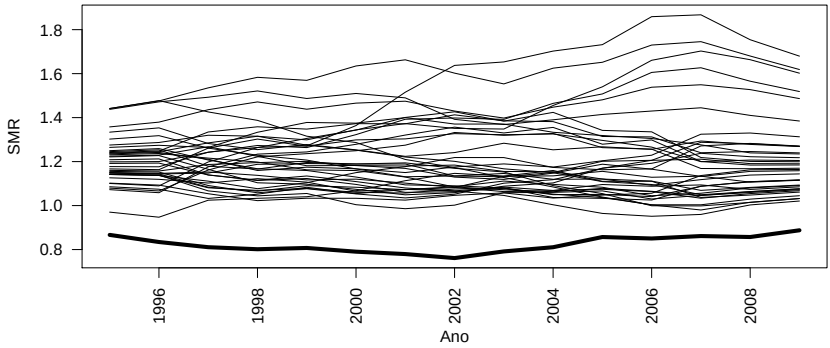

Figura 6.10: Séries temporais da SMR estimada pelo modelo 4. Linha grossa indica o município de Curitiba.

tempos. Também temos estatísticas resumo de $\psi_{i,t}$, o logaritmo do risco relativo, a posteriori.

Na Figura 6.10 podemos visualizar a SMR estimada pelo modelo 4. A linha mais larga é para o município de Curitiba. Espera-se que a média da SMR seja igual a 1. Como Curitiba tem mais nascidos vivos e apresenta SMR bem abaixo dos demais, a maioria dos municípios apresenta SMR maior que 1.

Na Figura 6.11 temos a SMR estimada pelo modelo 4 para os anos 1995, 2000, 2004 e 2009. Como havíamos visto, Curitiba tem o risco mais baixo. Embora o número de áreas seja pequeno, é perceptível uma suavidade na variação espacial do risco. Os municípios a leste (do litoral) apresentam um valor menor. E alguns municípios ao norte de Curitiba são os que apresentam os maiores valores.

```
q <- quantile(smr4, 0:5/5)
par(mfrow=c(2,2), mar=c(0,0,1,0))
for (i in c(1, 6, 10, 15)) {
  plot(cwbm, col=gray(1-(smr4[,1]-q[1])/(q[6]-q[1])))
  title(main=paste("Ano", 1995:2009)[i])
}
image(x=c(-50.3,-50.2), y=seq(-25.5,-24.5,leng=21),
      z=matrix(quantile(smr4, 1:20/21), 1),  col=gray(19:0/19),
      brea=quantile(smr4, 0:20/20), add=TRUE)
text(rep(-50.1,6), seq(-25.5, -24.5, leng=6), format(q,dig=2))
```

Os seis municípios que apresentam o maior risco médio estimado ao longo dos anos pelo modelo 4 podem ser extraídos.

```
    GEOCODIGO             NOME
376   4105201        Cerro Azul
391   4127882   Tunas do Paraná
375   4122206 Rio Branco do Sul
385   4103107   Bocaiúva do Sul
369   4128633    Doutor Ulysses
```

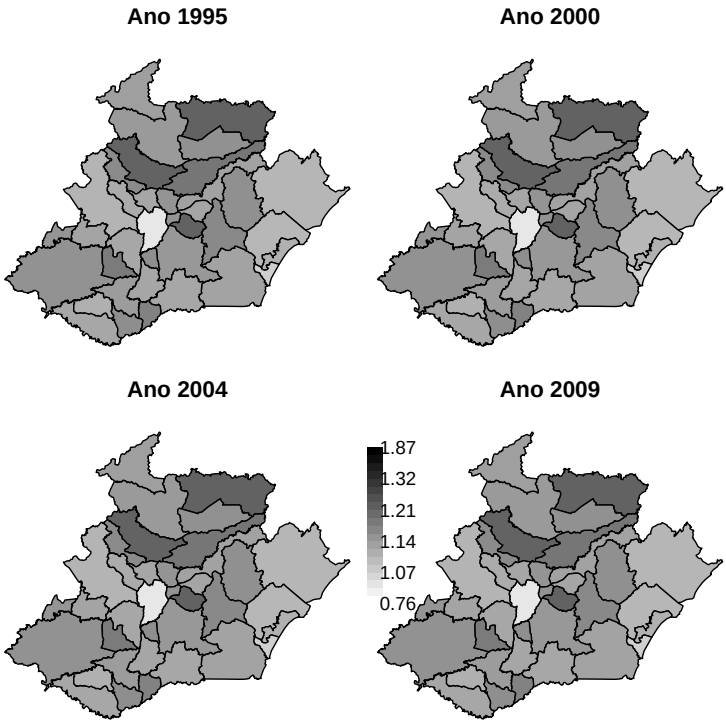

Figura 6.11: Mapas do risco relativo estimado pelo modelo 4 para alguns dos anos.

```
390   4100202      Adrianópolis
```

Agora, vamos considerar Curitiba e mais três municípios selecionados aleatoriamente. e visualizar as séries de dados observados para esses municípios, juntamente com o valor estimado pelo modelo. Vamos considerar também intervalo de 95% de credibilidade multiplicando $E_{i,t}$ pelo exponencial dos quantis 2,5% e 97,5% de $\psi_{i,t}$. As séries para esses quatro municípios são visualizadas na Figura 6.12.

```
set.seed(123)
(msel <- c("Curitiba", sample(setdiff(cwbm$NOME, "Curitiba"),3)))
```

```
[1] "Curitiba"          "Itaperuçu"         "Adrianópolis"
[4] "Rio Branco do Sul"
```

```
li.e <- matrix(micwb$esperado*exp(mods$m4$summary.linear.pred[,3]), 37)
ls.e <- matrix(micwb$esperado*exp(mods$m4$summary.linear.pred[,5]), 37)
```

Na Figura 6.12, notamos que, em geral, número de óbitos estimados está razoavelmente próximo dos observados. Observamos que para os municípios com número de casos menor, a discrepância entre o observado e

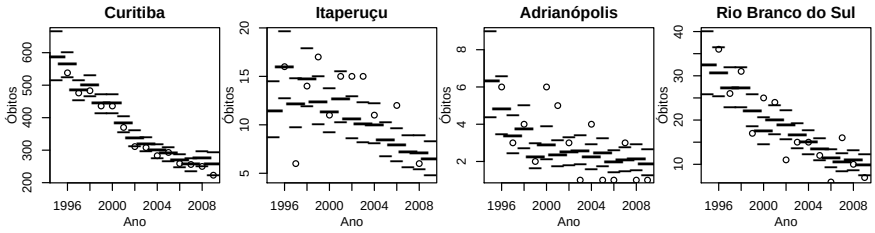

Figura 6.12: Séries de número de óbitos infantis para os quatro municípios, valor predito e intervalo de 95% de credibilidade.

o estimado é maior. Isso também pode ser visto no gráfico da direita da Figura 6.8.

Referências

A presente versão do texto trata apenas de inferência baseada na função de verossimilhança. No Capítulo 2 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 3 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 4 apresenta os modelos de regressão com efeitos aleatórios, nele apresentamos como exemplo inicial o modelo geoestatístico, e nos baseamos em 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 re-

ferências. 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 & R Development Core Team (2012).

Nesta segunda parte do livro nos tratamos brevemente da abordagem Bayesiana para estimação de parâmetros. Muitos livros cobrem esta abordagem, citamos como as principais referências para a construção deste texto os seguintes livros: Gelman et al. (2003), Banerjee et al. (2004) entre outras. Para a parte de modelos hierárquicos Bayesianos tanto com efeitos fixos como com aleatórios uma referência muito comum é Banerjee et al. (2004). Como pacotes adicionais foram utilizados o *MCMCpack* (Martin et al., 2011), o pacote *dclone* (Sólymos, 2010) junto com o software JAGS (Plummer, 2003). Gilks et al. (1996) foi talvez o primeiro livro a apresentar MCMC voltado a modelagem estatística de uma forma completa e abrangente à época e ainda hoje permanece uma referência para os conceito fundamentais do tema. Com o desenvolvimento da área e da modelagem estatística uma série de outros textos surgiram, sejam voltados para os aspectos de fundamentos ou computacionais, ou com aplicações e modelos estatísticos de mais simples a altamente complexos. Gammerman & Lopes (2006) é um dos textos dedicado ao tema. Robert & Casella (2004) discutem métodos Monte Carlo de forma mais geral e uma versão complementar e aplicada com ênfase nos aspectos computacionais utilizando o R é apresentada em Robert & Casella (2010).

A literatura sobre modelos hierárquicos sob a perspectiva Bayesiana é extensa. Fazemos aqui apenas duas recomendações iniciais como possíveis introduções ao tema. A primeira é o texto de Schmidt (2008) apresentado no 18^o SINAPE que oferece uma excelente introdução aos princípios e aspectos computacionais relacionados e esta rica classe de modelos. Modelos hieráquicos são também discutidos em detalhes e através de um extensa seleção de exemplos em Gelman & Hill (2006).

Referências Bibliográficas

AITKIN, M. **Statistical Inference. An integrated Bayesian Likelihood Approach**. Boca Raton, FL: Chapman & Hall/CRC, 2010.

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.

ASSUNÇÃO, R. M.; RODRIQUES, E. C.; KRAINSKI, E. T. **Campos Aleatórios de Markov e Distribuições Especificadas Através das Densidades Condicionais**. Associação Brasileira de Estatística, 2010.

AZZALINI, A. **Statistical Inference Based on the likelihood**. Chapman and Hall/CRC, 1996. 352p.

BADDELEY, A.; TURNER, R. Spatstat: an R package for analyzing spatial point patterns. **Journal of Statistical Software**, v.12, n.6, p.1–42, 2005, ISSN 1548-7660.

BANERJEE, S.; CARLIN, B. P.; GELFAND, A. E. **Hierarchical Modeling and Analysis for Spatial Data**. London: Chapman & Hall, 2004.

BESAG, J.; KOOPERBERG, C. On conditional and intrinsic autoregressions. **Biometrika**, v.82, n.4, p.733–746, 1995.

BIVAND, R.; WITH CONTRIBUTIONS BY MICAH ALTMAN; ANSELIN, L.; ASSUNÇÃO, R.; BERKE, O.; BERNAT, A.; BLANCHET, G.; BLANKMEYER, E.; CARVALHO, M.; CHRISTENSEN, B.; CHUN, Y.; DORMANN, C.; DRAY, S.; HALBERSMA, R.; KRAINSKI, E.; LEGENDRE, P.; LEWIN-KOH, N.; LI, H.; MA, J.; MILLO, G.; MUELLER, W.; ONO, H.; PERES-NETO, P.; PIRAS, G.; REDER, M.; TIEFELSDORF, M.; YU., D. **spdep: Spatial dependence: weighting schemes, statistics and models**, 2012. R package version 0.5-45.

BOLKER, B.; R DEVELOPMENT CORE TEAM. **bbmle: Tools for general maximum likelihood estimation**, 2012. R package version 1.0.4.1.

BOX, G. E. P.; TIAO, G. C. **Bayesian Inference in Statistical Analysis**. Wiley-Interscience, 1992. 608p.

CAMELETTI, M.; LINDGREN, F.; SIMPSON, D.; RUE, H. Spatio-temporal modeling of particulate matter concentration through the SPDE approach. Norwegian University of Science and Technology, 2011. Rel. téc.

CARTER, C.; KOHN, R. On Gibbs sampling for state space model. **Biometrika**, v.81, n.3, p.541–553, 1994.

CARVALHO, M. S.; ANDREOZZI, V. L.; CODEÇO, C. T.; CAMPOS, D. P.; BARBOSA, M. T. S.; SHIMAKURA, S. E. **Análise de Sobrevivência: teoria e aplicações em saúde**. 2. ed. Editora Fiocruz, 2011. 432 p.

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.

DEGROOT, M. H. **Optimal Statistical Decisions**. Hardcover, 1970.

DIGGLE, P. J. **Statistical Analysis of Spatial Point Patterns**. 2. ed. Hodder Education Publishers, 2003.

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.

EIDSVIK, J.; MARTINO, S.; RUE, H. Approximate Bayesian inference in spatial generalized linear mixed models. **Scandinavian Journal of Statistics**, v.36, p.1–22, 2009.

FAHRMEIR, L.; TUTZ, G. **Multivariate Statistical Modelling Based on Generalized Linear Models**. 2. ed. Berlin: Springer-Verlag, 2001.

FOX, J.; WEISBERG, S. **An R Companion to Applied Regression**. 2. ed. Thousand Oaks, CA, USA: Sage Publications, 2011.

FRÜWIRTH-SCHNATTER, S. Data augmentation and dynamic linear mo-

dels. **Journal of Time Series Analysis**, v.15, n.2, p.183–202, 1994.

GAMMERMAN, D.; LOPES, G. F. **Markov chain Monte Carlo: stochastic simulation for Bayesian inference**. Taylor & Francis, 2006.

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.

GELFAND, A. E.; SMITH, A. F. M. Sampling-Based Approaches to Calculating Marginal Densities. **Sampling-Based Approaches to Calculating Marginal Densities**, v.85, n.410, p.398–409, 1990.

GELMAN, A.; CARLIN, J. B.; STERN, H. S.; RUBIN, D. B. **Bayesian Data Analysis, Second Edition (Chapman & Hall/CRC Texts in Statistical Science)**. Chapman and Hall/CRC, 2003. 696p.

GELMAN, A.; HILL, J. **Data Analysis Using Regression and Multilevel/Hierarchical Models**. Cambridge University Press, 2006. 648p.

GEMAN, S.; GEMAN, D. Stochastic Relaxation, Gibbs Distributions and the Bayesian Restoration of Images. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v.6, p.721–741, 1984.

GILAT, A.; SUBRAMANIAM, V. **Numerical Methods with MATLAB**. Wiley, 2010. 512p.

GILKS, W. R.; RICHARDSON, S.; SPIEGELHALTER, D. J. E. **Markov Chain Monte Carlo in Practice**. London: Chapman and Hall, 1996.

GIOLO, S. R.; COLOSIMO, E. A. **Análise de sobreviência aplicada**. Edgard Blucher, 2006. 392 p.

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.

HASTINGS, W. K. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. **Biometrika**, v.57, n.1, p.97–109, 1970.

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.

KUSS, M.; RASMUSSEN, C. E. Assessing approximate inference for binary Gaussian process classification. **Journal of Machine Learning Research**, v.6, p.1679–1704, 2005.

LELE, S. R. Estimability and Likelihood Inference for Generalized Linear Mixed Models Using Data Cloning. **Journal of the American Statistical Association**, v.105, n.492, p.1617–1625, 2010a.

LELE, S. R. Model complexity and information in the data: Could it be a house built on sand? **Ecology**, v.91, n.12, p.3493–3496, 2010b.

MARTIN, A. D.; QUINN, K. M.; PARK, J. H. MCMCpack: Markov Chain Monte Carlo in R. **Journal of Statistical Software**, 2011.

MCCULLAGH, P.; NELDER, J. A. **Generalized Linear Models**. Second ed ed. Chapman and Hall/CRC, 1989. 532p.

METROPOLIS, N.; ROSENBLUTH, A. W.; ROSENBLUTH, M. N.; TELLER, A. H.; TELLER, E. Equation of State Calculations by Fast Computing Machines. **The Journal of Chemical Physics**, v.21, n.6, p.1087–1092, 1953.

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.

PEMSTEIN, D.; QUINN, K. M.; MARTIN, A. D. The Scythe Statistical Library: An Open Source C++ Library for Statistical Computation. **Journal of Statistical Software**, v.42, n.12, 2011.

PETRIS, G.; PETRONE, S.; CAMPAGNOLI, P. **Dynamic Linear Models with R**. Springer-Verlag, New York, 2009.

PLUMMER, M. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling, 2003.

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, 2004. 675p.

ROBERT, C.; CASELLA, G. **Introducing Monte Carlo Methods with R**. Springer, 2010. 284p.

ROYALL, R. **Statistical evidence**. Chapman and Hall, 1997.

RUE, H. Fast sampling of Gaussian Markov random fields. **Journal of the Royal Statistical Society, Series B**, v.63, p.325–338, 2001.

RUE, H.; HELD, L. **Gaussian Markov Random Fields: Theory and Applications**. London: Chapman & Hall, 2005.

RUE, H.; MARTINO, S. Approximate Bayesian inference for hierarchical Gaussian Markov random fields models. **Journal of statistical Planning and Inference**, v.137, p.3177–3192, 2007.

RUE, H.; MARTINO, S.; CHOPIN, N. Approximate Bayesian inference for latent Gaussian models using integrated nested Laplace approximations. **Journal Royal Statistical Society B**, v.71, p.319–392, 2009a.

RUE, H.; MARTINO, S.; LINDGREN, F. **INLA: Functions which allow to perform a full Bayesian analysis of structured (geo-)additive models using Integrated Nested Laplace Approximaxion**, 2009b. R package version 0.0.

RUE, H. V.; MARTINO, S.; CHOPIN, N. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. **Journal of the Royal Statistical Society: Series B (Statistical Methodology)**, v.71, n.2, p.319–392, 2009c.

RUIZ-CÁRDENAS, R.; KRAINSKI, E. T.; RUE, H. Direct fitting of dynamic models using integrated nested Laplace approximations - INLA. **Computational Statistics and Data Analysis**, v.56, n.6, p.1808–1828, 2012.

SCHMIDT, H. S. M. . A. D. S. . A. M. **Modelo hierárquicos e aplicações**. Associação Brasileira de Estatística, 2008. 279p.

SCHWEPPE, F. C. Evaluation of likelihood functions for Gaussian signals. **IEEE Trans. Info. Theory**, v.11, p.61–70, 1965.

SHEPHARD, N. Partial Non-Gaussian State Space. **Biometrika**, v.81, n.1, p.115–131, 1994.

SMITH, A. F. M.; SKENE, A. M.; SHAW, J. E. H.; NEYLOR, J. C. Progress with numerical and graphical methods for practical Bayesian statistics. **Journal of statistical Planning and Inference**, v.36, p.75–82, 1987.

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.

SÓLYMOS, P. dclone: Data Cloning in R. **The R Journal**, v.2, n.2, p.29–37, 2010.

SPIEGELHALTER, D. J.; BEST, N. G.; CARLIN, B. P.; VAN DER LINDE, A. Bayesian measures of model complexity and fit (with discussion). **Journal of the Royal Statistical Society, Series B**, v.64, p.583–639, 2001.

TOLOI, P. A. M. . C. M. C. **Analise de Series Temporais**. Associação Brasileira de Estatística e Edgard Blücher, 2004. 544p. Projeto Fisher.

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**. 4. ed. New York: Springer, 2002. 512p.

VIVAR, J. C. Modelos espaço-temporais para dados de área na família exponencial, 2007. Tese (Doutorado) - Universidade Federal do Rio de Janeiro.

VIVAR, J. C.; FERREIRA, M. A. R. Spatio-temporal models for Gaussian areal data. **Journal of Computational and Graphical Statistics**, v.18, n.3, p.658–674, 2009.

WEST, M.; HARRISON, P. J. **Bayesian Forecasting & Dynamic Models**. 2. ed. Springer Verlag, 1997. 700p.

WINKELMANN, R. Duration Dependence and Dispersion in Count-Data Models. **Journal of Business & Economic Statistics**, v.13, n.4, p.467–474, 1995.