Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Estatística

Caio Gomes Alves

Estimação por máxima verossimilhança em modelos espaciais lineares mistos generalizados baseada na aproximação de Laplace

Curitiba
2024

Caio Gomes Alves

Estimação por máxima verossimilhança em modelos espaciais lineares mistos generalizados baseada na aproximação de Laplace

Trabalho de Conclusão de Curso apresentado à disciplina Laboratório B do Curso de Graduação em Estatística da Universidade Federal do Paraná, como exigência parcial para obtenção do grau de Bacharel em Estatística.

Orientador(a): Prof. Dr. Paulo Justiniano Ribeiro Junior

Curitiba
2024

Aos meus pais.

Agradecimentos

Agradeço imensamente aos meus pais, por sempre acreditarem em meu potencial, sendo uma base sólida durante toda a minha vida.

Aos meus amigos, por permanecerem ao meu lado durante os anos e terem me acompanhado nesta jornada.

Ao PET-Estatística, que foi parte fundamental da minha experiência acadêmica.

E aos meus professores, por me mostrarem como é possível ter maestria na arte de dar aula, por me incentivarem nas minhas aspirações acadêmicas e por fazerem renovar todos os dias o meu amor pela Estatística.

True ignorance is not the absence of knowledge, but the refusal to acquire it.
Karl Popper.

Resumo

A modelagem geoestatística para dados não-gaussianos usualmente se dá por meio de métodos baseados em reamostragem por MCMC (*Monte Carlo Markov Chain*), que apresentam problemas inerentes quanto à complexidade computacional e convergência dos estimadores, que são agravados com o crescimento da quantidade de dados amostrados. Este trabalho tem como objetivo análise e modificação de algoritmos computacionais para a estimação de parâmetros de modelos espaciais lineares mistos generalizados (*SGLMMs*), por meio da maximização da log-verossimilhança usando a aproximação de Laplace. Utilizou-se a linguagem de programação R, a avaliação das funções foi feita por meio de um estudo de simulação, com o objetivo de investigar o comportamento dos estimadores obtidos sob diferentes condições, que mostra que os mesmos possuem propriedades como o não-viés para amostras grandes e consistência. Posteriormente, foi realizado o ajuste de diferentes modelos à duas bases de dados reais, que possuem dados não-gaussianos, como motivação de aplicação dos métodos desenvolvidos.

Palavras-chave: Estatística Espacial. Modelos Mistos. Dados Não-Gaussianos.

Sumário

1 Introdução

Geoestatística refere-se ao conjunto de métodos utilizados para analisar dados que seguem a seguinte estrutura: considere o vetor de observações $y = (y_1, y_2, \ldots, y_n)^\top$, coletadas nas posiçoes $x_i = (x_1, x_2, \ldots, x_n)^\top$, de uma região A no espaço, mas que poderiam ter sido coletadas em qualquer outro conjunto de pontos arbitrários de A. Cada observação y é dada como uma realização parcial de um processo espacial contínuo não observado (latente), denotado $S(x)$, nos pontos amostrais x_i. Um exemplo desses processos espaciais contínuos é a temperatura em um local, pois todos os pontos no espaço possuem alguma temperatura, mas é impossível medí-la em todos os pontos, portanto selecionam-se uma série de pontos (estações meteorológicas, por exemplo) para se amostrar a temperatura, e as demais são inferidas desse processo.

A escolha de tais pontos pode ser aleatória, estruturada (em grades, em delineamentos amostrais, etc.) ou por conveniência. O objetivo principal da modelagem geoestatística é recuperar esse processo latente $S(x)$, para ser possível estimar os valores nos demais pontos não amostrados (Cressie (1993)), usualmente pela média do processo ($\hat{\mathbb{E}}[S(x)]$). Como esse processo não é diretamente observável, os modelos geoestatísticos podem ser classificados como modelos de efeitos aleatórios (mistos), com alguma estrutura de dependência espacial entre esses efeitos.

Contudo, em muitos casos, o fenômeno de interesse não pode ser bem modelado por uma distribuição Normal, como no caso de contagens e proporções, ou presença de heterocedasticidade nos dados. Para tanto, extensões foram propostas, para englobar distribuições da família exponencial que não sejam gaussianos. Dentre elas, Zhang (2002) e Banerjee, Carlin e Gelfand (2004) são exemplos em que o ajuste de modelos espaciais lineares mistos generalizados é feita por meio de métodos de MCMC (*Monte Carlo Markov Chain*), seguindo o paradigma Bayesiano.

Modelos baseados na análise da verossimilhança foram propostos por Christensen e Ribeiro Jr (2002), mas que ainda se baseiam em algoritmos MCMC. Além das propriedades ótimas de estimadores de máxima verossimilhança (Casella e Berger (2011)), esse tipo de estimador permite a obtenção de perfis de verossimilhança (para exploração intervalar dos estimadores) e a comparação direta entre modelos (por AIC, por exemplo). Na prática, o uso de métodos bayesianos apresentam problemas quanto à convergência dos algoritmos, além do alto tempo computacional necessário para a geração das estimativas.

A avaliação desses modelos é complicada, pois a maximização da verossimilhança exige a resolução de uma integral de alta dimensionalidade, e que não possui solução analítica nos casos não-gaussianos. Tendo isso em mente, Bonat e Ribeiro Jr (2016) propõem um método baseado na aproximação de Laplace, uma técnica matemática que, por meio de transformações no integrando, permite trocar um problema de integração multidimensional por um problema de maximização multidimensional, sem ocorrer a perda

significativa de informação. No material suplementar do artigo, os autores apresentam códigos na linguagem de programação R (R Core Team (2024)), com funções usadas para realizar a estimação dos parâmetros, e comparam os resultados obtidos com os gerados por algoritmos MCMC.

Levando isso em consideração, este trabalho foi desenvolvido com o intuito de analisar as funções apresentadas em Bonat e Ribeiro Jr (2016), modificando-as para simplificar e unificar a sintaxe, explorar as propriedades dos estimadores gerados por meio de um estudo de simulação e usar as função em duas aplicações a dados reais.

O trabalho está dividido da seguinte maneira: no capítulo 2 é apresentada uma revisão de literatura, com explicações mais aprofundadas sobre Modelos Espaciais Lineares Mistos Generalizados (*Spacial Generalized Linear Mixed Models, ou SGLMMs, em inglês*), a teoria por trás da aproximação de Laplace e a justificativa do seu uso nessa classe de modelos. No capítulo 3 são apresentadas as funções usadas na modelagem, as bases de dados utilizadas e os recursos computacionais utilizados. No capítulo 4 é feito um estudo de simulação, para avaliar o comportamento dos estimadores obtidos pela aproximação de Laplace, sob diferentes condições, e comparar os resultados com os gerados por algoritmos MCMC. Após isso, ainda no capítulo 4, é feito o ajuste do modelo em duas bases de dados reais, para motivar o potencial de aplicações dessas funções. Os apêndices apresentam o código R usado para definir cada função, bem como os códigos utilizados para os ajustes.

2 Revisão de Literaura

2.1 Modelos Espaciais Lineares Mistos Generalizados

Usualmente, considera-se que os dados y_i provenientes de um processo geoestatístico, coletados (leia-se medidos) no conjunto de pontos $\{x_i, i = 1, \ldots, n\}$, possuem um processo contínuo latente $S(x)$ sobre toda a região amostral, que dita como esses dados são gerados. Para isso, considere que $S(x)$ é um processo gaussiano com média $\mu(x_i)$ e variância σ^2, com função de correlação entre os pontos denotada por $\rho(u) = Corr\{S(x), S(x')\}$, em que u denota a distância entre x e x' e que condicionais em $S(x)$, os y_i são realizações de variáveis aleatórias mutualmente independentes $Y(x_i)$, normalmente distribuídas, com médias condicionais $\mathbb{E}[Y(x_i)|S(x)] = S(x_i)$ e variâncias condicionais τ^2.

Levando isso em consideração, o modelo para $Y(x_i)$ é dado por $Y_i = S(x_i) + Z_i$, com Z_i sendo variáveis aleatórias mutualmente independentes e normalmente distribuídas, com média 0 e variância τ^2. Quando as observações desse processo geoestatístico não possuem respostas gaussianas, como contagens ou proporções, usamos o seguinte modelo hierárquico para denotar o modelo espacial linear misto generalizado:

$$\begin{aligned} [Y(x)|S(x)] &\sim f(.; \mu(x), \psi) \\ g(\mu(x)) &= D\beta + S(x) \end{aligned} \tag{2.1}$$

Assume-se que os componentes de $Y(x)$ são condicionalmente independentes dado o processo latente $S(x) = \sigma U(x; \phi) + \tau Z$. O preditor linear do modelo é ligado à média por meio da uma função de ligação $\mathbf{g}$. Os efeitos fixos são denotados por $D\beta$, onde D representa a matriz $n \times p$ de delineamento experimental, contendo p covariáveis, e β um vetor $p \times 1$ de parâmetros de regressão a serem estimados.

Os efeitos aleatórios espacialmente correlacionados são dados por $\sigma U(x; \phi)$, onde σ (*sill*) denota a variância dos efeitos aleatórios e $U(x; \phi)$ é a variância unitária de um Campo Gaussiano Aleatório (*Gaussian Random Field*), com função de correlação $\rho(u, \phi)$, que descreve a estrutura da dependência espacial entre os elementos de $U(x; \phi)$. Os efeitos aleatórios não correlacionados são dados por $\tau Z \sim N(0, \tau^2 I)$, onde τ^2 (normalmente chamado de efeito pepita, ou *nugget*) representa a soma das variações não-espaciais e de micro escala no modelo. Dessa forma, a parte dos efeitos aleatórios (completos) do preditor linear é gaussiana, com matriz de covariância $\Sigma = \sigma^2 U(x; \phi) + \tau^2 I$.

Assume-se que a função de correlação ρ seja positiva definida, e dependa apenas da distância entre dois pontos, dada por $u_{ij} = ||x_i - x_j||$. Existem diversas funções válidas para ρ, dentre elas as mais usuais são a exponencial, a esférica e a *Matérn*. A função de correlação exponencial tem a forma $\rho(u; \phi) = \exp(-\frac{u}{\phi})$, e possui decaimento rápido, mas nunca chega a zero, de modo que pontos distantes ainda sejam correlacionados, ainda que muito fracamente. A função de correlação esférica tem a forma $\rho(u; \phi) = 1 - \frac{3}{2}\left(\frac{u}{\phi}\right) + \frac{1}{2}\left(\frac{u}{\phi}\right)^3$,

caso $u < \phi$, e zero caso contrário. É usado em casos que a correlação entre os pontos é suave até um ponto (ϕ, chamado de alcance) e zero para dados mais distantes que isso.

A função de corelação Matèrn tem a forma $\rho(u;\phi,\kappa) = \{2^{\kappa-1}\Gamma(\kappa)\}^{-1}\left(\frac{u}{\phi}\right)^{\kappa} K_{\kappa}\left(\frac{u}{\phi}\right)$, em que $K_{\kappa}(\cdot)$ denota a função de Bessel modificada de segunda espécie, de ordem κ, e $\Gamma(\cdot)$ é a função Gamma convencional. O parâmetro κ controla a suavidade e o decaimento da corelação entre os pontos, assim, quanto maior o valor de κ, mais suave é o decaimento. Vale citar que a função de correlação exponencial é um caso particular da Matèrn, com $\kappa = 0.5$.

A definição correta da função de correlação espacial é crucial para uma boa modelagem dos dados, e nesse sentido a Matèrn é a mais flexível, podendo assumir diferentes formas dependendo de κ. A estimação de κ é complicada, e em Diggle e Ribeiro Jr. (2007) os autores propõem uma heurísitca onde diferentes modelos são ajustados considerando um *grid* para κ, usualmente $\kappa = \{1, 1.5, 2, 2.5, \dots\}$, e verifica-se qual gera o melhor ajuste.

2.2 Estimação dos Parâmetros

Um dos principais objetivos da modelagem geoestatística é a estimação do vetor de parâmetros $\theta = (\beta, \sigma^2, \tau^2, \phi, \psi)$. Em *Model-based Geostatistics*, (Diggle e Ribeiro Jr. (2007)), os autores propõem uma abordagem de estimação baseada na maximização da função de verossimilhança marginal, visto que a superfície da log-verossimilhança do modelo é multidimensional, e de difícil investigação direta.

Supondo um modelo com parâmetros (α, ν), com verossimilhança denotada por $L(\alpha, \nu)$, a função de verossimilhança marginal para α é definida como:

$$L_p(\alpha) = L(\alpha, \hat{\nu}(\alpha)) = \max_{\nu}(L(\alpha, \nu)) \tag{2.2}$$

Ou seja, considera-se a variação da função de verossimilhança com relação a α quando, para cada valor de α, é definido para ν o valor que maximiza a log-verossimilhança com α fixado. Assim, reduz-se a dimensionalidade da superfície de verossimilhança, facilitando a inspeção. Além disso, é possível calcular intervalos de confiança (aproximados) para parâmetros individuais, de maneira similar aos casos uniparamétricos para a log-verossimilhança.

A aplicação de métodos baseados em verossimilhança a modelos geoestatísticos para dados não-gaussianos é complicada e possui dificuldades computacionais, que surgem devido à alta dimensionalidade do vetor de efeitos aleatórios $S(x) = \{S(x_1), \dots, S(x_n)\}$.

A maximização da função de verossimilhança marginal do modelo é obtida integrando-se os efeitos aleatórios da distribuição conjunta definida em (2.1), como segue:

$$L_p(\theta; y(x)) = \int_{\Re^n} f(y(x)|S(x)) f(S(x)) dS(x) \tag{2.3}$$

O primeiro termo do produto dado em (2.3) é a distribuição amostral de $y(x)$, dado o vetor de efeitos aleatórios $S(x)$, enquanto que assume-se que o segundo possua distribuição Normal Multivariada. Exceto no caso em que a distribuição de $f(y(x)|S(x))$ também é gaussiana, a verossimilhança marginal é analiticamente intratável, por se tratar do produto de duas distribuições.

Dessa forma, a maximização da função de verossimilhança marginal requer a solução de integrais complexas. No contexto geoestatístico, os valores do vetor $S(x)$ são dependentes, de modo que a integral em (2.3) possui tantas dimensões quanto observações na amostra coletada. Com isso, métodos de integração numérica, como quadratura gaussiana, quadratura de Gauss-Hermite e Gauss-Hermite adaptativo (Pinheiro e Bates (1995)), são problemáticos, pois a acurácia é afetada pela alta dimensionalidade (Breslow e Clayton (1993)).

Outros métodos foram propostos para a aproximação de (2.3), dentre eles os mais comuns são a de verossimilhança hierárquica (obtida através de penalização de acordo com a verossimilhança da distribuição assumida para $S(x)$), proposta por Lee e Nelder (1996) e aprimorada por Banerjee, Carlin e Gelfand (2004). Métodos de integração por Monte Carlo foram propostos, com Geyer e Thompson (1992) contribuindo com a teoria para maximização para dados correlacionados/dependentes. Zhang (2002) desenvolveu versões baseadas no algoritmo EM para estimação dos parâmetros. Christensen (2004) descreve uma metodologia baseada em aproximações utilizando algoritmo MCMC (*Monte Carlo Marokv Chain*) para simular da distribuição condicional de $S(x)$.

Apesar de bem-desenvolvidos, os métodos baseados em integração por Monte Carlo são computacionalmente intensivos, lentos na estimação e precisam ser verificadas quanto à convergência e acurácia (Geyer (1994), McCulloch (1997)). Com isso em mente, Bonat e Ribeiro Jr (2016) propõem uma abordagem baseada em aproximação de Laplace para a integral em (2.3).

2.3 Aproximação de Laplace

Normalmente utilizado para análise de dados longitudinais, a aproximação de Laplace (Tierney e Kadane (1986)) é um método utilizado para aproximar o integrando para obter uma expressão fechada analiticamente tratável, permitindo assim a maximização da forma aproximada da verossimilhança marginal. O método é utilizado para aproximar integrais da seguinte forma:

$$\int_{\Re^n} \exp\left(\varrho(u)\, du\right) \approx (2\pi)^{n/2} \left|-\varrho''(\hat{u})\right|^{-1/2} \exp\left(\varrho(\hat{u})\right) \tag{2.4}$$

Em que $\varrho(u)$ é uma função unimodal e limitada, de uma variável u n-dimensional, sendo $\hat{u}$ o valor para a qual $\varrho(u)$ é maximizado. Portanto, muda-se um problema de integração para um problema de maximização multidimensional (que são normalmente

melhor comportados), sendo necessário maximizar o o integrando e o hessiano ($\varrho''(\hat{u})$) analítica ou numericamente.

Assumindo que a distribuição $f(y(x)|S(x))$ seja da família exponencial de distribuições (Binomial, Poisson, Beta, Gamma, Binomial Negativa, etc.), podendo ser escrita da seguinte forma:

$$f(y(x)|S(x);\beta) = \exp\left\{y(x)^\top(D\beta + S(x)) - 1^\top b(D\beta + S(x)) + 1^\top c(y(x))\right\} \tag{2.5}$$

Sendo $b(.)$ e $c(.)$ funções conhecidas, e considerando a distribuição Normal Multivariada, dada por:

$$f(S(x);\Sigma) = (2\pi)^{-n/2}|\Sigma|^{-1/2}\exp\left\{-\frac{1}{2}S(x)^\top\Sigma^{-1}S(x)\right\} \tag{2.6}$$

Pode-se ver que o integrando em (2.2) é o produto de (2.5) e (2.6). Assim, a função de verossimilhança marginal tem forma passível de ser aplicada na aproximação de Laplace, com:

$$\begin{aligned}\varrho\left(S(x)\right) =& y(x)^\top(D\beta + S(x)) - 1^\top b(D\beta + S(x)) + 1^\top c(y(x)) \\ &- \frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}S(x)^\top\Sigma^{-1}S(x)\end{aligned} \tag{2.7}$$

A otimização da função (2.4) requer o valor máximo $\hat{s}$ de (2.7), um problema de otimização numérica de alta dimensionalidade. Bonat e Ribeiro Jr (2016) utilizam o algoritmo de Newton-Raphson para encontrar $\hat{s}$, que consiste no esquema iterativo a seguir:

$$s_{i+1} = s_i + \varrho''(s_i)^{-1}\varrho'(s_i) \tag{2.8}$$

Com $\varrho'(s)$ sendo o gradiente de $\varrho(s)$. Com isso, temos as expressões genéricas para as derivadas (gradiente e hessiano) para o algoritmo de Newton-Raphson, dados por:

$$\begin{aligned}\varrho'(s) &= \left\{y(x) - b'(D\beta + s)\right\}^\top - s^\top\Sigma^{-1} \\ \varrho''(s) &= -\operatorname{diag}\left\{b''(D\beta + s)\right\} - \Sigma^{-1}\end{aligned} \tag{2.9}$$

De modo que a aproximação de Laplace para a log-verossimilhança é dada por:

$$\begin{aligned}l(\theta;y(x)) =& \frac{n}{2}\log(2\pi) - \frac{1}{2}\log\left|\operatorname{diag}\left\{b''(D\beta + \hat{s}(\theta))\right\} + \Sigma^{-1}\right| + y(x)^\top(D\beta + \hat{s}(\theta)) \\ &- 1^\top b(D\beta + \hat{s}(\theta)) + 1^\top c(y(x)) - \frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}\hat{s}(\theta)^\top\Sigma^{-1}\hat{s}(\theta)\end{aligned} \tag{2.10}$$

Para a maximização de (2.10) os autores utilizam o algoritmo BFGS, implementado na função `optim()` do R, com o modelo parametrizado como $\theta = (\beta, \log(\sigma^2), \log(\phi), \log(\tau^2), \log(\psi)$.

Sendo $\hat{\theta}$ o estimador de máxima verossimilhança de θ, o mesmo tem sua distribuição assintótica dada por:

$$\hat{\theta} \sim N\left(\theta, I_O^{-1}(\hat{\theta})\right) \tag{2.11}$$

Com $I_O^{-1}(\hat{\theta})$ denotando a matriz de informação observada de θ. Para que a maximização dos parâmetros seja computacionalmente eficiente, é necessário que os valores iniciais para θ sejam bem especificados. Para isso, os autores sugerem ajustar um modelo linear generalizado (utilizando a função `glm()` do R) para obter os valores iniciais de β. Baseados nesses valores, computa-se $\hat{\mu}$ e os resíduos $\hat{r} = (y - \hat{\mu})$.

A variância amostral de $\hat{r}$ é usada como estimativa inicial para σ^2. Caso o modelo contenha efeito de pepita (τ^2), um percentual de σ^2 é usado como estimativa inicial (usualmente 10%). Para ϕ os autores sugerem usar 10% da maior distância entre dois pontos observados na amostra.

3 Material e Métodos

3.1 Funções para ajuste

Em Bonat e Ribeiro Jr (2016), os autores disponibilizam as funções na linguagem de programação R para ajuste de **SGLMMs** por meio do link <http://leg.ufpr.br/doku.php/publications:papercompanions:sglmm>, no arquivo `functionssglmm.r`, e as utilizam para ajustar dois modelos, um Binomial e outro Poisson a dois conjuntos de dados.

Para os modelos ajustados, é necessário construir a matriz de covariância Σ para os dados, que é feito pela função `monta.sigma` que se utiliza da função `geoR::varcov.spatial` para calcular a matriz de covariância espacial, dados os parâmetros especificados.

A implementação do algoritmo de maximização por Newton-Raphson foi feita por meio da função `newton.raphson` e para sua utilização é necessário conhecer $\varrho(S(x))$, o integrando da verossimilhança marginal, que depende da distribuição dos dados, bem como seu gradiente $\varrho'(S(x))$ e hessiano $\varrho''(S(x))$.

Como o objetivo é estimar para dados não-gaussianos, os mesmos foram implementados por meio das funções `Q.b`, `Q.b.grad` e `Q.b.hess` (respectivamente). As distribuições que foram incluídas são a Binomial, Poisson, Binomial Negativa, Gamma e Beta, (calculadas conforme (2.7)).

Para a avaliação da aproximação de Laplace, foi implementada a função `laplace`, que recebe `Q.b`, `Q.b.grad` e `Q.b.hess` e o otimizador escolhido, podendo ser `"BFGS"`, que usará a função `optim` para otimizar o vetor de parâmetros, ou `"NR"`, para utilizar a função `newton.raphson` para otimizar.

A partir disso, a função `laplace` é incluída na função `loglik.sglmm`, que é responsável pela avaliação da log-verossimilhança do modelo, retornando o negativo da matriz de informação observada.

Como especificado anteriormente, para a aproximação de Laplace ser computacionalmente eficiente, é necessário que os valores iniciais dos parâmetros sejam uma boa estimativa. Para tal, a função `start.values` ajusta um modelo linear generalizado para os dados, considerando a distribuição estipulada (podendo ser Binomial, Poisson, Gamma, Binomial Negativa e Beta) e retorna transformações das estimativas como valores iniciais para $\hat{\theta}$.

A função que estima os parâmetros do modelo é `sglmm`, que utiliza a função `bbmle::mle2` para maximizar a log-verossimilhança dada por `loglik.sglmm`. Ela retorna uma lista, com $\hat{\theta}$, os valores preditos para os efeitos aleatórios de cada ponto amostral e o modelo maximizado, que possibilita explorar propriedades dos estimadores intervalares, como intervalos de confiança e o perfil de verossimilhança de cada parâmetro. O esquema geral do funcionamento das funções é apresentado no diagrama da Figura 1.

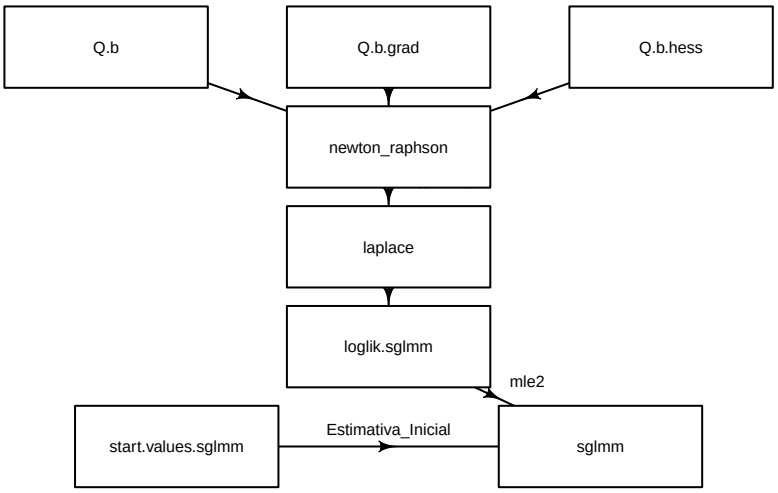

Figura 1 – Diagrama do funcionamento das funções

O código R com todas as funções modificadas está presente no Apêndice A, e o código utilizado para a estimação, criação de tabelas e gráficos, está presente no Apêndice B.

3.2 Bases de dados

Duas bases de dados serão utilizadas para avaliar a performance das funções: `Weed`, disponibilizada no pacote `geoCount` (Jing e De Oliveira (2015)), e `SPT`, disponibilizada pelo aluno de mestrado em Geotecnia pela Universidade Federal do Paraná, Lucas Michael Luzzi, apresentados no artigo **Modelagem Geoestatística de Parâmetros Geotécnicos do Solo de Um Aeroporto**, que será defendido no COBRAMSEG2024.

A base de dados `Weed` consiste na contagem de ervas daninhas em uma plantação da fazenda Bjertorp, no sudoeste da Suécia. Imagens foram capturadas por câmera e um software de detecção de imagens foi utilizado para fazer a estimação da quantidade de plantas, que foi posteriormente comparado às contagens exatas (Guillot, Lorén e Rudemo (2009)). A figura 2, mostra os gráficos dos dados.

Os pontos coloridos no primeiro gráfico indicam o valor dos dados, em que os da cor azul representam dados no primeiro quartil, os verdes no segundo quartil, os amarelos no terceiro quartil e os vermelhos no quarto quartil. É possível verificar que o gráfico de pontos sugere possível dependência espacial pela distribuição das cores, não indicando tendência com as coordenadas (`x1` e `x2`).

O quarto gráfico (o histograma com a densidade empírica) indica uma distribuição fortemente assimétrica e distante da Normal para a resposta . Como os dados são de

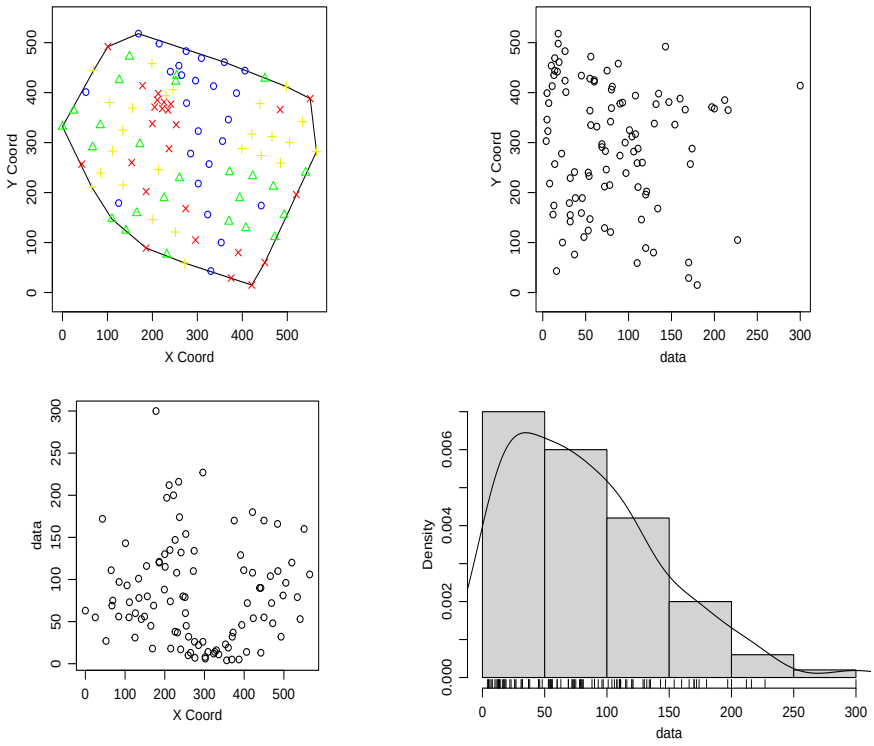

Figura 2 – Gráfico da base de dados Weed.

contagens, a distribuição de Poisson é uma candidata natural para a modelagem, com a distribuição Binomial Negativa também sendo uma possível candidata. Não há covariáveis a serem incluídas na modelagem.

A base de dados `SPT` se refere à ensaios de sondagem por penetração realizados no solo do Aeroporto Internacional Afonso Pena. O estudo tem como objetivo contar a quantidade de marteladas necessárias para afundar o solo 20 centímetros, em diferentes profundidades. Foram feitas 51 perfurações, com o ensaio sendo feito em todas elas em 15 profundidades diferentes, separadas por 1 metro cada. Assim, o objetivo é estimar a densidade do solo em cada profundidade, para se obterem informações geotécnicas que podem ser usadas em construções, fundações, pavimentação, entre outras aplicações.

O gráfico 3 mostra a distribuição da contagem de marteladas necessárias em cada profundidade para afundar o solo em 20 centímetros. Podemos ver que a quantidade necessária cresce, assim como a variância dessa contagem, conforme a profundidade no solo aumenta. Novamente, como os dados são de contagens, a distribuição de Poisson será usada na modelagem. Duas profundidades serão escolhidas para modelar, uma mais superficial e outra mais profunda, para comparar como se comportam as estimações do processo espacial nas duas. A mais superficial será a camada 4 metros e a mais profunda será a camada 13 metros.

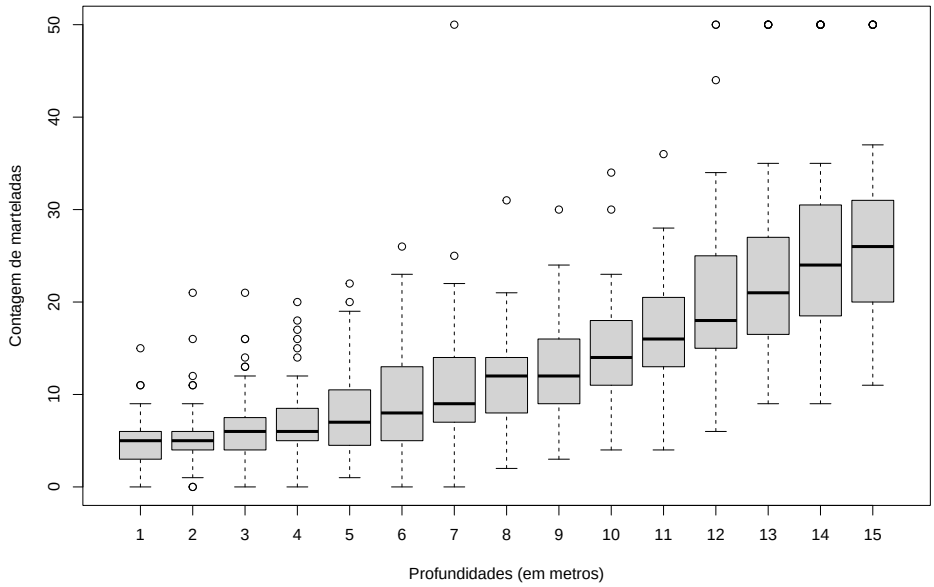

Figura 3 – Boxplot de SPT por profundidade

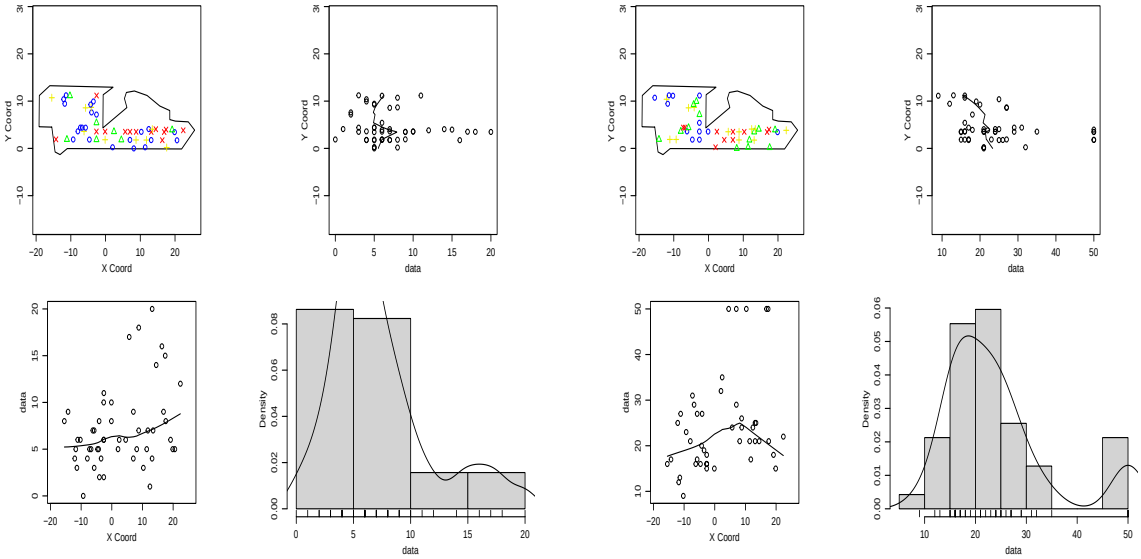

Figura 4 – Gráfico da base de dados SPT, para profundidades 4 e 13 metros

3.3 Recursos computacionais

Para o ajuste dos dados e para a computação será utilizada a linguagem de programação R, versão 4.4.1 (R Core Team (2024)). Foram utilizados os pacotes `geoR` e `geoRglm` para análise de dados espaciais, `MASS` (Venables e Ripley (2002)), `lme4` (Bates et al. (2015)) e `bblme` (Bolker e R Development Core Team (2023)) para a otimização dos modelos por máxima verossimilhança, e `parallel` e `doParallel` (Corporation e Weston (2022)) para execução em paralelo dos modelos.

4 Resultados e Discussão

4.1 Estudo de simulação

Foi realizado um estudo de simulação, para verificar as propriedades para os estimadores obtidos pela função `sglmm`, seguindo o seguinte esquema para simulação: uma *seed* para reprodução é definida, gera-se uma amostra de tamanho n usando a função `geoR::grf`, utilizando os parâmetros $\theta = (\beta_0 = 2, \sigma^2 = 0.5, \phi = 30, \tau^2 = 0.05)$, com função de covariância espacial exponencial, em posições de uma malha irregular de 200×200. As amostras y_i (geradas de um campo aleatório gaussiano não condicional) são usados para simular valores de uma Poisson, com $\lambda_i = y_i$.

Esses valores simulados da Poisson são então usados para gerar um vetor inicial de estimativas usando a função `start.values.sglmm`, que serão usados para ajustar o modelo usando a função `sglmm`. A simulação, ao fim, irá retornar apenas as estimativas pontuais dos modelos ajustados.

4.1.1 Diferentes tamanhos amostrais

Para explorar o comportamento dos estimadores para diferentes tamanhos amostrais, foram geradas 1000 repetições de amostras com $n = 50, 100, 200$ valores. A partir dos valores simulados, podemos ver como se comportam a média e o erro quadrático médio dos estimadores, bem como sua variância e seu viés.

Tabela 1 – Estimadores para diferentes tamanhos de amostras

Parâmetro	n	Média	Variância	EQM	Viés
$\beta_0 = 2$	$n = 50$	2.0043	0.0539	0.0538	-0.0000
	$n = 100$	2.0070	0.0476	0.0476	0.0000
	$n = 200$	1.9920	0.0410	0.0411	0.0000
$\sigma^2 = 0.5$	$n = 50$	0.4461	0.0320	0.0349	0.0029
	$n = 100$	0.4611	0.0241	0.0256	0.0015
	$n = 200$	0.4642	0.0171	0.0183	0.0013
$\phi = 30$	$n = 50$	27.2201	388.7737	396.1129	7.3392
	$n = 100$	29.4890	270.7679	270.7582	-0.0097
	$n = 200$	27.6280	135.1790	140.6701	5.4912
$\tau^2 = 0.05$	$n = 50$	0.0490	0.0058	0.0058	-0.0000
	$n = 100$	0.0491	0.0033	0.0033	-0.0000
	$n = 200$	0.0431	0.0016	0.0017	0.0000

Pela tabela 1 podemos ver que a média dos estimadores se aproximam dos valores reais dos parâmetros que geraram as amostras, e se tornam mais precisos conforme o tamanho da amostra cresce. De maneira similar, podemos ver que tanto o erro quadrático médio quanto a variância dos estimadores diminui conforme o tamanho da amostra cresce.

Além disso, o viés dos estimadores (com exceção de $\hat{\phi}$) são próximos de zero, condizendo com o não-viés assintótico dos estimadores por máxima verossimilhança.

Esses resultados obtidos podem ser comparados com as estimativas geradas utilizando algoritmos MCMC para máxima verossimilhança, por meio das funções do pacote `geoRglm` (Christensen e Ribeiro Jr (2002)), apresentadas na tabela 2.

Tabela 2 – Estimadores MCMC para diferentes tamanhos de amostras

Parâmetro	n	Média	Variância	EQM	Viés
$\beta_0 = 2$	$n = 50$	2.0037	0.0543	0.0542	-0.0000
	$n = 100$	2.0077	0.0479	0.0480	0.0000
	$n = 200$	1.9937	0.0409	0.0409	-0.0000
$\sigma^2 = 0.5$	$n = 50$	0.4547	0.0346	0.0366	0.0020
	$n = 100$	0.4750	0.0237	0.0243	0.0006
	$n = 200$	0.4820	0.0166	0.0169	0.0003
$\phi = 30$	$n = 50$	28.0700	458.9902	462.2560	3.2658
	$n = 100$	28.7707	217.6475	218.9410	1.2935
	$n = 200$	26.0654	103.3541	118.7315	15.3774
$\tau^2 = 0.05$	$n = 50$	0.6508	94.9249	95.1910	0.2661
	$n = 100$	0.0989	0.0207	0.0231	0.0024
	$n = 200$	0.0512	0.0031	0.0031	-0.0000

Não há diferença muito aparente entre os estimadores obtidos para β_0 e σ^2, com ambas as metodologias obtendo estimativas consistentes, com baixa variância e não-viesadas. Para ϕ, ambas estimam com algum grau de viés, mas com a estimativa por aproximação de Laplace diminuindo o viés conforme o tamanho da amostra aumenta, enquanto a estimativa por MCMC aumenta conforme o tamanho de amostra cresce.

Por fim, a estimativa de τ^2 é consistente e não-viesada independentemente do tamanho da amostra utilizando a aproximação de Laplace, enquanto que a estimativa por MCMC é viesada para pequenas amostras, sendo corrigida para amostras maiores. A grande diferença entre as estimativas é o tempo computacional utilizado para ajuste, com a simulação utilizando a função `sglmm` demorando 3, 6.8 e 22.8 minutos para ajustar as 1000 repetições de tamanho $n = 50, 100, 200$ respectivamente, enquanto que a simulação utilizando as funções do pacote `geoRglm` demoraram 12.6, 42.5 e 135.4 minutos para ajustar as 1000 repetições.

4.1.2 Especificação incorreta da função de correlação espacial

Foram comparados os resultados anteriores com os estimadores gerados utilizando as funções Matèrn (com $\kappa = 1, 2$) e Esférica. Em todos os casos, o tamanho da amostra é de $n = 100$, e os parâmetros usados para geração são os especificados anteriormente. Os resultados serão comparados com as estimativas considerando a função de correlação espacial correta (exponencial), já apresentados na tabela1.

Tabela 3 – Estimadores para diferentes funções de covariância

Parâmetro	Correlação	Média	Variância	EQM	Viés
$\beta_0 = 2$	Exponencial	2.0070	0.0476	0.0476	0.0000
	Matèrn, $\kappa = 1$	2.0053	0.0501	0.0501	-0.0000
	Matèrn, $\kappa = 2$	2.0069	0.0472	0.0472	0.0000
	Esférica	2.0073	0.0472	0.0472	0.0000
$\sigma^2 = 0.5$	Exponencial	0.4611	0.0241	0.0256	0.0015
	Matèrn, $\kappa = 1$	0.4678	0.0592	0.0601	0.0010
	Matèrn, $\kappa = 2$	0.4197	0.0223	0.0288	0.0064
	Esférica	0.3920	0.0216	0.0333	0.0116
$\phi = 30$	Exponencial	29.4890	270.7679	270.7582	-0.0097
	Matèrn, $\kappa = 1$	73.5345	1991.4140	3884.6752	1893.2612
	Matèrn, $\kappa = 2$	18.8675	138.1816	261.9758	123.7942
	Esférica	12.3838	51.1202	361.3991	310.2789
$\tau^2 = 0.05$	Exponencial	0.0491	0.0033	0.0033	-0.0000
	Matèrn, $\kappa = 1$	0.0792	0.0042	0.0051	0.0008
	Matèrn, $\kappa = 2$	0.0897	0.0047	0.0062	0.0016
	Esférica	0.1143	0.0054	0.0095	0.0041

Pela tabela 3 podemos ver que a especificação incorreta da função de correlação espacial impacta significativamente as estimativas obtidas para o parâmetro ϕ, com o viés pelas especificações por Matèrn crescendo conforme κ cresce (o que é esperado, visto que a exponencial é um caso particular da Matèrn, com $\kappa = 0.5$) e com a esférica estimando muito acima do verdadeiro valor. Novamente, compararemos com os valores estimados por algoritmos MCMC, presentes na tabela 4.

Tabela 4 – Estimadores MCMC para diferentes funções de covariância

Parâmetro	Correlação	Média	Variância	EQM	Viés
$\beta_0 = 2$	Exponencial	2.0077	0.0479	0.0480	0.0000
	Matèrn, $\kappa = 1$	2.0085	0.0477	0.0477	0.0000
	Matèrn, $\kappa = 2$	2.0420	0.0487	0.0504	0.0017
	Esférica	2.0954	0.0869	0.0959	0.0090
$\sigma^2 = 0.5$	Exponencial	0.4750	0.0237	0.0243	0.0006
	Matèrn, $\kappa = 1$	0.4657	0.0235	0.0246	0.0012
	Matèrn, $\kappa = 2$	0.4586	0.0256	0.0273	0.0017
	Esférica	0.4671	0.0504	0.0514	0.0010
$\phi = 30$	Exponencial	28.7707	217.6475	218.9410	1.2935
	Matèrn, $\kappa = 1$	28.0138	217.9608	221.6878	3.7270
	Matèrn, $\kappa = 2$	40.5729	164.1697	275.7919	111.6222
	Esférica	158.0871	5163.2075	21564.3418	16401.1343
$\tau^2 = 0.05$	Exponencial	0.0989	0.0207	0.0231	0.0024
	Matèrn, $\kappa = 1$	0.1409	0.0714	0.0796	0.0082
	Matèrn, $\kappa = 2$	0.0000	0.0000	0.0025	0.0025
	Esférica	0.0000	0.0000	0.0025	0.0025

Nesse caso, é possível perceber que as estimativas para as funções de correlação

espacial Matèrn foram piores do que a esférica, com viéses muito maiores do que os apresentados na tabela 3 e valores para o alcance ϕ muito acima do valor real (30). Percebe-se também que, apesar de ter sido especificado que havia efeito de pepita (τ^2), as funções para estimação por MCMC estimaram todos os mil modelos com $\tau^2 = 0$.

4.1.3 Diferentes regiões amostrais

Por fim, foi verificado o comportamento dos estimadores quando aumentamos e diminuímos a região da qual a amostra é coletada. Os dados foram simulados em uma malha menor (50×50) e uma maior (500×500), para conferir o comportamento dos estimadores.

Tabela 5 – Estimadores para diferentes regiões amostrais

Parâmetro	Malha	Média	Variância	EQM	Viés
$\beta_0 = 2$	50×50	2.0014	0.1959	0.1957	-0.0002
	200×200	2.0070	0.0476	0.0476	0.0000
	500×500	2.0053	0.0159	0.0159	0.0000
$\sigma^2 = 0.5$	50×50	0.3579	0.0481	0.0682	0.0201
	200×200	0.4611	0.0241	0.0256	0.0015
	500×500	0.4567	0.0217	0.0236	0.0019
$\phi = 30$	50×50	19.5434	261.2456	3336.4146	3075.1690
	200×200	29.4890	270.7679	270.7582	-0.0097
	500×500	32.7543	454.7422	2238.9854	1784.2433
$\tau^2 = 0.05$	50×50	0.0372	0.0013	0.0015	0.0002
	200×200	0.0491	0.0033	0.0033	-0.0000
	500×500	0.0706	0.0103	0.0107	0.0004

Novamente, comparemos com os valores estimados por algoritmos MCMC, presentes na tabela 6.

Tabela 6 – Estimadores MCMC para diferentes regiões amostrais

Parâmetro	Malha	Média	Variância	EQM	Viés
$\beta_0 = 2$	50×50	1.9974	0.1999	0.1997	-0.0002
	200×200	2.0077	0.0479	0.0480	0.0000
	500×500	2.0175	0.0159	0.0162	0.0003
$\sigma^2 = 0.5$	50×50	0.3609	0.0426	0.0620	0.0193
	200×200	0.4750	0.0237	0.0243	0.0006
	500×500	0.4749	0.0170	0.0176	0.0006
$\phi = 30$	50×50	17.0598	198.6525	3555.5247	3356.8722
	200×200	28.7707	217.6475	218.9410	1.2935
	500×500	34.5307	392.4463	2029.8207	1637.3745
$\tau^2 = 0.05$	50×50	0.1335	0.0328	0.0398	0.0069
	200×200	0.0989	0.0207	0.0231	0.0024
	500×500	0.1986	4.6986	4.7159	0.0174

Pode-se perceber com a tabela 5 que o aumento no espaço para a amostra faz com que ocorra um aumento no erro para a estimação de τ^2. Isso era esperado, pois o efeito de pepita é mais fácil de estimar quanto mais densa for a distribuição dos dados no espaço (Isaaks e Srivastava (1989)). Além disso, a estimação de ϕ novamente é afetada, pois para regiões muito pequenas os pontos amostrais estão muito próximos, subestimando o valor de ϕ, enquanto que no caso contrário os pontos podem estar afastados demais, o que faz com que a correlação seja superestimada.

A diferença entre as estimativas por aproximação de Laplace e MCMC está no efeito de pepita τ^2, em que o método por MCMC superestimou o valor em todos os casos, além de ter tido viés maior em quase todas as estimativas. Vale citar, que em todos os casos, o tempo computacional necessário para as simulações foi muito maior para os algoritmos MCMC do que para a função `sglmm`.

4.2 Ajuste à base de dados Weed

O objetivo da modelagem para a base de dados `Weed` é poder mapear a propensão de ervas daninhas na região do estudo por meio das contagens realizadas nos pontos amostrais. Assim, é possível fazer a aplicação de herbicidas de maneira localizada, em regiões com altas contagens, ao invés de aplicar em toda a região, poupando recursos no combate às ervas daninhas.

Como visto anteriormente, a ditribuição da contagem é bastante assimétrica, e por se tratar de uma variável discreta, a modelagem seguindo a distribuição Normal não é a mais correta. Portanto, foram ajustados modelos espaciais lineares mistos generalizados, considerando duas distribuições: a Poisson e a Binomial Negativa.

4.2.1 Ajustes considerando distribuição Poisson

Foram ajustados modelos apenas com o intercepto ($\beta = (\beta_0)$), e diferentes modelos de covariância espacial, inclusão de efeito de pepita (*nugget*), para verificar o que melhor se ajusta aos dados. Para cada um deles, temos as estimativas pontuais dos parâmetros ($\theta = (\beta, \sigma^2, \phi, \tau^2)$), bem como o valor da log-verossimilhança, que será usado para seleção do modelo.

O seguinte código em R mostra o esquema de modelagem para um desses modelos, considerando que há efeito de pepita (`nugget = T`), distribuição Poisson para a resposta (`family = "poisson"`), modelo somente com o intercepto (a fórmula para o modelo é `y 1`, com `y` sendo o nome da coluna com a variável resposta) e função de correlação espacial Matèrn, com $\kappa = 1$:

```r
# Valores iniciais:
theta_ini <- start.values.sglmm(y ~ 1, data = Weed, family = "poisson",
                                coords = Weed[, 1:2], nugget = T)

# Modelo:
ajuste_weed_1 <- sglmm(y ~ 1, data = Weed, coords = Weed[, 1:2],
                       family = "poisson", inits = theta_ini, nugget = T,
                       cov.model = "matern", kappa = 1)
```

A tabela 7 indica os modelos ajustados, as estimativas (pontuais) dos parâmetros e o valor da log-verossimilhança maximizada. Por meio dela, é possível verificar que, nos modelos que incorporam apenas o intercepto, o que teve melhor ajuste é o com função de covariância espacial esférica e efeito de pepita (τ^2).

Tabela 7 – Estimativas dos parâmetros

Modelo	$\hat{\beta}_0$	$\log(\hat{\sigma^2})$	$\log(\hat{\phi})$	$\log(\hat{\tau^2})$	logLik
Exponencial $+ \tau^2$	4.0687	-0.0859	4.2546	-8.4476	-518.6567
Matèrn$(\kappa = 1) + \tau^2$	4.0415	-0.1347	3.6391	-3.5915	-518.1001
Matèrn$(\kappa = 1.5) + \tau^2$	4.0336	-0.1838	3.3540	-2.8767	-518.2936
Matèrn$(\kappa = 2) + \tau^2$	4.0297	-0.2172	3.1676	-2.5945	-518.5377
Esférico $+ \tau^2$	4.1464	0.8799	5.9958	-2.9536	-521.6954
Exponencial	4.0686	-0.0856	4.2548	-	-518.6550
Matèrn$(\kappa = 1)$	4.0375	-0.1082	3.5835	-	-518.2112
Matèrn$(\kappa = 1.5)$	4.0258	-0.1281	3.2410	-	-518.8948
Matèrn$(\kappa = 2)$	4.0204	-0.1421	3.0185	-	-519.6152
Esférico	4.0260	0.1513	5.1439	-	-518.4223

Podemos perceber que não houve muita diferença nos valores da log-verossimilhança entre os modelos com τ^2 e os sem, pois as estimativas pontuais para o mesmo foram muito próximas de zero. Portanto, iremos considerar os modelos sem efeito de pepita, dentre os quais, o com função de covariância espacial Matèrn, com $\kappa = 1$ é o que tem menor log-verossimilhança. Para corroborar essa decisão, podemos aplicar o teste da razão de verossimilhanças, por meio da função `anova` nos modelos com e sem efeito de pepita:

```
## Likelihood Ratio Tests
## Model 1: fit_weed_1[[9]], [loglik.sglmm]: (Intercept)+logsigma2+logphi+
##           logtau2
## Model 2: fit_weed_1_2[[9]], [loglik.sglmm]: (Intercept)+logsigma2+logphi
##   Tot Df Deviance  Chisq Df Pr(>Chisq)
## 1      4   1036.2
## 2      3   1036.4 0.2221  1     0.6374
```

Como o resultado foi não significativo para a inclusão do efeito de pepita, manteremos o modelo mais parcimonioso. Para além das estimativas pontuais, podemos obter intervalos de confiança para os parâmetros por meio do perfilhamento da verossimilhança, utilizando a função `stats::profile`. Com essa abordagem, podemos verificar como cada estimativa se comporta quando fixamos as demais nas estimativas de máxima verossimilhança, podendo assim verificar possíveis assimetrias em algum dos parâmetros, muito comuns naqueles que estimam a variância (como σ^2 e τ^2, no caso de modelos espaciais lineares mistos generalizados). Os perfis para cada parâmetro estimado pelo modelo são apresentados na figura 5.

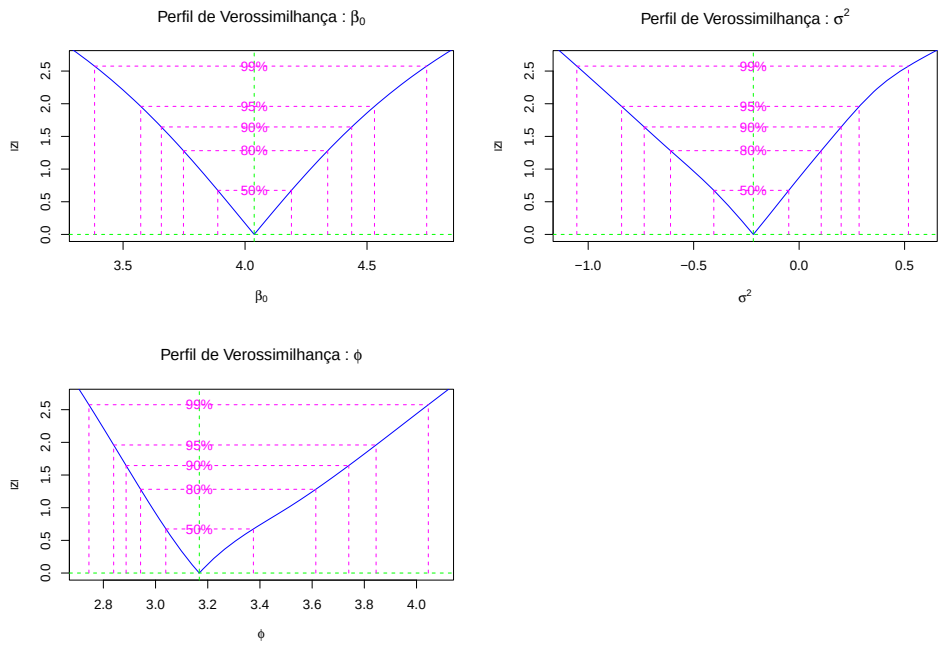

Figura 5 – Perfis de verossimilhança

4.2.2 Ajustes considerando distribuição Binomial Negativa

Podemos comparar esses resultados com os ajustes para modelos que consideram que os dados seguem uma distribuição binomial negativa, que incorpora mais um parâmetro a ser estimado: ψ, que é um parâmetro de precisão necessário para o ajuste do modelo. Os resultados são apresentados na tabela 8.

Dentre os modelos apresentados, o que tem melhor ajuste o com função de correlação esférica e com efeito de pepita (τ^2), ainda que pequeno. Assim como no caso anterior, podemos realizar o teste da razão de verossimilhanças para avaliar a inclusão desse parâmetro no modelo:

```
## Likelihood Ratio Tests
## Model 1: fit_weed_esf_nb[[9]], [loglik.sglmm]: (Intercept)+logsigma2+logphi+
```

Tabela 8 – Estimativas dos parâmetros

Modelo	$\hat{\beta}_0$	$\hat{\sigma^2}$	$\hat{\phi}$	$\hat{\tau^2}$	$\hat{\psi}$	logLik
Exponencial + τ^2	4.0687	0.9178	70.4360	-	17300.5860	-518.6555
Matèrn($\kappa = 1$) + τ^2	4.0462	0.8853	37.1159	0.0014	78.4743	-518.1562
Matèrn($\kappa = 1.5$) + τ^2	4.0636	0.8241	28.9260	0.0015	17.3070	-518.3908
Matèrn($\kappa = 2$) + τ^2	4.1334	0.6844	30.1216	1e-04	5.6871	-518.4510
Esférico + τ^2	4.0833	0.7939	171.3226	4e-04	8.9417	-518.0799
Exponencial	4.0693	0.9174	70.5114	-	1617.2677	-518.6602
Matèrn($\kappa = 1$)	4.0475	0.8846	37.1822	-	66.8343	-518.1607
Matèrn($\kappa = 1.5$)	4.0669	0.8221	28.9158	-	16.8341	-518.3893
Matèrn($\kappa = 2$)	4.1345	0.6844	30.1157	-	5.6858	-518.4501
Esférico	4.0822	0.7940	171.3547	-	8.9200	-518.0782

```
##           logtau2+logprec
## Model 2: fit_weed_esf_2_nb[[9]], [loglik.sglmm]: (Intercept)+logsigma2+
##           logphi+logprec
##   Tot Df Deviance  Chisq Df Pr(>Chisq)
## 1      5   1036.2
## 2      4   1036.2 0.0035  1     0.9529
```

Novamente, a inclusão do efeito de pepita não é estatisticamente significativo, então permanecemos com o modelo mais parcimonioso. Podemos perfilhar a verossimilhança para encontrar as estimativas intervalares, como visto na figura 6.

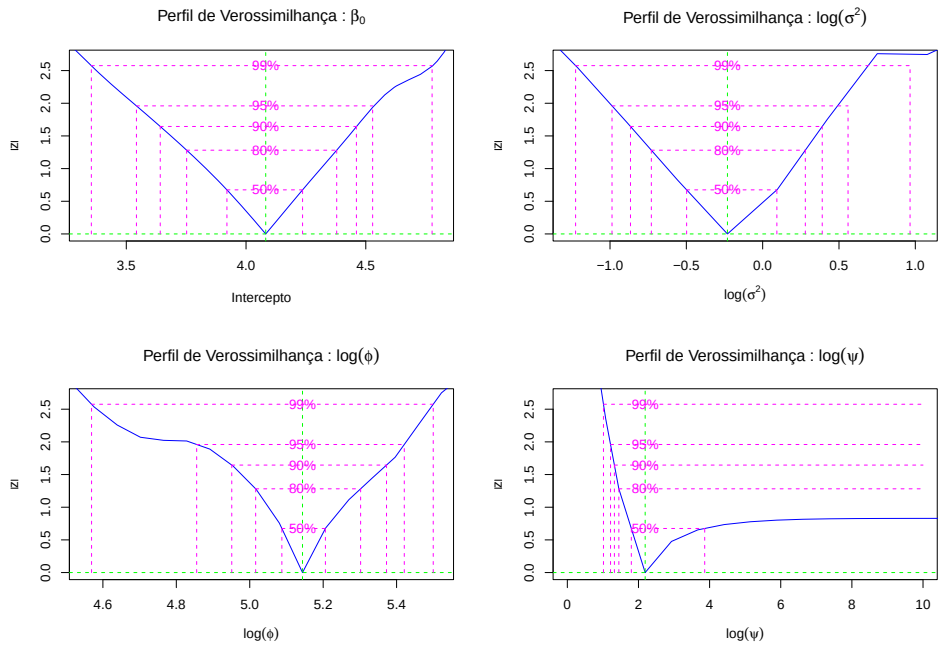

Figura 6 – Perfis de verossimilhança

Com a estimativa intervalar dos parâmetros, podemos ver que há uma assimetria bem pronunciada para a variável de precisão ψ, além de que nesse caso temos um valor

estimado para o alcance ϕ maior do que para o modelo Poisson.

4.2.3 Predição espacial

Podemos realizar predição espacial para os pontos não obsevados por meio de krigagem, utilizando as estimativas dos parâmetros retornados pelo modelo. Ficamos com o modelo considerando a distribuição de Poisson, pois o mesmo é mais simples (possui um parâmetro a menos) e a diferença na log-verossimilhança não foi significativa. Para realizar a predição nos demais pontos amostrais, serão utilizadas as funções `geoR::pred_grid` para criar o *grid* de predição, `geoR::krige.control` e `geoR::output.control` para controlar a krigagem e a saída das funções e `geoR::krige.conv` para realizar a krigagem convencional.

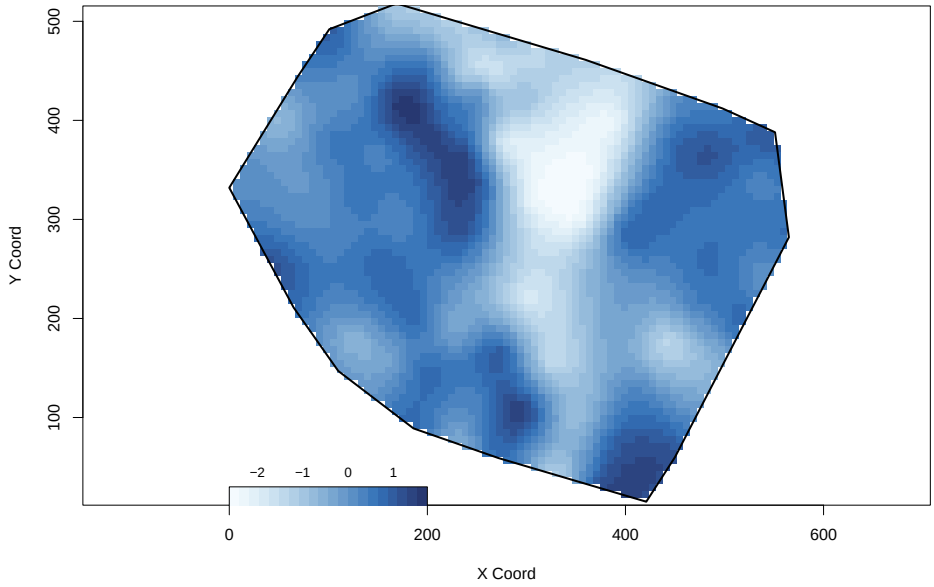

Figura 7 – Predição espacial para dados Weed

É possível verificar que a superfície (contínua) se assemelha aos valores coletados na amostra, indicando que os pontos mais escuros possuem uma contagem de ervas daninhas mais alta do que nos pontos mais claros. Assim, é possível identificar regiões problemáticas para aplicação de herbicidas de maneira localizada.

4.3 Ajuste à base de dados SPT

Foram ajustados modelos que levam em consideração apenas o intercepto ($\beta = (\beta_0)$), e diferentes modelos de covariância espacial, com e sem a inclusão de efeito de pepita (*nugget*), para verificar o que melhor se ajusta aos dados. Para cada um deles, temos as estimativas pontuais dos parâmetros ($\theta = (\beta, \sigma^2, \phi, \tau^2)$), bem como o valor da log-verossimilhança, que será usado para seleção do modelo.

Tabela 9 – Estimativas para profundidade 4 metros

Modelo	$\hat{\beta}_0$	$\hat{\sigma^2}$	$\hat{\phi}$	$\hat{\tau^2}$	logLik
Exponencial + τ^2	1.8724	0.0339	7.1546	0.1521	-139.4723
Matèrn($\kappa = 1$) + τ^2	1.8731	0.0290	5.8920	0.1571	-139.4389
Matèrn($\kappa = 1.5$) + τ^2	1.8736	0.0279	5.1650	0.1584	-139.4199
Matèrn($\kappa = 2$) + τ^2	1.8740	0.0276	4.6447	0.1588	-139.4065
Esférico + τ^2	1.8677	0.0568	5.6793	0.1319	-139.5590
Exponencial	1.8655	0.1906	0.2419	0.0000	-139.9930
Matèrn($\kappa = 1$)	1.8654	0.1906	0.1789	0.0000	-139.9898
Matèrn($\kappa = 1.5$)	1.8653	0.1906	0.1543	0.0000	-139.9872
Matèrn($\kappa = 2$)	1.8656	0.1906	0.0384	0.0000	-140.0011
Esférico	1.8639	0.1909	0.9757	0.0000	-139.9393

Tabela 10 – Estimativas para profundidade 13 metros

Modelo	$\hat{\beta}_0$	$\hat{\sigma^2}$	$\hat{\phi}$	$\hat{\tau^2}$	logLik
Exponencial + τ^2	3.0828	0.1195	2.0688	0.0001	-165.1408
Matèrn($\kappa = 1$) + τ^2	3.0869	0.1207	1.2324	0.0000	-164.8336
Matèrn($\kappa = 1.5$) + τ^2	3.0903	0.1208	0.9287	0.0001	-164.7785
Matèrn($\kappa = 2$) + τ^2	3.0925	0.1207	0.7664	0.0002	-164.7789
Esférico + τ^2	3.0883	0.1248	4.9898	0.0001	-165.2246
Exponencial	3.0828	0.1196	2.0676	0.0000	-165.1396
Matèrn($\kappa = 1$)	3.0868	0.1207	1.2323	0.0000	-164.8328
Matèrn($\kappa = 1.5$)	3.0902	0.1209	0.9280	0.0000	-164.7751
Matèrn($\kappa = 2$)	3.0925	0.1209	0.7659	0.0000	-164.7747
Esférico	3.0884	0.1249	4.9893	0.0000	-165.2236

As tabelas 9 e 10 apresentam os modelos ajustados, as estimativas pontuais e o valor da log-verossimilhança retornado pelo modelo. Avaliando a log-verossimilhança para os modelos ajustados para a profundidade 4 metros, podemos ver que todos retornaram valores muito semelhantes, enquanto que os modelos para a profundidade de 13 metros possuem log-verossimilhanças bem diferentes. Por isso, ficaremos com o modelo com função de correlação espacial Matèrn, com $\kappa = 2$ e efeito de pepita para a profundidade 4 metros e com o modelo com função de correlação espacial Matèrn, com $\kappa = 2$ e sem efeito de pepita para a profundidade 13 metros.

Há uma diferença notável nos valores estimados para os interceptos, que indica que y aumenta conforme a profundidade aumenta. Além disso, há também diferença nos valores de $\hat{\phi}$, em que o modelo para a profundidade 4 metros é 4.6446511, enquanto que para a profundidade 13 metros é 0.7659458. Ou seja, os pontos nas profundidades mais superficiais são mais correlacionados entre si do que os pontos mais profundos, um indicativo que o solo profundo é muito diverso, podendo ser originário de diferentes épocas que sofreram compactação.

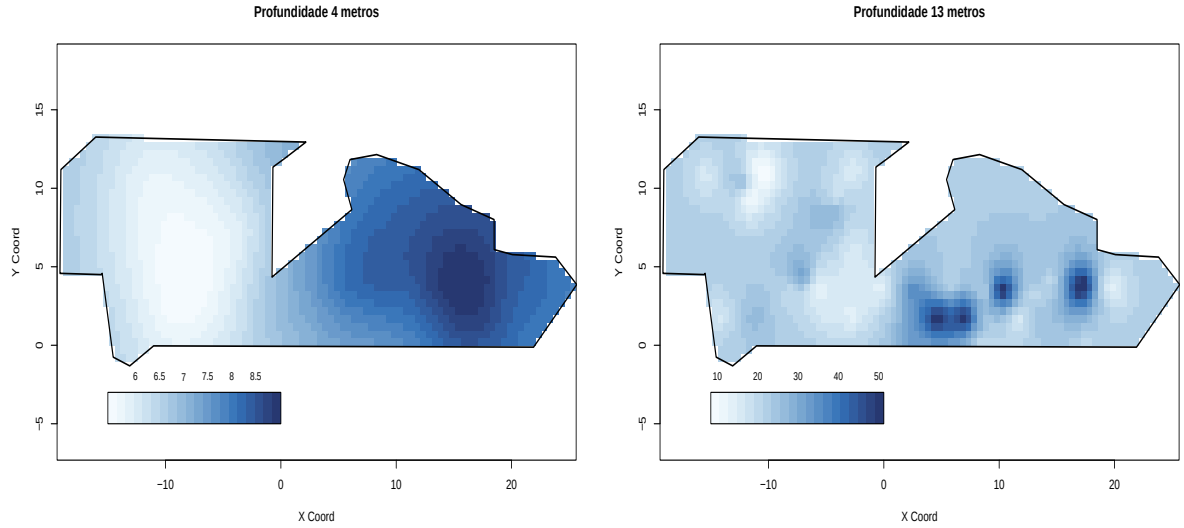

Figura 8 – Predição espacial para as duas profundidades

4.3.1 Predição espacial

Novamente, performaremos a predição espacial utilizando krigagem, para comparar as duas profundidades. Foi construído um grid para predição, e a interpolação foi realizada por meio das funções do pacote `geoR`, como mencionadas anteriormente.

Percebemos que a predição para essas duas profundidades diferem muito na maneira como os valores preditos se comportam. Como o valor do alcance estimado para a profundidade 4 é muito grande (em relação à escala dos dados), temos que a correlação entre os pontos é muito grande, o que indica a quase uniformidade de valores estimados pela krigagem.

Já para a profundidade 13 metros podemos perceber a alta variabilidade no processo que gera os dados, que possui uma região com valores tipicamente maiores (à direita) e outros com valores menores (à esquerda). Com essa modelagem, é possível determinar que a região à direita possui valores maiores para a contagem na sondagem SPT, indicando um solo mais compactado e mais duro, informação essa que pode ajudar no planejamento geotécnico das construções do Aeroporto.

5 Considerações Finais

Como visto, os estimadores obtidos por meio da aproximação de Laplace possuem propriedades estatísticas ótimas, como o não-viés assintótico e ser erro quadrático médio consistente, além de vantagens computacionais sobre os ajustes baseados no paradigma bayesiano, visto que não é necessário avaliar a convergência do método. Ainda que seja computacionalmente intensivo, o ajuste é obtido por uma maximização de alta dimensionalidade, que pode ser resolvida por diversas heurísticas além das apresentadas neste trabalho.

A comparação com os métodos baseados em algoritmos MCMC, por meio dos estudos de simulação, mostrou que os valores estimados são muito semelhantes, mas com a vantagem para os métodos baseados na aproximação de Laplace por terem um tempo computacional muito menor para estimação, não-viés assintótico, e simplicidade no uso, com uma sintaxe condizendo com outras funções para ajuste de modelos na linguagem de programação R, como `lm` e `glm`.

O ajuste às bases de dados reais foi satisfatória, sendo possível comparar os diferentes modelos ajustados de maneira direta, por meio do valor retornado pela log-verossimilhança, e pelo teste de razão de verossimilhança para considerar a inclusão de variáveis no modelo. A exploração dos perfis de verossimilhança se provaram úteis, visto que permitem visualizar o comportamento do estimador na sua vizinhança, indicando possíveis assimetrias e tendências assintóticas.

O desenvolvimento de mais recursos para a função, como inclusão de transformações para os dados (por meio do método de Box-Cox), avaliação do modelo considerando anisotropia presente nos dados, inclusão de métodos para visualizar graficamente e sumarizar o modelo , incluir mais distribuições da família exponencial de distribuições, permitir alterar a função de ligação $g(\cdot)$ do modelo (que atualmente está considerando a ligação canônica), dentre outros, será avaliada, bem como a implementação das funções como parte do pacote para a linguagem R `geoR`.

Referências

BANERJEE, S.; CARLIN, B. P.; GELFAND, A. E. *Hierarchical modeling and analysis for spatial data*. Boca Raton, Fla: Chapman & Hall/CRC, 2004. (Monographs on statistics and applied probability, 101). ISBN 9781584884101.

BATES, D. et al. Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, v. 67, n. 1, p. 1–48, 2015.

BOLKER, B.; R Development Core Team. *bbmle: Tools for General Maximum Likelihood Estimation*. [S.l.], 2023. R package version 1.0.25.1. Disponível em: <https://CRAN.R-project.org/package=bbmle>.

BONAT, W. H.; Ribeiro Jr, P. J. Practical likelihood analysis for spatial generalized linear mixed models. *Environmetrics*, v. 27, n. 2, p. 83–89, mar. 2016. ISSN 1180-4009, 1099-095X.

BRESLOW, N. E.; CLAYTON, D. G. Approximate Inference in Generalized Linear Mixed Models. *Journal of the American Statistical Association*, v. 88, n. 421, p. 9–25, mar. 1993. ISSN 0162-1459, 1537-274X. Disponível em: <https://www.tandfonline.com/doi/full/10.1080/01621459.1993.10594284>.

CASELLA, G.; BERGER, R. L. *Inferência estatística*. São Paulo: Cengage Learning, 2011.

CHRISTENSEN, O.; Ribeiro Jr, P. georglm - a package for generalised linear spatial models. *R-NEWS*, v. 2, n. 2, p. 26–28, 2002. ISSN 1609-3631.

CHRISTENSEN, O. F. Monte Carlo Maximum Likelihood in Model-Based Geostatistics. *Journal of Computational and Graphical Statistics*, v. 13, n. 3, p. 702–718, set. 2004. ISSN 1061-8600, 1537-2715.

CORPORATION, M.; WESTON, S. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*. [S.l.], 2022. R package version 1.0.17. Disponível em: <https://CRAN.R-project.org/package=doParallel>.

CRESSIE, N. A. C. *Statistics for Spatial Data*. 1. ed. [S.l.]: Wiley, 1993. (Wiley Series in Probability and Statistics). ISBN 9780471002550 9781119115151.

DIGGLE, P. J.; Ribeiro Jr., P. J. *Model-based Geostatistics*. Guildford Boulder: Springer London NetLibrary, Inc. [distributor], 2007. (Springer series in statistics). ISBN 9780387485362.

GEYER, C. J. On the Convergence of Monte Carlo Maximum Likelihood Calculations. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, v. 56, n. 1, p. 261–274, jan. 1994. ISSN 1369-7412, 1467-9868.

GEYER, C. J.; THOMPSON, E. A. Constrained Monte Carlo Maximum Likelihood for Dependent Data. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, v. 54, n. 3, p. 657–683, jul. 1992. ISSN 1369-7412, 1467-9868.

GUILLOT, G.; LORéN, N.; RUDEMO, M. Spatial Prediction of Weed Intensities From Exact Count Data and Image-Based Estimates. *Journal of the Royal Statistical Society Series C: Applied Statistics*, v. 58, n. 4, p. 525–542, set. 2009. ISSN 0035-9254, 1467-9876. Disponível em: <https://academic.oup.com/jrsssc/article/58/4/525/7113480>.

ISAAKS, E. H.; SRIVASTAVA, R. M. *Applied geostatistics*. New York: Oxford University Press, 1989. ISBN 9780195050127 9780195050134.

JING, L.; De Oliveira, V. geoCount: An R package for the analysis of geostatistical count data. *Journal of Statistical Software*, v. 63, n. 11, p. 1–33, 2015. Disponível em: <http://www.jstatsoft.org/v63/i11/>.

LEE, Y.; NELDER, J. A. Hierarchical Generalized Linear Models. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, v. 58, n. 4, p. 619–656, nov. 1996. ISSN 1369-7412, 1467-9868.

MCCULLOCH, C. E. Maximum Likelihood Algorithms for Generalized Linear Mixed Models. *Journal of the American Statistical Association*, v. 92, n. 437, p. 162–170, mar. 1997. ISSN 0162-1459, 1537-274X.

PINHEIRO, J. C.; BATES, D. M. Approximations to the log-likelihood function in the nonlinear mixed-effects model. *Journal of Computational and Graphical Statistics*, [American Statistical Association, Taylor & Francis, Ltd., Institute of Mathematical Statistics, Interface Foundation of America], v. 4, n. 1, p. 12–35, 1995. ISSN 10618600.

R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2024. Disponível em: <https://www.R-project.org/>.

TIERNEY, L.; KADANE, J. B. Accurate Approximations for Posterior Moments and Marginal Densities. *Journal of the American Statistical Association*, v. 81, n. 393, p. 82–86, mar. 1986. ISSN 0162-1459, 1537-274X.

VENABLES, W. N.; RIPLEY, B. D. *Modern Applied Statistics with S*. Fourth. New York: Springer, 2002. ISBN 0-387-95457-0. Disponível em: <https://www.stats.ox.ac.uk/pub/MASS4/>.

ZHANG, H. On Estimation and Prediction for Spatial Generalized Linear Mixed Models. *Biometrics*, v. 58, n. 1, p. 129–136, mar. 2002. ISSN 0006-341X, 1541-0420.

Apêndices

APÊNDICE A - Funções para ajuste dos modelos

```
# Pacotes utilizados:
library(geoR)
library(geoRglm)
library(MASS)
library(lme4)
library(bbmle)

# Funções:

# Função para montar a matriz de correlação espacial:
monta.sigma <- function(cov.pars, cov.model, nugget = 0, kappa, mat.dist){
    Sigma <- varcov.spatial(dists.lowertri = mat.dist,
                            cov.model = cov.model,kappa = kappa,
                            nugget = nugget, cov.pars = cov.pars)
    return(Sigma)
}

# Função para avaliação da Normal Multivariada:
gauss.mult <- function(b, det.Sigma, inv.Sigma){
    n <- length(b)
    dens <- (-n/2) * log(2 * pi) - 0.5 * det.Sigma -
        0.5 * t(b) %*% inv.Sigma %*% b
    return(dens)
}

# Algoritmo de Newton-Raphson:
newton.raphson <- function(initial, escore, hessiano, tol=0.0001,
                           max.iter, n.dim, ...) {
  solucao <- matrix(NA, max.iter, n.dim)
  solucao[1,] <- initial
  for (i in 2:max.iter) {
    HSS <- hessiano(initial, ...)
    ESC <- t(escore(initial, ...))
    solucao[i,] <- initial - solve(HSS, ESC)
    initial <- solucao[i,]
    tolera <- abs(solucao[i,] - solucao[i-1,])
    if (all(tolera < tol)) break
  }
```

```
  saida <- list(HSS = HSS, solution = initial)
  return(saida)
}

# Função para o integrando:
Q.b <- function(b, Xbeta, Y, det.Sigma, inv.Sigma, family,
                prec = NULL, ntrial = NULL){
  eta <- Xbeta + b
  dens <- switch (
      family,
      "poisson" = sum(dpois(Y, lambda = exp(eta),log=TRUE)) +
          gauss.mult(b, det.Sigma = det.Sigma,
                     inv.Sigma = inv.Sigma),
      "binomial" = sum(dbinom(Y, size = ntrial,
                              prob = (1/(1 + exp(- eta))), log = TRUE)) +
          gauss.mult(b, det.Sigma = det.Sigma,
                     inv.Sigma = inv.Sigma),
      "negative.binomial" = sum(dnbinom(Y, size = prec, mu = exp(eta),
                                        log = TRUE)) +
          gauss.mult(b,det.Sigma = det.Sigma,
                     inv.Sigma = inv.Sigma),
      "gamma" = sum(dgamma(Y, shape = prec, scale = exp(eta)/prec,
                           log=TRUE)) +
          gauss.mult(b,det.Sigma = det.Sigma,
                     inv.Sigma = inv.Sigma),
      "beta" = sum(dbeta(Y, shape1 = (1/(1 + exp(- eta))) * prec,
                         shape2 = (1 - (1/(1 + exp(- eta)))) * prec,
                         log = TRUE)) +
          gauss.mult(b, det.Sigma = det.Sigma,
                     inv.Sigma = inv.Sigma),
      "geometric" = sum(dnbinom(Y, size = 1, prob = (1/(1 + exp(eta))),
                                log = TRUE)) +
          gauss.mult(b, det.Sigma = det.Sigma,
                     inv.Sigma = inv.Sigma)
  )
  return(as.numeric(dens))
}

# Função para o gradiente:
Q.b.grad <- function(b, Xbeta, Y, det.Sigma, inv.Sigma, family,
                     prec = NULL, ntrial = NULL){
```

```
    grad <- switch(
        family,
        "poisson" = {
            t((Y - exp(Xbeta + b))) - t(b) %*% inv.Sigma
        },
        "binomial" = {
            b1 <- ntrial * (1/(1 + exp(-(Xbeta + b))))
            t(Y - b1) - t(b) %*% inv.Sigma
        },
        "negative.binomial" = {
            et1 <- exp(Xbeta)
            et2 <- exp(b)
            et12 <- et1 * et2
            t((Y - (prec + Y) * (et12 * (et12 + prec)^-1))) -
                t(b) %*% inv.Sigma
        },
        "gamma" = {
            p1 <- -prec + prec * (exp(-Xbeta - b)) * Y
            t(p1) - t(b) %*% inv.Sigma
        },
        "beta" = {
            eta <- Xbeta + b
            mu <- 1/(1 + exp(- eta))
            D.mu.b <- exp(-eta) / ((1 + exp(-eta))^2)
            dg1 <- digamma((1 - mu) * prec)
            dg2 <- digamma(mu * prec)
            logY <- log(Y / (1 - Y))
            part1 <- D.mu.b * prec * (dg1 - dg2 + logY)
            t(part1) - t(b) %*% inv.Sigma
        },
        "geometric" = {
            eta <- exp(Xbeta + b)
            t((Y - (1 + Y) * (eta * (eta + 1)^-1))) -
                t(b) %*% inv.Sigma
        }
    )
}

# Função para o hessiano:
Q.b.hess <- function(b, Xbeta, Y, det.Sigma, inv.Sigma,
                     family, prec = NULL, ntrial = NULL) {
```

```
    Hess <- switch(
        family,
        "poisson" = {
            eta <- exp(Xbeta + b)
            diag(inv.Sigma) <- eta + diag(inv.Sigma)
            -inv.Sigma
        },
        "binomial" = {
            b1 <- 1/(1 + exp(-(Xbeta + b)))
            b2 <- exp(2 * Xbeta + 2 * b) / ((1 + exp(Xbeta + b))^2)
            D <- b1 - b2
            diag(inv.Sigma) <- ntrial * D + diag(inv.Sigma)
            -inv.Sigma
        },
        "negative.binomial" = {
            et1 <- exp(Xbeta)
            et2 <- exp(b)
            et12 <- et1 * et2
            p1 <- et12 * ((et12 + prec)^-1)
            p2 <- p1^2
            D <- p1 - p2
            diag(inv.Sigma) <- (prec + Y) * D + diag(inv.Sigma)
            -inv.Sigma
        },
        "gamma" = {
            p2 <- prec * Y * exp(-Xbeta - b)
            diag(inv.Sigma) <- p2 + diag(inv.Sigma)
            -inv.Sigma
        },
        "beta" = {
            eta <- Xbeta + b
            mu <- 1/(1 + exp(-eta))
            p1 <- mu * prec
            p2 <- (1 - mu) * prec
            med <- mu * (1 - mu)
            logY <- log(Y / (1 - Y))
            d2 <- (1 - mu)^2 - mu^2
            part1 <- -prec^2 * (trigamma(p1) + trigamma(p2)) * med
            part2 <- prec * (digamma(p2) - digamma(p1) + logY) * d2
            part3 <- (part1 + part2) * med
            diag(inv.Sigma) <- -part3 + diag(inv.Sigma)
```

```
            -inv.Sigma
        },
        "geometric" = {
            et12 <- exp(Xbeta + b)
            p1 <- et12 * ((et12 + 1)^-1)
            p2 <- p1^2
            D <- p1 - p2
            diag(inv.Sigma) <- (1 + Y) * D + diag(inv.Sigma)
            -inv.Sigma
        }
    )
    return(Hess)
}

# Algoritmo para aproximação de Laplace:
laplace <- function(Q.b, gr, hess, otimizador, n.dim,
                    method.integrate, ...) {
    log.integral <- -sqrt(.Machine$double.xmax)
    inicial <- rep(0, n.dim)
    pred <- NULL
    if (method.integrate == "BFGS") {
        temp <- try(optim(inicial, Q.b, gr = gr, ..., method = otimizador,
                          hessian = TRUE,
                          control = list(fnscale = -1)), silent = TRUE)
    } else if (method.integrate == "NR") {
        temp <- try(newton.raphson(initial = inicial, escore = gr,
                                   hessiano = hess,
                                   n.dim = n.dim, max.iter = 100, ...),
                    silent = TRUE)
    } else if (method.integrate == "QNR") {
        temp <- try(qq.newton.raphson(initial = inicial, escore = gr,
                                      hessiano = hess,
                                      n.dim = n.dim, max.iter = 100, ...),
                    silent = TRUE)
    }
    if (class(temp) != "try-error" && method.integrate == "BFGS") {
        log.integral <- temp$value + ((n.dim / 2) * log(2 * pi) -
                                      0.5 * determinant(-temp$hessian)$modulus)
        pred <- temp$par
    } else if (class(temp) != "try-error" && method.integrate == "NR") {
        value <- Q.b(b = temp$solution, ...)
```

```
        log.integral <- value + ((n.dim / 2) * log(2 * pi) -
                                 0.5 * determinant(-temp[1][[1]])$modulus)
        pred <- temp$solution
    }
    return(list(log.integral = log.integral, pred = pred))
}

# Função para avaliação da log-verossimilhança:
loglik.sglmm <- function(par, Y, X, kappa, nugget, mat.dist, cov.model,
                         family, method.integrate = "NR",
                         ntrial = 1, offset = NA){
    I = -sqrt(.Machine$double.xmax)
    n <- length(Y)
    n.beta <- dim(X)[2]
    beta <- as.numeric(par[1:n.beta])
    Xbeta <- X %*% beta
    if(is.na(offset)[1] != TRUE){
        Xbeta <- cbind(X, log(offset)) %*% c(beta, 1)
    }
    sigma <- exp(as.numeric(par[c(n.beta + 1)]))
    phi <- exp(as.numeric(par[c(n.beta + 2)]))
    if(nugget == TRUE){
        tau2 <- exp(as.numeric(par[c(n.beta+3)]))
    }
    if(nugget == FALSE){
        tau2 <- 0
    }
    if(family == "negative.binomial"   & nugget == TRUE){
        prec <- exp(as.numeric(par[c(n.beta+4)]))
    }
    if(family == "gamma" & nugget == TRUE){
        prec <- exp(as.numeric(par[c(n.beta+4)]))
    }
    if(family == "beta" & nugget == TRUE){
        prec <- exp(as.numeric(par[c(n.beta+4)]))
    }
    if(family == "negative.binomial" & nugget == FALSE){
        prec <- exp(as.numeric(par[c(n.beta+3)]))
    }
    if(family == "gamma" & nugget == FALSE){
        prec <- exp(as.numeric(par[c(n.beta+3)]))
```

```
    }
    if(family == "beta" & nugget == FALSE){
        prec <- exp(as.numeric(par[c(n.beta+3)]))
    }
    if (!is.null(kappa)) {
        kappa = as.numeric(kappa)
    }
    Sigma <- as.matrix(forceSymmetric(
        monta.sigma(cov.pars = c(sigma,phi),
                    cov.model = cov.model, nugget = tau2,
                    kappa = kappa, mat.dist = mat.dist)$varcov)
        )
    chol.Sigma <- try(chol(Sigma),silent=TRUE)
    det.Sigma <- try(sum(log(diag(chol.Sigma)))*2,silent=TRUE)
    inv.Sigma <- try(chol2inv(chol.Sigma), silent=TRUE)
    if(class(chol.Sigma)[1] != "try-error"){
        if(class(inv.Sigma)[1] != "try-error"){
            if (any(family == c("poisson", "binomial", "geometric"))) {
                I <- laplace(Q.b, gr = Q.b.grad, hess = Q.b.hess,
                             method.integrate = method.integrate,
                             otimizador="BFGS", n.dim = n, Xbeta = Xbeta,
                             Y = Y, det.Sigma = det.Sigma,
                             inv.Sigma = inv.Sigma, family = family)
            }
        }
    }
    if(class(chol.Sigma)[1] != "try-error"){
        if(class(inv.Sigma)[1] != "try-error"){
            if (any(family == c("negative.binomial", "gamma", "beta"))) {
                I <- laplace(Q.b, gr = Q.b.grad, hess = Q.b.hess,
                             method.integrate = method.integrate,
                             otimizador="BFGS", n.dim = n, Xbeta = Xbeta,
                             Y = Y, det.Sigma = det.Sigma,, prec = prec,
                             inv.Sigma = inv.Sigma, family = family)
            }
        }
    }
    return(-I[[1]])
}

# Função para calcular os valores preditos para os efeitos aleatórios:
```

```
preditos <- function(par, Y, X, kappa, nugget, mat.dist, cov.model,
                     family, method.integrate = "NR",
                     ntrial = 1, offset = NA){
    I = -sqrt(.Machine$double.xmax)
    n <- length(Y)
    n.beta <- dim(X)[2]
    beta <- as.numeric(par[1:n.beta])
    Xbeta <- X %*% beta
    if(is.na(offset)[1] != TRUE){
        Xbeta <- cbind(X, log(offset)) %*% c(beta, 1)
    }
    sigma <- exp(as.numeric(par[c(n.beta + 1)]))
    phi <- exp(as.numeric(par[c(n.beta + 2)]))
    if(nugget == TRUE){
        tau2 <- exp(as.numeric(par[c(n.beta+3)]))
    }
    if(nugget == FALSE){
        tau2 <- 0
    }
    if(family == "negative.binomial"   & nugget == TRUE){
        prec <- exp(as.numeric(par[c(n.beta+4)]))
    }
    if(family == "gamma" & nugget == TRUE){
        prec <- exp(as.numeric(par[c(n.beta+4)]))
    }
    if(family == "beta" & nugget == TRUE){
        prec <- exp(as.numeric(par[c(n.beta+4)]))
    }
    if(family == "negative.binomial" & nugget == FALSE){
        prec <- exp(as.numeric(par[c(n.beta+3)]))
    }
    if(family == "gamma" & nugget == FALSE){
        prec <- exp(as.numeric(par[c(n.beta+3)]))
    }
    if(family == "beta" & nugget == FALSE){
        prec <- exp(as.numeric(par[c(n.beta+3)]))
    }
    if (!is.null(kappa)) {
        kappa = as.numeric(kappa)
    }
    Sigma <- as.matrix(forceSymmetric(
```

```
        monta.sigma(cov.pars = c(sigma,phi),
                    cov.model = cov.model, nugget = tau2,
                    kappa = kappa, mat.dist = mat.dist)$varcov)
        )
    chol.Sigma <- try(chol(Sigma),silent=TRUE)
    det.Sigma <- try(sum(log(diag(chol.Sigma)))*2,silent=TRUE)
    inv.Sigma <- try(chol2inv(chol.Sigma), silent=TRUE)
    if(class(chol.Sigma)[1] != "try-error"){
        if(class(inv.Sigma)[1] != "try-error"){
            if (any(family == c("poisson", "binomial", "geometric"))) {
                I <- laplace(Q.b, gr = Q.b.grad, hess = Q.b.hess,
                             method.integrate = method.integrate,
                             otimizador="BFGS", n.dim = n, Xbeta = Xbeta,
                             Y = Y, det.Sigma = det.Sigma,
                             inv.Sigma = inv.Sigma, family = family)
            }
        }
    }
    if(class(chol.Sigma)[1] != "try-error"){
        if(class(inv.Sigma)[1] != "try-error"){
            if (any(family == c("negative.binomial", "gamma", "beta"))) {
                I <- laplace(Q.b, gr = Q.b.grad, hess = Q.b.hess,
                             method.integrate = method.integrate,
                             otimizador="BFGS", n.dim = n, Xbeta = Xbeta,
                             Y = Y, det.Sigma = det.Sigma,, prec = prec,
                             inv.Sigma = inv.Sigma, family = family)
            }
        }
    }
    return(I$pred)
}

# Função para estimar os valores iniciais para o modelo:
start.values.sglmm <- function(formula, data, coords, nugget,
                               family, ntrial = 1, offset = NULL){
    if (is.null(offset)) {
        offset <- rep(1, nrow(data))
    }
    mf <- model.frame(formula,data)
    Y <- model.response(mf)
    X <- model.matrix(formula ,data=data)
```

```
    if( family == "binomial"){
        response <- cbind(Y,ntrial -Y)
        fit <- glm(response ~ -1 + X, data = data, family = "binomial")
        print(logLik(fit))
        esp <- predict(fit, type = "response")
        res <<- Y/ntrial - esp
        sigma = sd(Y/ntrial - esp)
        phi <- 0.1 * max(dist(coords))
        saida <- c(coef(fit), log(sigma), log(phi))
        names(saida) <- c(colnames(X), "logsigma", "logphi")
        if(nugget == TRUE){
            nugget = 0.1 * sigma
            saida <- c(saida,"logtau" = log(nugget))
        }
    }
    if(family == "poisson"){
        fit <- glm(Y ~ -1 + X, family= "poisson",
                   data = data, offset = log(offset))
        print(logLik(fit))
        esp <- predict(fit)
        sigma <- var(esp - log(Y + 1))
        phi <- 0.1 * max(dist(coords))
        saida <- c(coef(fit), log(sigma), log(phi))
        names(saida) <- c(colnames(X), "logsigma", "logphi")
        if(nugget == TRUE){
            nugget = 0.1 * sigma
            saida <- c(saida, "logtau" = log(nugget))
        }
    }
    if(family == "negative.binomial"){
        fit <- glm.nb(Y ~ -1 + X + offset(log(offset)), data = data)
        print(logLik(fit))
        esp <- predict(fit)
        sigma <- var( esp - log(Y + 1))
        phi <- 0.1 * max(dist(coords))
        saida <- c(coef(fit), log(sigma), log(phi))
        names(saida) <- c(colnames(X), "logsigma", "logphi")
        if(nugget == TRUE){
            nugget = 0.1 * sigma
            saida <- c(saida,"logtau" = log(nugget))
        }
```

```
        saida <- c(saida, "logtheta" = log(fit$theta))
    }
    if(family == "gamma"){
        fit <- glm(Y ~ -1 + X, family = Gamma(link = "log"),
                   data = data)
        te <- summary(fit)
        print(logLik(fit))
        esp <- predict(fit)
        sigma <- var(esp - log(Y))
        phi <- 0.1 * max(dist(coords))
        saida <- c(coef(fit), log(sigma), log(phi))
        names(saida) <- c(colnames(X), "logsigma", "logphi")
        if(nugget == TRUE){
            nugget = 0.1 * sigma
            saida <- c(saida, "logtau" = log(nugget))
        }
        saida <- c(saida, "logtheta" = te$dispersion)
    }
    if(family == "beta"){
        fit <- betareg(Y ~ -1 + X, data = data)
        te <- summary(fit)
        n.beta <- dim(X)[2]
        print(logLik(fit))
        esp <- predict(fit)
        sigma <- var(esp - Y)
        phi <- 0.1 * max(dist(coords))
        saida <- c(coef(fit)[1:n.beta], log(sigma), log(phi))
        names(saida) <- c(colnames(X), "logsigma", "logphi")
        if(nugget == TRUE){
            nugget = 0.1 * sigma
            saida <- c(saida, "logtau" = log(nugget))
        }
        saida <- c(saida, "logtheta" = log(as.numeric(coef(fit)[n.beta+1])))
    }
    if(family == "geometric"){
        fit <- glm(Y ~ -1 + X + offset(log(offset)), data = data,
                   family = negative.binomial(theta = 1))
        print(logLik(fit))
        esp <- predict(fit)
        sigma <- var(esp - log(Y + 1))
        phi <- 0.1 * max(dist(coords))
```

```
        saida <- c(coef(fit), log(sigma), log(phi))
        names(saida) <- c(colnames(X), "logsigma", "logphi")
        if(nugget == TRUE){
            nugget = 0.1 * sigma
            saida <- c(saida,"logtau" = log(nugget))
        }
    }
    return(saida)
}

# Função para ajuste do modelo:
sglmm <- function(formula, cov.model, kappa, inits, data,
                  coords, nugget, family, ntrial = 1, offset = 1,
                  method.optim = "BFGS", method.integrate = "NR",
                  predict = TRUE){
    formula <- as.formula(formula)
    mf <- model.frame(formula,data = data)
    Y <- model.response(mf)
    X <- model.matrix(formula, data = data)
    mat.dist <- dist(coords)
    names <- c(colnames(X), "logsigma2", "logphi")
    n.beta <- dim(X)[2]
    if(family == "negative.binomial" & nugget == TRUE){
        names <- c(names,"logtau2","logprec")
    }
    if(family == "gamma" & nugget == TRUE){
        names <- c(names,"logtau2","logprec")
    }
    if(family == "beta" & nugget == TRUE){
        names <- c(names,"logtau2","logprec")
    }
    if(family == "negative.binomial" & nugget == FALSE){
        names <- c(names,"logprec")
    }
    if(family == "gamma" & nugget == FALSE){
        names <- c(names,"logprec")
    }
    if(family == "beta" & nugget == FALSE){
        names <- c(names,"logprec")
    }
    if(nugget == TRUE & family == "poisson"){
```

```
    names <- c(names,"logtau2")
}
if(nugget == TRUE & family == "binomial"){
    names <- c(names,"logtau2")
}
if(nugget == TRUE & family == "geometric"){
    names <- c(names,"logtau2")
}
parnames(loglik.sglmm) <- names
if (is.null(inits)) {
    inits <- start.values.glgm(formula, data, coords, nugget,
                               family, ntrial, offset = offset)
}
names(inits) <- names
estimativas <- mle2(loglik.sglmm, start = inits,
                    vecpar = TRUE,
                    method = method.optim,
                    control = list(maxit=1000),
                    skip.hessian = FALSE,
                    data = list(Y = Y, X = X, mat.dist = mat.dist,
                                cov.model = cov.model,
                                nugget = nugget, ntrial = ntrial,
                                family = family, kappa =  kappa,
                                method.integrate = method.integrate,
                                offset = offset))
preditos <- preditos(par = coef(estimativas), Y = Y,
                     X = X, kappa = kappa, nugget = nugget,
                     mat.dist = mat.dist, cov.model = cov.model,
                     family = family, method.integrate = method.integrate,
                     offset = offset)
n.pars <- length(coef(estimativas))
summary.estimativas <- summary(estimativas)
summary.estimativas@coef[,1][c(n.beta + 1):n.pars] <-
    exp(summary.estimativas@coef[,1][c(n.beta+1):n.pars])
std.error = sqrt(exp(summary.estimativas@coef[,1][c(n.beta+1):n.pars])^2 *
                 (summary.estimativas@coef[,2][c(n.beta+1):n.pars]^2))
summary.estimativas@coef[,2] <- c(summary.estimativas@coef[,2][1:n.beta],
                                  std.error)
summary.estimativas@coef[,3] <- summary.estimativas@coef[,1]/
    summary.estimativas@coef[,2]
summary.estimativas@coef[,4] <- NA
```

```
    if(predict == TRUE) {
        saida <- list()
        saida[1][[1]] <- summary.estimativas
        saida[7][[1]] <- logLik(estimativas)
        saida[2][[1]] <- preditos
        saida[3][[1]] <- coords
        saida[4][[1]] <- cov.model
        saida[5][[1]] <- family
        saida[6][[1]] <- exp(coef(estimativas)[c(n.beta + 1):n.pars])
        saida[8][[1]] <- ifelse(cov.model == "matern", kappa, "NULL")
        saida[9][[1]] <- estimativas
        return(saida)}
    if(predict == FALSE){
        return(estimativas)
    }
}
```

APÊNDICE B - Script para ajuste dos modelos

```r
#----Ajuste para Weed----

# Dados presentes no pacote geoCount:
load("~/R/geoCount/data/Weed.RData")

# Nome das colunas:
names(Weed) <- c("x1", "x2", "y", "Estimado")

weed_geo <- as.geodata(Weed, coords.col = 1:2,
                       data.col = 3)

bordas <- Weed[alphahull::chull(Weed$x1,
                                Weed$x2), 1:2]

weed_geo$borders <- bordas

plot(weed_geo)

#----Modelos para Poisson----

# Ajustes com efeito de pepita:

# Valores iniciais:
theta_ini <- start.values.sglmm(y ~ 1, data = Weed, family = "poisson",
                                coords = Weed[, 1:2], nugget = T)

# Modelo matern com k = 1:
fit_weed_1 <- sglmm(y ~ 1, data = Weed, coords = Weed[, 1:2],
                    family = "poisson", inits = theta_ini, nugget = T,
                    cov.model = "matern", kappa = 1)

# Modelo matern com k = 1.5:
fit_weed_1.5 <- sglmm(y ~ 1, data = Weed, coords = Weed[, 1:2],
                      family = "poisson", inits = theta_ini, nugget = T,
                      cov.model = "matern", kappa = 1.5)

# Modelo matern com k = 2:
fit_weed_2 <- sglmm(y ~ 1, data = Weed, coords = Weed[, 1:2],
```

```
                    family = "poisson", inits = theta_ini, nugget = T,
                    cov.model = "matern", kappa = 2)

# Modelo exponencial:
fit_weed_exp <- sglmm(y ~ 1, data = Weed, coords = Weed[, 1:2],
                      family = "poisson", inits = theta_ini, nugget = T,
                      cov.model = "exponential", kappa = NULL)

# Modelo esférico:
fit_weed_esf <- sglmm(y ~ 1, data = Weed, coords = Weed[, 1:2],
                      family = "poisson", inits = theta_ini, nugget = T,
                      cov.model = "spherical", kappa = NULL)

# Ajustes sem efeito de pepita:

# Valores iniciais:
theta_ini_2 <- start.values.sglmm(y ~ 1, data = Weed, family = "poisson",
                                  coords = Weed[, 1:2], nugget = F)


# Modelo matern com k = 1:
fit_weed_1_2 <- sglmm(y ~ 1, data = Weed, coords = Weed[, 1:2],
                      family = "poisson", inits = theta_ini_2, nugget = F,
                      cov.model = "matern", kappa = 1)

# Modelo matern com k = 1.5:
fit_weed_1.5_2 <- sglmm(y ~ 1, data = Weed, coords = Weed[, 1:2],
                        family = "poisson", inits = theta_ini_2, nugget = F,
                        cov.model = "matern", kappa = 1.5)

# Modelo matern com k = 2:
fit_weed_2_2 <- sglmm(y ~ 1, data = Weed, coords = Weed[, 1:2],
                      family = "poisson", inits = theta_ini_2, nugget = F,
                      cov.model = "matern", kappa = 2)

# Modelo exponencial:
fit_weed_exp_2 <- sglmm(y ~ 1, data = Weed, coords = Weed[, 1:2],
                        family = "poisson", inits = theta_ini_2, nugget = F,
                        cov.model = "exponential", kappa = NULL)

# Modelo esférico:
```

```r
fit_weed_esf_2 <- sglmm(y ~ 1, data = Weed, coords = Weed[, 1:2],
                        family = "poisson", inits = theta_ini_2, nugget = F,
                        cov.model = "spherical", kappa = NULL)

# Perfis de verossimilhança para o melhor modelo:
perfil_beta0_1 <- profile(fit_weed_1_2[[9]], which = 1)
perfil_sigma_1 <- profile(fit_weed_1_2[[9]], which = 2)
perfil_phi_1 <- profile(fit_weed_1_2[[9]], which = 3)

# Gráfico dos perfis de verossimilhança:
par(mfrow = c(2, 2))
plot(perfil_beta0_1)
plot(perfil_sigma_1)
plot(perfil_phi_1)

#----Modelos para Binomial Negativa----

# Ajustes com efeito de pepita:

# Valores iniciais:
theta_ini_bn <- start.values.sglmm(y ~ 1, data = Weed, family = "negative.binomial",
                                   coords = Weed[, 1:2], nugget = T)

# Modelo exponencial:
fit_weed_exp_nb <- sglmm(y ~ 1, data = Weed, cov.model = "exponential", kappa = NULL,
                         coords = Weed[, 1:2], inits = theta_ini_bn, nugget = T,
                         family = "negative.binomial")

# Modelo matern com k = 1:
fit_weed_1_nb <- sglmm(y ~ 1, data = Weed, cov.model = "matern", kappa = 1,
                       coords = Weed[, 1:2], inits = theta_ini_bn, nugget = T,
                       family = "negative.binomial")

# Modelo matern com k = 1.5:
fit_weed_1.5_nb <- sglmm(y ~ 1, data = Weed, cov.model = "matern", kappa = 1.5,
                         coords = Weed[, 1:2], inits = theta_ini_bn, nugget = T,
                         family = "negative.binomial")

# Modelo matern com k = 2:
fit_weed_2_nb <- sglmm(y ~ 1, data = Weed, cov.model = "matern", kappa = 2,
                       coords = Weed[, 1:2], inits = theta_ini_bn, nugget = T,
```

```
                      family = "negative.binomial")

# Modelo esférico:
fit_weed_esf_nb <- sglmm(y ~ 1, data = Weed, cov.model = "spherical", kappa = NULL,
                         coords = Weed[, 1:2], inits = theta_ini_bn, nugget = T,
                         family = "negative.binomial")

# Ajustes sem efeito de pepita:

# Valores iniciais:
theta_ini_2_bn <- start.values.sglmm(y ~ 1, data = Weed, family = "negative.binomial",
                                     coords = Weed[, 1:2], nugget = F)

# Modelo exponencial:
fit_weed_exp_2_nb <- sglmm(y ~ 1, data = Weed, cov.model = "exponential", kappa = NULL,
                         coords = Weed[, 1:2], inits = theta_ini_2_bn, nugget = F,
                         family = "negative.binomial")

# Modelo matern com k = 1:
fit_weed_1_2_nb <- sglmm(y ~ 1, data = Weed, cov.model = "matern", kappa = 1,
                         coords = Weed[, 1:2], inits = theta_ini_2_bn, nugget = F,
                         family = "negative.binomial")

# Modelo matern com k = 1.5:
fit_weed_1.5_2_nb <- sglmm(y ~ 1, data = Weed, cov.model = "matern", kappa = 1.5,
                           coords = Weed[, 1:2], inits = theta_ini_2_bn, nugget = F,
                           family = "negative.binomial")

# Modelo matern com k = 2:
fit_weed_2_2_nb <- sglmm(y ~ 1, data = Weed, cov.model = "matern", kappa = 2,
                         coords = Weed[, 1:2], inits = theta_ini_2_bn, nugget = F,
                         family = "negative.binomial")

# Modelo esférico:
fit_weed_esf_2_nb <- sglmm(y ~ 1, data = Weed, cov.model = "spherical", kappa = NULL,
                           coords = Weed[, 1:2], inits = theta_ini_2_bn, nugget = F,
                           family = "negative.binomial")

# Perfis de verossimilhança para o melhor modelo:

p_b_0 <- profile(fit_weed_esf_2_nb[[9]], which = 1)
```

```r
p_s_0 <- profile(fit_weed_esf_2_nb[[9]], which = 2)
p_ph_0 <- profile(fit_weed_esf_2_nb[[9]], which = 3)
p_pr_0 <- profile(fit_weed_esf_2_nb[[9]], which = 4)

# Gráfico dos perfis de verossimilhança:
par(mfrow = c(2, 2))
plot(p_b_0)
plot(p_s_0)
plot(p_ph_0)
plot(p_pr_0)

# Krigagem e predição espacial:

gr <- pred_grid(bordas, by = 6)
kc <- krige.control(beta = coef(fit_weed_1_2[[9]])[1],
                    cov.pars = fit_weed_1_2[[6]],
                    cov.model = "matern", kappa = 1)
oc <- output.control(n.predictive = 1000, simulations.predictive = T,
                     threshold = 250)
pred <- krige.conv(weed_geo, locations = gr, borders = bordas,
                   krige = kc, output = oc)

image(pred, col = hcl.colors(20, "blues", rev = T),
      x.leg = c(0, 200), y.leg = c(0, 30))

#----Ajuste para dados SPT----

# Os dados me foram enviados diretamente:
spt <- readxl::read_xlsx("~/Downloads/SPT_PJ.xlsm")
bor <- read.delim("~/Downloads/bordersPJ.txt", sep = ",")
names(spt) <- c("Z", "x1", "x2", "y")

# Removendo valores (provavelmente) truncados:
spt$y[spt$y == 99] <- NA
spt$Z <- -1 * as.numeric(spt$Z)
spt$x2 <- max(spt$x2) - spt$x2

spt_geo <- as.geodata(spt, coords.col = 2:3, data.col = 4)
spt_geo$borders <- bor

plot(spt_geo)
```

```r
boxplot(y ~ Z, data = spt)

## Removendo NAs:
spt_4 <- subset(spt, Z == 4 & !is.na(y))

spt_4_geo <- as.geodata(spt_4, coords.col = 2:3,
                        data.col = 4)

spt_13 <- subset(spt, Z == 13 & !is.na(y))

spt_13_geo <- as.geodata(spt_13, coords.col = 2:3,
                         data.col = 4)

spt_4_geo$borders <- bor
spt_13_geo$borders <- bor

plot(spt_4_geo)
plot(spt_13_geo)

# Ajustes para profundidade 4 m:

# Valores iniciais sem pepita:
inits_4 <- start.values.sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                              nugget = F, family = "poisson")

# Valores iniciais com pepita:
inits_4_nug <- start.values.sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                                  nugget = T, family = "poisson")

# Modelos ajustados:

# Exponencial:
mod_exp_4 <- sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                   cov.model = "exponential", kappa = "NULL",
                   inits = inits_4, nugget = F, family = "poisson")

mod_exp_4_nug <- sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                       cov.model = "exponential", kappa = "NULL",
                       inits = inits_4_nug, nugget = T, family = "poisson")

# Esférico:
```

```r
mod_esf_4 <- sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                   cov.model = "spherical", kappa = NULL,
                   inits = inits_4, nugget = F, family = "poisson")

mod_esf_4_nug <- sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                       cov.model = "spherical", kappa = NULL,
                       inits = inits_4_nug, nugget = T, family = "poisson")

# Matern com k = 1:
mod_1_4 <- sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                 cov.model = "matern", kappa = 1,
                 inits = inits_4, nugget = F, family = "poisson")

mod_1_4_nug <- sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                 cov.model = "matern", kappa = 1,
                 inits = inits_4_nug, nugget = T, family = "poisson")

# Matern com k = 1.5:
mod_1.5_4 <- sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                   cov.model = "matern", kappa = 1.5,
                   inits = inits_4, nugget = F, family = "poisson")

mod_1.5_4_nug <- sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                       cov.model = "matern", kappa = 1.5,
                       inits = inits_4_nug, nugget = T, family = "poisson")

# Matern com k = 2:
mod_2_4 <- sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                 cov.model = "matern", kappa = 2,
                 inits = inits_4, nugget = F, family = "poisson")

mod_2_4_nug <- sglmm(y ~ 1, data = spt_4, coords = spt_4[, 2:3],
                     cov.model = "matern", kappa = 2,
                     inits = inits_4_nug, nugget = T, family = "poisson")

# Perfis de verossimilhança para o melhor modelo:
perf_b_1_4 <- profile(mod_2_4_nug[[9]], which = 1)
perf_s_1_4 <- profile(mod_2_4_nug[[9]], which = 2)
perf_p_1_4 <- profile(mod_2_4_nug[[9]], which = 3)
perf_t_1_4 <- profile(mod_2_4_nug[[9]], which = 4)
```

```r
# Gráfico dos perfis de verossimilhança:
par(mfrow = c(2, 2))
plot(perf_b_1_4)
plot(perf_s_1_4, xlim = c(-10, 2))
plot(perf_p_1_4, xlim = c(-2, 5))
plot(perf_t_1_4, xlim = c(-3, -0.8))

# Ajuste para profundidade 13 m:

# Valores iniciais sem pepita:
inits_13 <- start.values.sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
                               nugget = F, family = "poisson")

# Valores iniciais com pepita:
inits_13_nug <- start.values.sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
                                   nugget = T, family = "poisson")

# Modelos ajustados:

# Exponencial:
mod_exp <- sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
                 cov.model = "exponential", kappa = "NULL",
                 inits = inits_13, nugget = F, family = "poisson")

mod_exp_nug <- sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
                     cov.model = "exponential", kappa = "NULL",
                     inits = inits_13_nug, nugget = T, family = "poisson")

# Esférico:
mod_esf <- sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
                 cov.model = "spherical", kappa = NULL,
                 inits = inits_13, nugget = F, family = "poisson")

mod_esf_nug <- sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
                     cov.model = "spherical", kappa = NULL,
                     inits = inits_13_nug, nugget = T, family = "poisson")

# Matèrn com k = 1:
mod_1 <- sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
               cov.model = "matern", kappa = 1,
               inits = inits_13, nugget = F, family = "poisson")
```

```r
mod_1_nug <- sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
                   cov.model = "matern", kappa = 1,
                   inits = inits_13_nug, nugget = T, family = "poisson")

# Matern com k = 1.5:
mod_1.5 <- sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
                 cov.model = "matern", kappa = 1.5,
                 inits = inits_13, nugget = F, family = "poisson")

mod_1.5_nug <- sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
                     cov.model = "matern", kappa = 1.5,
                     inits = inits_13_nug, nugget = T, family = "poisson")

# Matern com k = 2:
mod_2 <- sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
               cov.model = "matern", kappa = 2,
               inits = inits_13, nugget = F, family = "poisson")

mod_2_nug <- sglmm(y ~ 1, data = spt_13, coords = spt_13[, 2:3],
                   cov.model = "matern", kappa = 2,
                   inits = inits_13_nug, nugget = T, family = "poisson")

# Perfis de verossimilhança para o melhor modelo:
perf_b_1 <- profile(mod_2[[9]], which = 1)
perf_s_1 <- profile(mod_2[[9]], which = 2)
perf_p_1 <- profile(mod_2[[9]], which = 3)

# Gráfico dos perfis de verossimilhança:
par(mfrow = c(2, 2))
plot(perf_b_1)
plot(perf_s_1)
plot(perf_p_1)

# Krigagem e predição espacial

par(mfrow = c(1, 1))

gr_4 <- pred_grid(bor, by = 0.5)
kc_4 <- krige.control(beta = coef(mod_2_4_nug[[9]])[1],
                      cov.pars = mod_1_4_nug[[6]][1:2],
                      cov.model = "matern", kappa = 2,
```

```
                        nugget = mod_2_4_nug[[6]][3])
oc_4 <- output.control(n.predictive = 1000, simulations.predictive = T,
                       threshold = 250)
pred_4 <- krige.conv(spt_4_geo, locations = gr_4, borders = bor,
                     krige = kc_4, output = oc_4)

image(pred_4, col = hcl.colors(20, "blues", rev = T),
      x.leg = c(-15, 0), y.leg = c(-15, -10))

gr_13 <- pred_grid(bor, by = 0.5)
kc_13 <- krige.control(beta = coef(mod_2[[9]])[1],
                       cov.pars = mod_2[[6]],
                       cov.model = "matern", kappa = 2)
oc_13 <- output.control(n.predictive = 1000, simulations.predictive = T,
                        threshold = 250)
pred_13 <- krige.conv(spt_13_geo, locations = gr_13, borders = bor,
                      krige = kc_13, output = oc_13)

image(pred_13, col = hcl.colors(20, "blues", rev = T),
      x.leg = c(-15, 0), y.leg = c(-15, -10))

#----Estudo de simulação----

## Usando processamento em paralelo:

library(parallel)
library(doParallel)

# Definindo as funções de simulação:

# Função usando aproximação de Laplace:
simulation_function <- function(i, n = 100, cov.model = "exponential", kappa = "NULL",
                                xlims = c(0, 200), ylims = c(0, 200)) {
  set.seed(i)
  sim.g <- grf(n = n, grid = "irreg", cov.pars = c(0.5, 30),
               mean = 2, nugget = 0.05, xlims = xlims,
               ylims = ylims, cov.model = "exponential")
  sim <- list(coords = sim.g$coords)
  attr(sim, "class") <- "geodata"
  sim$data <- rpois(n, lambda = exp(sim.g$data))
  sim_df <- data.frame(x1 = sim$coords[, 1],
```

```
                        x2 = sim$coords[, 2],
                        y = sim$data)
  val_ini_simu <- start.values.sglmm(y ~ 1, family = "poisson", data = sim_df,
                                     coords = sim_df[, 1:2], nugget = TRUE)
  ajuste_ini_simu <- sglmm(y ~ 1, cov.model = cov.model, kappa = kappa,
                           inits = val_ini_simu, data = sim_df,
                           coords = sim_df[, 1:2], nugget = TRUE, family = "poisson",
                           method.optim = "BFGS", method.integrate = "NR")
  return(c(ajuste_ini_simu[[9]]@coef[1], ajuste_ini_simu[[6]]))
}

# Função usando algoritmo MCMC:
simulation_function_2 <- function(i, n = 100, cov.model = "exponential", kappa = "NULL",
                                  xlims = c(0, 200), ylims = c(0, 200)) {
  set.seed(i)
  sim.g <- grf(n = n, grid = "irreg", cov.pars = c(0.5, 30),
               mean = 2, nugget = 0.05, xlims = xlims,
               ylims = ylims, cov.model = "exponential")
  sim <- list(coords = sim.g$coords)
  attr(sim, "class") <- "geodata"
  sim$data <- rpois(n, lambda = exp(sim.g$data))
  sim_df <- data.frame(x1 = sim$coords[, 1],
                       x2 = sim$coords[, 2],
                       y = sim$data)
  val_ini_simu <- start.values.sglmm(y ~ 1, family = "poisson", data = sim_df,
                                     coords = sim_df[, 1:2], nugget = TRUE)
  mcmc <- list(cov.pars = exp(val_ini_simu[2:4]), link = "log",
               beta = val_ini_simu[1], family = "poisson",
               cov.model = cov.model, kappa = kappa)
  S.prop <- mcmc.control(S.scale=0.1, thin=10)
  tune.S <- glsm.mcmc(sim, model = mcmc,
                      mcmc.input = S.prop)
  S.control <- mcmc.control(S.scale = 0.5, thin = 50, burn.in = 10000)
  S.sims <- glsm.mcmc(sim, model = mcmc, mcmc.in = S.control)
  lik.control <- prepare.likfit.glsm(S.sims)
  mc.fit <- likfit.glsm(lik.control, ini.phi = exp(val_ini_simu[3]),
                        fix.nugget.rel = F)
  return(c(mc.fit$beta, mc.fit$cov.pars, mc.fit$nugget.rel))
}
```

```r
# Cria o cluster, com n-1 núcleos para processamento:
no_cores <- detectCores() - 1
cl <- makeCluster(no_cores)

# Exporta as bibliotecas necessárias para o cluster:
clusterEvalQ(cl, {
  library(geoR)
  library(bbmle)
})

# Exporta as funções necessárias para o cluster:
clusterExport(cl, varlist = c("simulation_function", "simulation_function_2"
                              "start.values.sglmm", "gauss.mult",
                              "sglmm","loglik.sglmm", "parnames<-",
                              "mle2", "forceSymmetric", "monta.sigma",
                              "laplace", "preditos", "Q.b", "Q.b.grad",
                              "Q.b.hess", "newton.raphson"))

#----Ajustes para diferentes tamanhos de amostra:----

# Por aproximação de Laplace:

resultados_50 <- parLapply(cl, 1:1000, simulation_function, n = 50)
resultados_100 <- parLapply(cl, 1:1000, simulation_function, n = 100)
resultados_200 <- parLapply(cl, 1:1000, simulation_function, n = 200)

resultados_50 <- do.call(rbind, resultados_50)
resultados_100 <- do.call(rbind, resultados_100)
resultados_200 <- do.call(rbind, resultados_200)

eqm_b0_50 <- mean((resultados_50[, 1] - 2)^2)
eqm_b0_100 <- mean((resultados_100[, 1] - 2)^2)
eqm_b0_200 <- mean((resultados_200[, 1] - 2)^2)

eqm_s_50 <- mean((resultados_50[, 2] - 0.5)^2)
eqm_s_100 <- mean((resultados_100[, 2] - 0.5)^2)
eqm_s_200 <- mean((resultados_200[, 2] - 0.5)^2)

eqm_p_50 <- mean((resultados_50[, 3] - 30)^2)
eqm_p_100 <- mean((resultados_100[, 3] - 30)^2)
eqm_p_200 <- mean((resultados_200[, 3] - 30)^2)
```

```r
eqm_t_50 <- mean((resultados_50[, 4] - 0.05)^2)
eqm_t_100 <- mean((resultados_100[, 4] - 0.05)^2)
eqm_t_200 <- mean((resultados_200[, 4] - 0.05)^2)

media_n <- do.call(rbind, lapply(list(resultados_50,
                                      resultados_100,
                                      resultados_200),
                                 colMeans))

var_n <- do.call(rbind, lapply(list(resultados_50,
                                    resultados_100,
                                    resultados_200),
                               FUN = function(x) {apply(x, 2, var)}))

eqm_n <- matrix(c(eqm_b0_50, eqm_b0_100, eqm_b0_200,
                  eqm_s_50, eqm_s_100, eqm_s_200,
                  eqm_p_50, eqm_p_100, eqm_p_200,
                  eqm_t_50, eqm_t_100, eqm_t_200), nrow = 3)

vies_n <- eqm_n - var_n

# Por MCMC

resultados_50_mcmc <- parLapply(cl, 1:1000, simulation_function_2, n = 50)
resultados_100_mcmc <- parLapply(cl, 1:1000, simulation_function_2, n = 100)
resultados_200_mcmc <- parLapply(cl, 1:1000, simulation_function_2, n = 200)

resultados_50_mcmc <- do.call(rbind, resultados_50_mcmc)
resultados_100_mcmc <- do.call(rbind, resultados_100_mcmc)
resultados_200_mcmc <- do.call(rbind, resultados_200_mcmc)

eqm_b0_50_mcmc <- mean((resultados_50_mcmc[, 1] - 2)^2)
eqm_b0_100_mcmc <- mean((resultados_100_mcmc[, 1] - 2)^2)
eqm_b0_200_mcmc <- mean((resultados_200_mcmc[, 1] - 2)^2)

eqm_s_50_mcmc <- mean((resultados_50_mcmc[, 2] - 0.5)^2)
eqm_s_100_mcmc <- mean((resultados_100_mcmc[, 2] - 0.5)^2)
eqm_s_200_mcmc <- mean((resultados_200_mcmc[, 2] - 0.5)^2)

eqm_p_50_mcmc <- mean((resultados_50_mcmc[, 3] - 30)^2)
eqm_p_100_mcmc <- mean((resultados_100_mcmc[, 3] - 30)^2)
```

```r
eqm_p_200_mcmc <- mean((resultados_200_mcmc[, 3] - 30)^2)

eqm_t_50_mcmc <- mean((resultados_50_mcmc[, 4] - 0.05)^2)
eqm_t_100_mcmc <- mean((resultados_100_mcmc[, 4] - 0.05)^2)
eqm_t_200_mcmc <- mean((resultados_200_mcmc[, 4] - 0.05)^2)

media_n_mcmc <- do.call(rbind, lapply(list(resultados_50_mcmc,
                                           resultados_100_mcmc,
                                           resultados_200_mcmc),
                                      colMeans))


var_n_mcmc <- do.call(rbind,
                      lapply(list(resultados_50_mcmc,
                                  resultados_100_mcmc,
                                  resultados_200_mcmc),
                             FUN = function(x) {apply(x, 2, var)}))

eqm_n_mcmc <- matrix(c(eqm_b0_50_mcmc, eqm_b0_100_mcmc, eqm_b0_200_mcmc,
                       eqm_s_50_mcmc, eqm_s_100_mcmc, eqm_s_200_mcmc,
                       eqm_p_50_mcmc, eqm_p_100_mcmc, eqm_p_200_mcmc,
                       eqm_t_50_mcmc, eqm_t_100_mcmc, eqm_t_200_mcmc), nrow = 3)

vies_n_mcmc <- eqm_n_mcmc - var_n_mcmc

#----Ajustes para modelos de covariância incorretos:----

resultados_esfe <- parLapply(cl, 1:1000, simulation_function,
                             cov.model = "spherical")
resultados_mat_1 <- parLapply(cl, 1:1000, simulation_function,
                              cov.model = "matern", kappa = 1)
resultados_mat_2 <- parLapply(cl, 1:1000, simulation_function,
                              cov.model = "matern", kappa = 2)

resultados_esfe <- do.call(rbind, resultados_esfe)
resultados_mat_1 <- do.call(rbind, resultados_mat_1)
resultados_mat_2 <- do.call(rbind, resultados_mat_2)

media_cov <- do.call(rbind, (lapply(list(resultados_100,
                                         resultados_esfe,
                                         resultados_mat_1,
```

```
                                        resultados_mat_2),
                                    colMeans)))

var_cov <- do.call(rbind, lapply(list(resultados_100,
                                      resultados_esfe,
                                      resultados_mat_1,
                                      resultados_mat_2),
                                 FUN = function(x) {apply(x, 2, var)}))

eqm_b0_esfe <- mean((resultados_esfe[, 1] - 2)^2)
eqm_b0_mat1 <- mean((resultados_mat_1[, 1] - 2)^2)
eqm_b0_mat2 <- mean((resultados_mat_2[, 1] - 2)^2)

eqm_s_esfe <- mean((resultados_esfe[, 2] - 0.5)^2)
eqm_s_mat1 <- mean((resultados_mat_1[, 2] - 0.5)^2)
eqm_s_mat2 <- mean((resultados_mat_2[, 2] - 0.5)^2)

eqm_p_esfe <- mean((resultados_esfe[, 3] - 30)^2)
eqm_p_mat1 <- mean((resultados_mat_1[, 3] - 30)^2)
eqm_p_mat2 <- mean((resultados_mat_2[, 3] - 30)^2)

eqm_t_esfe <- mean((resultados_esfe[, 4] - 0.05)^2)
eqm_t_mat1 <- mean((resultados_mat_1[, 4] - 0.05)^2)
eqm_t_mat2 <- mean((resultados_mat_2[, 4] - 0.05)^2)

eqm_cov <- matrix(c(eqm_b0_100, eqm_b0_esfe, eqm_b0_mat1, eqm_b0_mat2,
                    eqm_s_100, eqm_s_esfe, eqm_s_mat1, eqm_s_mat2,
                    eqm_p_100, eqm_p_esfe, eqm_p_mat1, eqm_p_mat2,
                    eqm_t_100, eqm_t_esfe, eqm_t_mat1, eqm_t_mat2), nrow = 4)

vies_cov <- eqm_cov - var_cov

# Por MCMC

resultados_esfe_mcmc <- parLapply(cl, 1:1000, simulation_function_2,
                                  cov.model = "spherical")
resultados_mat_1_mcmc <- parLapply(cl, 1:1000, simulation_function_2,
                                   cov.model = "matern", kappa = 1)
resultados_mat_2_mcmc <- parLapply(cl, 1:1000, simulation_function_2,
                                   cov.model = "matern", kappa = 2)
```

```
resultados_esfe_mcmc <- do.call(rbind, resultados_esfe_mcmc)
resultados_mat_1_mcmc <- do.call(rbind, resultados_mat_1_mcmc)
resultados_mat_2_mcmc <- do.call(rbind, resultados_mat_2_mcmc)

media_cov_mcmc <- do.call(rbind, (lapply(list(resultados_100_mcmc,
                                              resultados_mat_1_mcmc,
                                              resultados_mat_2_mcmc,
                                              resultados_esfe_mcmc),
                                         colMeans)))

var_cov_mcmc <- do.call(rbind,
                        lapply(list(resultados_100_mcmc,
                                    resultados_mat_1_mcmc,
                                    resultados_mat_2_mcmc,
                                    resultados_esfe_mcmc),
                               FUN = function(x) {apply(x, 2, var)}))

eqm_b0_esfe_mcmc <- mean((resultados_esfe_mcmc[, 1] - 2)^2)
eqm_b0_mat1_mcmc <- mean((resultados_mat_1_mcmc[, 1] - 2)^2)
eqm_b0_mat2_mcmc <- mean((resultados_mat_2_mcmc[, 1] - 2)^2)

eqm_s_esfe_mcmc <- mean((resultados_esfe_mcmc[, 2] - 0.5)^2)
eqm_s_mat1_mcmc <- mean((resultados_mat_1_mcmc[, 2] - 0.5)^2)
eqm_s_mat2_mcmc <- mean((resultados_mat_2_mcmc[, 2] - 0.5)^2)

eqm_p_esfe_mcmc <- mean((resultados_esfe_mcmc[, 3] - 30)^2)
eqm_p_mat1_mcmc <- mean((resultados_mat_1_mcmc[, 3] - 30)^2)
eqm_p_mat2_mcmc <- mean((resultados_mat_2_mcmc[, 3] - 30)^2)

eqm_t_esfe_mcmc <- mean((resultados_esfe_mcmc[, 4] - 0.05)^2)
eqm_t_mat1_mcmc <- mean((resultados_mat_1_mcmc[, 4] - 0.05)^2)
eqm_t_mat2_mcmc <- mean((resultados_mat_2_mcmc[, 4] - 0.05)^2)

eqm_cov_mcmc <- matrix(c(eqm_b0_100_mcmc, eqm_b0_mat1_mcmc,
                         eqm_b0_mat2_mcmc, eqm_b0_esfe_mcmc,
                         eqm_s_100_mcmc, eqm_s_mat1_mcmc,
                         eqm_s_mat2_mcmc, eqm_s_esfe_mcmc,
                         eqm_p_100_mcmc, eqm_p_mat1_mcmc,
                         eqm_p_mat2_mcmc, eqm_p_esfe_mcmc,
                         eqm_t_100_mcmc, eqm_t_mat1_mcmc,
                         eqm_t_mat2_mcmc, eqm_t_esfe_mcmc), nrow = 4)
```

```
vies_cov_mcmc <- eqm_cov_mcmc - var_cov_mcmc

#----Ajustes para espaço amostral maior/menor----

# Por aproximação de Laplace:

resultados_menor <- parLapply(cl, 1:1000, simulation_function,
                              xlims = c(0, 50), ylims = c(0, 50))
resultados_maior <- parLapply(cl, 1:1000, simulation_function,
                              xlims = c(0, 500), ylims = c(0, 500))

resultados_menor <- do.call(rbind, resultados_menor)
resultados_maior <- do.call(rbind, resultados_maior)

media_grid <- do.call(rbind, (lapply(list(resultados_menor,
                                          resultados_100,
                                          resultados_maior),
                                     colMeans)))

var_grid <- do.call(rbind, lapply(list(resultados_menor,
                                       resultados_100,
                                       resultados_maior),
                                  FUN = function(x) {apply(x, 2, var)}))

eqm_b0_menor <- mean((resultados_menor[, 1] - 2)^2)
eqm_b0_maior <- mean((resultados_maior[, 1] - 2)^2)

eqm_s_menor <- mean((resultados_menor[, 2] - 0.5)^2)
eqm_s_maior <- mean((resultados_maior[, 2] - 0.5)^2)

eqm_p_menor <- mean((resultados_menor[, 3] - 30)^2)
eqm_p_maior <- mean((resultados_maior[, 3] - 30)^2)

eqm_t_menor <- mean((resultados_menor[, 4] - 0.05)^2)
eqm_t_maior <- mean((resultados_maior[, 4] - 0.05)^2)

eqm_grid <- matrix(c(eqm_b0_menor, eqm_b0_100, eqm_b0_maior,
                     eqm_s_menor, eqm_s_100, eqm_s_maior,
                     eqm_p_menor, eqm_p_100, eqm_p_maior,
                     eqm_t_menor, eqm_t_100, eqm_t_maior), nrow = 3)
```

```
vies_grid <- eqm_grid - var_grid

# Por MCMC:

resultados_menor_mcmc <- parLapply(cl, 1:1000, simulation_function_2,
                                   xlims = c(0, 50), ylims = c(0, 50))
resultados_maior_mcmc <- parLapply(cl, 1:1000, simulation_function_2,
                                   xlims = c(0, 500), ylims = c(0, 500))

resultados_menor_mcmc <- do.call(rbind, resultados_menor_mcmc)
resultados_maior_mcmc <- do.call(rbind, resultados_maior_mcmc)

media_grid_mcmc <- do.call(rbind, (lapply(list(resultados_menor_mcmc,
                                               resultados_100_mcmc,
                                               resultados_maior_mcmc),
                                          colMeans)))

var_grid_mcmc <- do.call(rbind,
                         lapply(list(resultados_menor_mcmc,
                                     resultados_100_mcmc,
                                     resultados_maior_mcmc),
                                FUN = function(x) {apply(x, 2, var)}))

eqm_b0_menor_mcmc <- mean((resultados_menor_mcmc[, 1] - 2)^2)
eqm_b0_maior_mcmc <- mean((resultados_maior_mcmc[, 1] - 2)^2)

eqm_s_menor_mcmc <- mean((resultados_menor_mcmc[, 2] - 0.5)^2)
eqm_s_maior_mcmc <- mean((resultados_maior_mcmc[, 2] - 0.5)^2)

eqm_p_menor_mcmc <- mean((resultados_menor_mcmc[, 3] - 30)^2)
eqm_p_maior_mcmc <- mean((resultados_maior_mcmc[, 3] - 30)^2)

eqm_t_menor_mcmc <- mean((resultados_menor_mcmc[, 4] - 0.05)^2)
eqm_t_maior_mcmc <- mean((resultados_maior_mcmc[, 4] - 0.05)^2)

eqm_grid_mcmc <- matrix(c(eqm_b0_menor_mcmc, eqm_b0_100_mcmc, eqm_b0_maior_mcmc,
                          eqm_s_menor_mcmc, eqm_s_100_mcmc, eqm_s_maior_mcmc,
                          eqm_p_menor_mcmc, eqm_p_100_mcmc, eqm_p_maior_mcmc,
                          eqm_t_menor_mcmc, eqm_t_100_mcmc, eqm_t_maior_mcmc),
                        nrow = 3)
```

```r
vies_grid_mcmc <- eqm_grid_mcmc - var_grid_mcmc

# Parar o cluster para paralelização:
stopCluster(cl)
```