MSc. Péricles Lopes Machado

Desenvolvimento de algoritmos para geração de modelos geoestatísticos usando aprendizado de máquina e computação de alta-performance

Porto Alegre, RS, Brasil

2022

MSc. Péricles Lopes Machado

Desenvolvimento de algoritmos para geração de modelos geoestatísticos usando aprendizado de máquina e computação de alta-performance

Tese de doutorado.

Universidade Federal do Rio Grande do Sul

Departamento de Engenharia de Minas

PPGE3M - Programa de Pós-Graduação em Engenharia de Minas, Metalúrgica e Materiais

Orientador: Prof. Dr. João Felipe C. L. Costa

Porto Alegre, RS, Brasil

2022

Este trabalho é dedicado à minha mãe,
por seu amor, inspiração e dedicação.

Agradecimentos

Os agradecimentos principais são direcionados ao professor João Felipe, pela orientação, aos colegas do LPM-UFRGS pelo apoio nessa jornada e à minha mãe, por todo amor e suporte ao longo de minha vida.

Agradecimentos especiais são direcionados à Petrobras e CNPq por patrocinar esse trabalho.

"O homem que tem coragem de desperdiçar
uma hora do seu tempo não descobriu o valor da vida."
Charles Darwin

Resumo

A evolução na capacidade de processamento dos computadores viabilizou a implementação de algoritmos de simulação geoestatística. Por meio da simulação, é possível criar uma realização de uma função aleatória com as mesmas características estatísticas dos dados amostrais. Como, na prática, é necessária a geração de várias realizações, esse processo pode ser custoso computacionalmente. Principalmente, se a grade-alvo contiver muitas células. Por isso, nesse trabalho, estuda-se os métodos de simulação espectral (Bandas Rotatórias e Método Integral de Fourier) como alternativa ao algoritmo de Simulação Sequencial Gaussiana (SSG).

Os métodos de simulação espectral eliminam a necessidade de um caminho aleatório no processo de geração da realização de uma função aleatória. Isso torna esses algoritmos extremamente amigáveis à aplicação de técnicas de paralelização e computação distribuída. Além disso, o esforço computacional necessário é significativamente reduzido. Conforme é mostrado nesse trabalho, a simulação espectral permite ganhos significativos de performance sem prejuízo na qualidade das realizações geradas.

Além da utilização dos métodos espectrais como alternativa de alta performance à simulação geoestatística, são estudadas técnicas de otimização matemática para obtenção de modelos de covariância. Baseada na Busca Tabu, é construído um algoritmo para computação automática do modelo de covariância. Também, inspirado em técnicas de validação cruzada, como *Jackknife* e *K-Fold*, é construído um algoritmo de simulação chamado *Ensemble Simulation*. O algoritmo proposto utiliza otimização matemática para encontrar o modelo de covariância mais adequado para uma vizinhança, com uma função objetivo construída a partir do erro de validação. Portanto, por meio da otimização e do *Ensemble Simulation* é desenvolvido um modo de simular a função aleatória sem explicitamente definir um modelo de covariância. Tendo em vista que a modelagem manual do modelo de covariância é a tarefa mais dispendiosa do fluxo de trabalho da simulação geoestatística, essas técnicas permitem a automatização do processo.

Em suma, nesse trabalho, são estudadas técnicas que viabilizam a automatização e melhoria da performance da simulação geoestatística e os resultados demonstram sua utilidade.

Palavras-chaves: Simulação geoestatística, métodos espectrais, aprendizado de máquina, validação cruzada, otimização.

Abstract

The evolution in the processing capacity of computers enabled the implementation of geostatistical simulation algorithms. It is possible to create a realization of a random function with the same statistical characteristics as the sample data through simulation. As, in practice, it is necessary to generate several realizations, this process can be computationally expensive. Mainly if the target grid is large. Therefore, in this work, we study the spectral simulation methods (Turning Bands and Fourier Integral Method) as an alternative to the Sequential Gaussian Simulation (SGS). Spectral simulation eliminates the need for a random path in generating the realization of a random function. This simplifies the usage of parallelization and distributed computing techniques.

Furthermore, the computational effort required is significantly reduced. Therefore, spectral simulation allows significant performance gains without compromising the quality of the generated realizations. In addition, we studied mathematical optimization techniques for obtaining covariance models. An algorithm based on Tabu Search for automatic computation of the covariance model was developed, and also, inspired by cross-validation methods such as Jackknife and K-Fold, a simulation algorithm called Ensemble Simulation is proposed. The proposed algorithm uses mathematical optimization to find the most suitable covariance model for a neighborhood, with an objective function built from the validation error. Therefore, a workflow is developed through optimization and Ensemble Simulation to simulate the random function without explicitly defining a covariance model. Since manual modeling of the covariance model is the most expensive task of the geostatistical simulation workflow, these techniques provided a way to automate this task. In short, in this work, techniques that enable the automation and improvement of the performance of geostatistical simulation were proposed and the results proved their usefulness.

Key-words: Geostatistical simulation, spectral methods, machine learning, cross validation, optimization.

Lista de símbolos

$\mathbb{Q}$	Conjunto de números racionais.
$\mathbb{Z}$	Conjunto de números inteiros.
$\mathbb{Z}^d$	Conjunto de vetores de números inteiros $v = (v_1, v_2, ..., v_d)$.
$\mathbb{R}$	Conjunto de números reais.
$\mathbb{R}^d$	Conjunto de vetores de números reais $\boldsymbol{v} = (v_1, v_2, ..., v_d)$.
$\mathbb{R}^+$	Conjunto de números reais positivos.
$\mathbb{C}$	Conjunto de números complexos $a + bi$.
$\mathbb{C}^d$	Conjunto de vetores de números complexos $\boldsymbol{v} = (v_1, v_2, ..., v_d)$.
$\boldsymbol{u}$	Posição espacial $\boldsymbol{u} = (u_x, u_y, u_z)$.
$\boldsymbol{h}$	Espaçamento (*lag*) definido pelo vetor $\boldsymbol{h} = (h_x, h_y, h_z)$.
h_x	Espaçamento na direção x.
h_y	Espaçamento na direção y.
h_z	Espaçamento na direção z.
$C_{zw}(\boldsymbol{h})$	Covariância no *lag* $\boldsymbol{h}$ para as variáveis z e w.
$C(\boldsymbol{h})$	Auto-covariância no *lag* $\boldsymbol{h}$.
z	Variável z.
$z(\boldsymbol{u})$	Valor da variável z na posição espacial $\boldsymbol{u}$.
m_z	Média da variável z.
$m_{z_{-h}}$	Média da cauda da variável z.
$m_{z_{+h}}$	Média da cabeça da variável z.
$N(\boldsymbol{h})$	Numero de pares para o *lag* $\boldsymbol{h}$.
$s_{zw}(k)$	Densidade espectral no índice inteiro k.
$s_{zw}(\boldsymbol{h})$	Densidade espectral no *lag* $\boldsymbol{h}$.
$i(\boldsymbol{u})$	$i(\boldsymbol{u}) = 1$ se tiver dado na posição u, $i(\boldsymbol{u}) = 0$ caso contrário.
I	Transformada de fourier de $i(\boldsymbol{u})$.
$\rho_{zw}(\boldsymbol{h})$	Correlação cruzada entre as funções z e w para o espaçamento $\boldsymbol{h}$.
$\rho(\boldsymbol{h})$	Auto-correlação para o espaçamento $\boldsymbol{h}$.
$C_{zw}(i, j, k)$	Tabela de covariância cruzada para as variáveis z e w na célula (i, j, k).
$\tilde{C}(\boldsymbol{h})$	Covariância experimental na posição $\boldsymbol{h}$.
$\tilde{C}(i, j, j)$	Tabela de covariância experimental.
$C^s(\boldsymbol{h})$	Covariância experimental suavizada.
$C^s(i, j, j)$	Tabela de covariância experimental suavizada.
$\Omega(\boldsymbol{u})$	Conjunto dos pontos $\boldsymbol{v}$ mais próximos da posição $\boldsymbol{u}$.
$s^s(k)$	Densidade espectral suavizada no índice k.
$s^s(\boldsymbol{u})$	Densidade espectral suavizada na posição $\boldsymbol{u}$.
$\mathscr{F}\{.\}$	Transformada de Fourier.

$< \boldsymbol{u}, \boldsymbol{v} >$	Produto interno entre os vetores $\boldsymbol{u}$ e $\boldsymbol{v}$.
$\|\|\boldsymbol{u}\|\|$	Norma do vetor $\boldsymbol{u}$.
δ_r	Tolerância radial.
Y	Variável aleatória.
$Y_1, Y_2, ..., Y_n$	Sequência com n amostras da variável aleatória Y.
δ_θ	Tolerância angular.
$C_\theta(r)$	Projeção da covariância $C(\boldsymbol{h})$ na direção θ, r é a posição na linha.
$\Delta(\boldsymbol{u}, \boldsymbol{v})$	Função distância em $\mathbb{R}^d$.
S_d	Esfera unitária em $\mathbb{R}^d$.
δ_z	Distribuição de probabilidade da variável z.
σ^2	Variância.
σ	Desvio padrão.
σ_z^2	Variância da variável z.
σ_z	Desvio padrão da variável z.
σ_{zw}^2	Variância cruzada das variáveis z e w.
$G(z)$	Função de distribuição Gaussiana padrão para a variável z.
$P(Y < y)$	Função de densidade de probabilidade.
$< \boldsymbol{u}, \theta_k >$	Projeção de $\boldsymbol{u}$ na direção θ_k.
$C_d(r)$	Covariância escalar com dimensão d.
ρ_k	Projeção de $\boldsymbol{u}$ na direção θ_k.
$z^*(\boldsymbol{u})$	Estimativa na posição $\boldsymbol{u}$.
$\lambda_k(\boldsymbol{u})$	Peso do k-ésimo valor na vizinhança $\Omega(\boldsymbol{u})$ de $\boldsymbol{u}$.
$\lambda_{\boldsymbol{v}}(\boldsymbol{u})$	Peso de $\boldsymbol{v}$ na vizinhança $\Omega(\boldsymbol{u})$ de $\boldsymbol{u}$.
z_k	k-ésima amostra de z.
$U(a, b)$	Distribuição uniforme definida no intervalo (a, b).
$y(\boldsymbol{v})$	Dado condicionante no ponto $\boldsymbol{v}$.
D	Domínio de estimativa/simulação.
CD	Conjunto de dados condicionantes.
$z^{(n)}(\boldsymbol{u})$	Simulação em $\boldsymbol{u}$ usando-se n processos estocásticos.
$Z_k(< \boldsymbol{u}, \theta_k >)$	Processo estocástico na direção θ_k e posição $\boldsymbol{u}$.
$\|A\|$	Tamanho do conjunto A.
T	Conjunto de dados de calibração.
T_k	k-ésima partição do conjunto de dados de calibração.
EQM	Erro quadrático médio.
β	Parâmetros do estimador, função radial ou covariância.

$z(\boldsymbol{u};\beta)$	Estimativa na posição $\boldsymbol{u}$ dado os parâmetros β do estimador.
$\Psi_\beta(\boldsymbol{u},\boldsymbol{v})$	Função radial com parâmetro de forma β.
$C(\boldsymbol{h};\beta)$	Covariância no espaçamento $\boldsymbol{h}$ e transformação anisotrópica β.
$\Omega(\boldsymbol{u},T)$	Vizinhos mais próximos de $\boldsymbol{u}$ no conjunto T.
$\widehat{z}_\beta(\boldsymbol{u})$	Estimador z com parâmetro de forma β otimizado.
$u^{(B)}$	numero de Van Der Carput na base B.
$\boldsymbol{\Theta}$	Matriz de anisotropia.
L	Lagraniano.
$c(\boldsymbol{h})$	Covariância unitária.
η	Efeito pepita.
γ_i	i-ésima amostra do variograma experimental γ.
$\gamma(\boldsymbol{h})$	Variograma em $\boldsymbol{h}$.

Lista de algoritmos

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CPU	Unidade Central de Processamento ou processador (*Central Processing Unit*).
GPU	Unidade de Processamento Gráfico ou placa de vídeo (*Graphics Processing Unit*).
HPC	Computação de alta performance (*High Performance computing*).
SGSIM	Simulação Sequencial Gaussiana.
FIM	Método Integral de Fourier (*Fourier Integral Methdod*).
FFT	Transformada Rápida de Fourier (*Fast Fourier Transform*).
CUDA	Arquitetura unificada de equipamentos de computação (*Compute Unified Device Architecture*).
OpenMP	Multi-processamento aberto (*Open Multi-processing*).
SGSIM	Simulação Sequencial Gaussiana.
SIS	Simulação Sequencial de Indicadores.
TBSIM	Simulação com Bandas Rotativa (*Turning Bands Simulation*).
MPS	Estatística multi-ponto.

Sumário

1 Introdução

O tempo é um recurso caro e limitado. E, por isso, na ciência da computação foi desenvolvida a análise de algoritmos, que é um campo de estudo dedicado ao desenvolvimento de técnicas para resolução eficiente dos mais diversos tipos de problemas computacionais que existem no dia-a-dia. Além disso, muito capital intelectual e financeiro é investido no aprimoramento dos computadores, uma das principais ferramentas que melhoraram significativamente a qualidade da vida humana, reduzindo o tempo para execução de tarefas repetitivas (como cálculos numéricos em modelos matemáticos aplicados nos mais diversos campos de estudo ou fabricação dos mais diferentes tipos de produtos).

Portanto, é função fundamental da engenharia e da ciência encontrar soluções que poupem tempo e recursos. Mas, resolver um problema computacional de forma correta, eficiente e elegante exige uma combinação de diversos fatores, como entendimento dos conceitos envolvidos, capacidade da tecnologia e recursos disponíveis (capital, maquinário, *software*, entre outros). Em computação, isso se traduz na sinergia entre a infra-estrutura computacional instalada e a disponibilidade de *software* que utilize plenamente esse potencial. O que se tem visto na prática é que há um descompasso entre a evolução do *hardware* e do *software* [1] (WIRTH, 1995) - fato conhecido como Lei de Wirth - , ou seja, muitos sistemas fundamentais que resolvem problemas críticos da humanidade ainda não usam totalmente a capacidade dos computadores modernos. Geralmente, isso é devido a natureza complexa do problema que muitas vezes dificulta a adaptação da solução para se utilizar mais intensamente uma infra-estrutura computacional moderna. Ainda, existe o fator financeiro que limita significativamente a substituição de *softwares* defasados, já que o custo de modernização pode ser proibitivo. Mas, existem situações em que as soluções clássicas precisam ser revistas e novas soluções, que ofereçam maior flexibilidade e usem de forma mais eficiente os recursos disponíveis, precisam ser desenvolvidas.

Nos últimos anos, além dos processadores terem se tornado mais rápidos, redes de computadores se tornaram comuns no mundo todo. Isso permitiu que a computação distribuída se desenvolvesse e suas técnicas fossem aplicadas na redução do tempo de execução dos mais diversos algoritmos (GEORGE; JEAN; TIM, 2005). Na computação distribuída, vários processadores (CPUs) são utilizados simultaneamente para se resolver uma tarefa maior ou garantir que um sistema funcione de forma robusta (resistindo à falhas de componentes do sistema). Dependendo do problema, pode-se obter ganhos de

[1] Mesmo dentro do *hardware* existe descompasso entre a evolução da CPU e da memória principal, como pode ser visto em Albrecht (2009), algo que obriga o desenvolvimento de técnicas de programação que reduzam o consumo de memória e demanda por acessos à memória principal, por exemplo. O conhecimento das limitações de *hardware* é fundamental para o desenvolvimento de metodologias de otimização de *software*.

desempenho diretamente proporcionais ao número de nós de processamento disponíveis no sistema. Mas, para que essas técnicas sejam aplicadas, é preciso que os algoritmos *sequenciais* [2] sejam transformados em um conjunto de tarefas que sejam despachadas para diferentes nós de processamento em uma rede, ou no mesmo *chip* [3], a *versão distribuída do algoritmo*. Essa distribuição não é trivial dependendo do grau de interdependência entre as diferentes partes de um algoritmo. Por exemplo, em muitas situações, para que um determinado cálculo seja efetuado corretamente, é preciso que uma série de passos seja executada em determinada ordem, algo que impossibilita o paralelismo. [4] [5]

Além da computação distribuída, outras técnicas para aceleração de algoritmos foram desenvolvidas, como:

- Programação *Multi-Thread*: essa técnica utiliza *threads*, que são linhas de execução que compartilham uma região de memória, o que facilita e acelera a comunicação de dados. Mas, por haver compartilhamento de recursos, isso pode criar regiões de concorrência, obrigando o desenvolvimento de rotinas de controle de *região crítica* [6].

- Programação usando placas gráficas (GPU) [7]: pacotes como CUDA (Arquitetura unificada de equipamentos de computação - *Compute Unified Device Architecture*) permitem que o programador utilize uma placa gráfica para realizar milhares de tarefas computacionais simultaneamente. GPUs são especialmente interessantes quando o algoritmo a ser paralelizado é baseado em muitas operações matriciais ou vetoriais.

- Programação funcional: embora não tenha sido criada com a computação distribuída em mente, essa técnica se casa muito bem com diversas metodologias para paralelização de algoritmos. Já que a representação de algoritmos via conjunto de pequenas funções e o conceito de delegação de tarefas reduzem bastante o esforço para se expressar um algoritmo paralelo. Afinal, uma forma elegante de visualizar um sistema distribuído é imaginar um conjunto de diferentes funções sendo executadas ao mesmo tempo, onde cada função pode ter seu próprio algoritmo, seus próprios dados de entrada e sua única obrigação é gerar uma saída que será integrada na solução final de um problema. Além disso, em teoria, programas funcionais não possuem efeitos

[2] Algoritmos pensados para serem executados em um único nó de processamento.
[3] Processadores modernos *multi-core* possuem mais de uma unidade de processamento na mesma pastilha, algo que reduz muito a latência da comunicação entre diferentes *threads*.
[4] Esse é um exemplo de *concorrência*, um dos principais entraves para o desenvolvimento eficiente de algoritmos distribuídos.
[5] Há uma anedota, em computação, que ilustra essa situação: "Nove mulheres grávidas ao mesmo tempo não parem um bebê em um mês."
[6] Nome dado a um trecho de código que manipula dados em uma região de memória que pode ser alterada por diferentes *threads*.
[7] Uma placa de vídeo moderna, como a *Titan X*, possui mais de 4.000 unidades de processamento e 6 GB de memória de trabalho (RAM). (JEONG et al., 2013)

colaterais, isto é, são unidades de execução isoladas que jamais alteram os dados de entrada e retornam saídas "enclausuradas" (objetos totalmente independentes). Existe uma estratégia de construção de sistemas distribuídos que se baseia fortemente em programação funcional para evitar a complexidade e ineficiência do controle de regiões críticas, é a *nada compartilhado - Shared Nothing* (STONEBRAKER, 1986). Nessa estratégia, cada processo é representado por uma função, que recebe uma cópia de todos recursos que precisa manipular, e gera um novo valor que é integrado pelo chamador na solução final.

- Programação concorrente: algumas etapas de um algoritmo não podem ser executadas simultaneamente por demandar resultados de tarefas anteriores (dependência inter-processos). Por isso, é preciso que algumas metodologias de sincronização, como *mutex* (exclusão mútua), semáforos, troca de mensagens, entre outras, sejam implementadas. Lidar com concorrência e sincronização é o maior desafio que aparece durante o desenvolvimento de qualquer sistema distribuído, já que esse problema está estritamente atrelado à natureza de cada algoritmo.

Obviamente, existem mais paradigmas que podem ser aplicados na elaboração de uma versão paralela de um determinado algoritmo, como, por exemplo, programação orientada a eventos (TILKOV; VINOSKI, 2010), programação descritiva (DREYBAND; NILVA; SHAPIRO, 2001), entre outros paradigmas, mas, para fins introdutórios, a lista anterior é suficiente.

Neste trabalho, diferentes algoritmos de geoestatística [8] são estudados e técnicas de computação distribuída são aplicadas na geração de versões mais eficientes de *softwares* fundamentais no dia-a-dia do profissional da área. Grande parte do esforço de desenvolvimento é concentrado na adaptação dos algoritmos de simulação, que são uma das tarefas mais computacionalmente intensivas no fluxo de atividades padrão da modelagem geoestatística. A geoestatística é revisitada no texto, sob o ponto de vista computacional, como, por exemplo, a simulação espectral, que será analisada sob a ótica da facilidade de paralelização e geração de *software* eficiente. Além disso, o custo de cada abordagem é analisado, medindo-se o impacto na qualidade dos resultados e a redução do tempo de execução.

Um dos objetivos desse trabalho, além da aplicação da computação distribuída à geoestatística, é aferir o poder de novas metodologias computacionais, como a *Simulação Espectral* e técnicas de otimização matemática e aprendizado de maquina. Estas técnicas oferecem muitas possibilidades de aprimoramento da qualidade dos resultados, redução

[8] Geoestatística é um ramo de estatística especializado no estudo de fenômenos espaciais ou espaço-temporais. Técnicas geoestatísticas fornecem modelos que podem ser usados na quantificação de reservas minerais, níveis de poluentes, predição de temperatura, entre outras aplicações em geologia, meteorologia e outras ciências.

da necessidade de intervenção humana e otimização computacional, como, por exemplo, aceleração de código via GPU (HUNGER et al., 2015) . Já que algoritmos de simulação espectral podem ser facilmente quebrados em sub-rotinas com baixo grau de acoplamento. Em especial, nesse trabalho, é verificado que o *Turning Bands* (EMERY; LANTUÉJOUL, 2006) e o *Fourier Integral Method* (PARDO-IGUZQUIZA; CHICA-OLMO, 1993) podem ser altamente paralelizados e ainda assim produzirem resultados de qualidade. Outro resultado interessante é a redução do tempo de modelagem que pode ser proporcionada pelas *tabelas de covariância* (uma estrutura de dados que pode tornar desnecessária a definição explicita de um modelo de continuidade espacial para simulação e estimativa) (YAO; JOURNEL, 1998) (MARCOTTE, 1996) [9]. Técnicas de otimização matemática e aprendizado de máquina são utilizadas na construção semi-automática de tabelas de covariância, ilustrando como a combinação de metodologias modernas de computação podem contribuir para reduzir o esforço humano necessário para realizar o tempo necessário para se construir modelos de covariância.

Nas seções e capítulos seguintes, o problema da tese é definido com maior precisão e uma revisão bibliográfica do estado da arte é realizada. Alguns resultados também são apresentados, mostrando o potencial das abordagens propostas. O produto final dessa tese é um conjunto de algoritmos distribuídos e uma metodologia baseada em otimização matemática e aprendizado de máquina para construção de modelos de covariância. Além disso, todos algoritmos apresentados estão implementados em uma nova plataforma na área de geoestatística, o AR2GAS (*Ar2tech Geostats As a Service*).

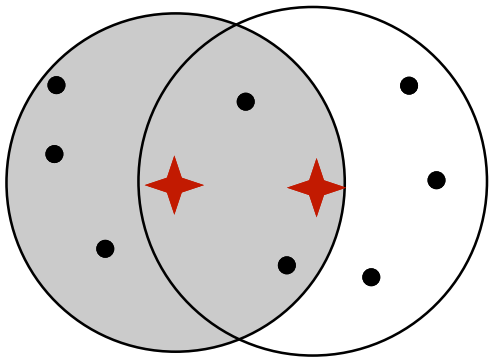
1.1 O problema

Na geoestatística, um dos problemas fundamentais é a simulação de modelos geológicos, uma tarefa computacionalmente intensiva que permite a quantificação das incertezas associadas aos teores e massas de um depósito mineral. Por isso, esforços estão sendo realizados para adaptar algoritmos clássicos para versões modernas levando em conta técnicas de programação paralela em CPUs e GPUs (TAHMASEBI et al., 2012), (MARIETHOZ, 2010), (PEREDO; ORTIZ, 2011). Essa tarefa não é trivial, haja vista que alguns algoritmos possuem características que dificultam significativamente a implementação de uma versão distribuída. Exemplos desses algoritmos são a *Simulação Sequencial Gaussiana* (SGSIM) (GOOVAERTS, 1997) (ISAAKS, 1991), *Simulação Sequencial de Indicadores* (SIS) (JOURNEL; ALABERT, 1989) (ISAAKS, 1984), *Simulação Sequencial Direta* (SOARES, 2001) e outros algoritmos de simulação sequencial baseados em MPS (estatística multi-ponto -*Multiple-Point Statistics*) como SNESIM (STREBELLE, 2002), IMPALA

[9] Essa estrutura pode ser computada de forma muito eficiente, utilizando-se FFT e GPU. Algo que é discutido com maiores detalhes nos próximos capítulos.

(STRAUBHAAR et al., 2011) e *Amostragem Direta* (*Direct Sampling*) (MARIETHOZ; RENARD; STRAUBHAAR, 2010).

Figura 1 – Zona de conflito entre duas vizinhanças. As estrelas vermelhas representam dois pontos que são simulados simultaneamente e os pontos pretos representam os dados de condicionamento.



Nos algoritmos de simulação sequencial, existem situações onde há sobreposição entre as vizinhanças (situação descrita na Fig. 1). Nesses casos, o caminho sequencial aleatório obriga que os estágios de condicionamento e geração de valores usando-se a vizinhança de busca de uma simulação sejam realizados de forma sequencial - ou seja, um passo após o outro. Duas vizinhanças são consideradas sobrepostas, quando na interseção entre as vizinhanças existem nós a serem simulados gerando um problema de sincronização, já que isso cria uma interdependência de resultados. Para lidar com esse problema, o desenvolvedor de uma versão paralela desses algoritmos precisa adotar alguma das diferentes estratégias descritas a seguir (MARIETHOZ, 2010):

- Paralelizar os cálculos após as matrizes de krigagem (MATHERON, 1963) já estarem construídas, obrigando a adição de zonas de exclusão mútua no código (*mutexes*). Na literatura, essa abordagem é chamada de paralelização no nível do nó. (NUNES; ALMEIDA, 2010) Essa estratégia permite a reprodutibilidade de resultados, já que o caminho aleatório é obedecido.

- Analisar o caminho sequencial aleatório e verificar quais são os próximos N nós que não compartilham dados de vizinhança (vizinhanças independentes entre si). Esses nós podem ser simulados paralelamente sem ser necessária a criação de uma zona de exclusão na região em que as matrizes de krigagem são criadas. Essa técnica é conhecida como paralelização no nível do caminho (VARGAS; CAETANO; FILIPE, 2007). As simulações nessa estratégia também podem ser reproduzidas, já que a etapa de pré-processamento, antes da simulação de cada nó, garante que nunca dois nós a serem simulados estejam presentes em vizinhanças de buscas conflitantes.

- Resolução de conflitos, via sincronização. Nesse caso, os nós seguintes do caminho aleatório são enviados para os nós de processamento e os processos trocam mensagens

entre si para verificar se há interseções entre suas vizinhanças de busca. Como os conflitos entre vizinhanças podem ocorrer de forma aleatória, não é possível reproduzir futuramente as mesmas simulações. (MARIETHOZ, 2010)

- Não utilizar um caminho totalmente aleatório, adotando-se uma estratégia para dividir o espaço em grupos maiores independentes. E, dentro de cada grupo, caminhos sequenciais aleatórios são gerados. O paralelismo é implementado simulando-se nós de cada grupo de forma independente. (RASERA; MACHADO; COSTA, 2015)
- A solução trivial é paralelizar no nível das realizações, onde cada realização é processada em um processador ou *thread* diferente e, no final, os resultados são enviados para um processador central.

Cada uma dessas estratégias possui vantagens e desvantagens, principalmente quanto à facilidade de implementação e consumo de recursos. A solução trivial (paralelização no nível das realizações) é a que mais demanda recursos, já que cada nó de processamento precisa ter memória suficiente para manter uma cópia da realização sendo executada e o consumo de rede no final de cada realização pode ser proibitivo. Além disso, essa solução não é útil no estágio de preparação de parâmetros, pois a execução de cada realização continuará sequencial. Todas as outras estratégias se focam na redução do tempo de execução de cada realização. Algo que permite tanto a redução do tempo de preparação de parâmetros quanto possibilita que estruturas de *grids* bem maiores sejam simuladas. Obviamente, essas estratégias podem ser combinadas para se obter uma solução que melhor se adeque a uma determinada infraestrutura computacional.

Analisando o caso do SGSIM e das simulações sequenciais *clássicas*, nota-se que a paralelização eficiente desses algoritmos demanda uma grande reestruturação do modo que foram inicialmente pensados. Com exceção da solução trivial (paralelizar cada realização em nós diferentes), todas as outras estratégias de paralelismo acabam resultando em uma versão mais complexa do algoritmo de simulação. Sem falar que, em alguns casos, a etapa de sincronismo pode reduzir a eficiência do paralelismo. Uma questão que surge naturalmente, então, é: "é possível paralelizar a realização de uma simulação sem aumentar a complexidade do algoritmo resultante e garantir grande eficiência?"

Quando se paraleliza a krigagem, por exemplo, percebe-se o quanto o fato de se ter nós independentes facilita e garante a geração de uma versão paralela eficiente desse algoritmo de estimativa - já que isso permite a utilização de GPUs para se acelerar os cálculos (RAVÉ et al., 2014), por exemplo. Afinal, na krigagem para estimar cada nó só é necessário a montagem de uma matriz com dados fixos de condicionamento e a resolução de um sistema linear. Essa tarefa é independente e pode ser distribuída para tantos processadores quanto forem necessários (no caso extremo, cada nó da malha de estimativa pode ser estimado em um processador diferente). Em computação distribuída, quando

se tem um problema onde o paralelismo é simples e imediato e não demanda qualquer tratamento de sincronização, esse problema é chamado de *embaraçosamente paralelo* (WILKINSON; ALLEN, 1999). Essa é uma classe especial de problemas porque qualquer técnica de paralelismo pode ser aplicado com eficiência máxima nos membros dessa classe. A krigagem é um importante exemplo de problema dessa classe. No SGeMS, por exemplo, esse algoritmo é facilmente paralelizado utilizando-se OpenMP (*Open Multi-Processing* ou multi-processamento aberto) (DAGUM; MENON, 1998).

A facilidade da paralelização da krigagem e outros algoritmos de estimativa é um importante motivador. Será que é possível encontrar modos de se paralelizar a simulação de modo que seja possível obter algoritmos embaraçosamente paralelos?

Os algoritmos de simulação sequencial são obviamente não-triviais (no sentido de computação distribuída), já que demandam complexas adaptações. Mas, na literatura, ao longo das últimas décadas, surgiu uma nova classe de algoritmos de simulação, os *Métodos Espectrais* - cujos principais representantes são o *Turning Bands* (Bandas rotativas) (EMERY; LANTUÉJOUL, 2006) e o *FIM* (*Fourier Integral Method* - Método Integral de Fourier) (PARDO-IGUZQUIZA; CHICA-OLMO, 1993). Essa classe de algoritmos possui a característica comum de transformar o problema de simulação em um conjunto de diversas tarefas simples e independentes.

No caso do FIM, por exemplo, após os dados serem transformados via transformada de Fourier e a densidade espectral s ser construída, a simulação se transforma no problema de se gerar números aleatórios entre $-\pi$ e π e se computar o valor simulado no domínio espectral. No domínio espectral, o acoplamento espacial entre os dados desaparece e cada nó pode ser simulado de forma extremamente paralela sem qualquer rotina de sincronização. O condicionamento é feito utilizando-se krigagem dos resíduos. Em outras palavras, o FIM é um algoritmo embaraçosamente paralelo, já que não há mais qualquer necessidade de sincronização durante a realização da simulação (principal fator que dificulta a paralelização dos algoritmos de simulação sequencial).

Os métodos espectrais são a melhor alternativa para se construir sistemas de simulação mais eficientes. Em alguns casos, os resultados geoestatísticos podem ser até melhores que os obtidos por algoritmos clássicos.

O problema da simulação é um exemplo de como a computação distribuída pode demandar a criação ou adoção de novas técnicas. Afinal, além da qualidade dos resultados, um nova característica observada passa a ser a escalabilidade [10] do algoritmo. Nesse trabalho, métodos espectrais são estudados e são mostrados resultados que ilustram a eficiência e qualidade dessas técnicas.

Outro problema chave explorado nesse trabalho é a criação semi-automática de

[10] Facilidade e/ou possibilidade de se reduzir o tempo de execução quando mais unidades de processamento são adicionadas à infraestrutura de processamento.

modelos de covariância, que são o elemento fundamental de qualquer algoritmo geoestatístico. Nesse trabalho, o meta-algoritmo *busca tabu* (GLOVER, 1986) é combinado com a técnica de *ensemble simulation* (KLOECKNER et al., 2019) [11] para se construir um algoritmo iterativo que incorpore erros de validação cruzada para evoluir uma solução inicial, possivelmente não satisfatória. A metodologia pode ser aplicada para incorporação de características locais de um banco de dados, permitindo a incorporação de informações que não foram adequadamente capturadas pela modelagem linear tradicional. Essa técnica pode ser vista como uma generalização natural do processo de simulação sequencial, no caso, substituindo a variância de krigagem pelo erro empírico de validação cruzada (que pode ser visto como um bom aproximador prático da variância local real dos dados (ASMUSSEN; GLYNN, 2011)).

Para se construir uma metodologia semi-automática de incorporação de informação, o meta-algoritmo *busca tabu* (GLOVER, 1986) mostra-se uma alternativa interessante, concorrendo com técnicas como algoritmos genéticos ou enxame de partículas. Nesse algoritmo, a criação de "filhos" (vizinhos na terminologia do algoritmo) é definida de acordo com o problema estudado, podendo inclusive utilizar o candidato "pai" como semente pra um processo iterativo de otimização local como o clássico método de Newton, BFGS (algoritmo de Broyden–Fletcher–Goldfarb–Shanno) (BROYDEN, 1970) entre outros algoritmos baseados em alguma noção de gradiente. O objetivo da "busca tabu", e de algoritmos não baseados em gradiente, é aumentar a robustez do processo de otimização, reduzindo o impacto de mínimos locais, problema que afeta significativamente metodologias clássicas. Além disso, por ser um meta-algoritmo, isto é, uma estrutura algorítmica de alto nível que não especifica detalhes de seus passos, a *busca tabu* fornece um estrutura excelente para construção de *workflows* baseados em combinação de diferentes estratégias. No problema estudado nesta tese, o processo de criação de vizinhos, realizado durante a busca, é implementado utilizando um misto de técnicas baseadas em *simulated annealing* e técnicas baseadas em métodos tradicionais de otimização numérica. Nesses processos de otimização, a validação cruzada contribui na construção de uma métrica de erro empírica utilizada tanto na melhoria local do modelo de covariância quanto na definição da variância local usada no processo de *ensemble simulation*.

Processos de otimização fornecem metodologias objetivas para aprimorar modelos empíricos ou para criar candidatos a solução que podem ser filtrados pelo modelador, auxiliando no processo de tomada de decisão e modelagem. É importante sempre salientar que essas metodologias não devem ser vistas como panaceia, o modelador ainda exerce um papel fundamental escolhendo modelos que melhor aderem ao contexto do problema

[11] *Ensemble estimation* é um algoritmo de estimativa que combina um conjunto de estimadores base usando o erro local associado a cada estimador para ponderar sua contribuição na estimativa final. No caso do estimador fornecer uma estimativa para a variância da estimação (variância de krigagem, por exemplo), pode-se construir o *ensemble simulation*, que também usa os erros locais de cada estimador para ponderar a variância no nó a ser simulado.

estudado. Quando bem aplicado, os processos de otimização aumentam significativamente o poder de análise do modelador. Todo processo de modelagem semi-automática é um processo evolutivo, necessitando contínua retro-alimentação para constante evolução dos modelos. Em linhas gerais, podemos ver esse processo como uma complementação das técnicas lineares clássicas de estimativa (como a krigagem), permitindo uma solução aproximada de modelos não-lineares de grande complexidade. Fugindo do escopo específico desse trabalho, essa metodologias podem ser adaptadas para lidar com modelos não-estacionários, via otimização local de modelos de covariância incorporando tanto o problema de minimização de erro global, quanto o problema de reprodução de variâncias locais.

Em suma, o problema estudado neste trabalho é: "Como utilizar otimização matemática, aprendizado de máquina e computação de alta performance para se construir implementações mais robustas e eficientes de algoritmos fundamentais de geoestatística e que reduzam a necessidade de intervenção humana"

1.2 Meta e objetivos

Tendo em vista a evolução recente na capacidade de processamento dos computadores modernos e o surgimento de metodologias eficientes para otimização computacional, algumas perguntas que naturalmente surgem:

- Técnicas de otimização podem contribuir para reduzir a necessidade de intervenção humana no processo de modelagem geoestatística?
- Computação de alta performance permite a ampliação da capacidade de resolução de problemas das técnicas clássicas de modelagem geoestatística?
- Como o aprendizado de máquina pode contribuir para a melhoria dos processos de modelagem clássicos? Como utilizar essas técnicas para informações não-triviais contidas nos dados que não são adequadamente capturadas pelos métodos estacionários clássicos?

A principal meta desse trabalho é investigar essas questões e, a partir das respostas obtidas, construir uma nova geração de *software* para geoestatística que utilize tanto técnicas de otimização matemática, aprendizado de máquina, quanto os recursos disponíveis em uma infraestrutura computacional moderna como *clusters* de computadores e processadores *multi-core*. Em outras palavras, neste trabalho, é investigada e proposta a utilização de HPC (*High Performance Computing* - Computação de Alto Desempenho) e de *Machine Learning* na construção de técnicas de modelagem, estimativa e simulação estado da arte. O objetivo fundamental é combinar a evolução da computação nas últimas décadas com metodologias estabelecidas tanto em otimização matemática quanto em

geoestatística para se construir ferramentas que reduzam a necessidade de intervenção humana, reduzam o tempo de execução dos processos de modelagem clássicos e incorporem aos modelos gerados mais detalhes contidos nos dados.

Para atingir essa meta, é preciso revisitar alguns dos principais algoritmos utilizados na geoestatística e analisar os desafios específicos de cada um. Além disso, os algoritmos são implementados na plataforma *AR2GAS*, permitindo que usuários comuns possam utilizar as técnicas facilmente. Os algoritmos a seguir são estudados:

- Geração de tabelas de covariância ou mapa de covariância;
- Simulação Sequencial Gaussiana;
- Método Integral de Fourier (FIM - *Fourier Integral Method*);
- Simulação por *Turning Bands*;
- Computação eficiente de variogramas, correlogramas, entre outras estatísticas, usando-se FFT -*Fast Fourier Transform* ou Transformada Rápida de Fourier - (COOLEY; TUKEY, 1965), GPU e *Multi-thread* (MARCOTTE, 1996);
- *Ensemble simulation* para construção de modelos base de covariância combinado com *busca tabu* para aprimoramento iterativo do estimador.

Enfim, o fluxo de atividade típico da geoestatística é revisitado. No caso, construindo-se novas versões desses algoritmos, reduzindo o tempo de execução e a complexidade de execução dessas tarefas. É importante lembrar que algumas técnicas como a FIM não demandam modelos de covariância explícitos e a tabelas de covariância podem ser usadas como modelos em técnicas de estimativa e simulação clássicas como SGSIM e krigagem.

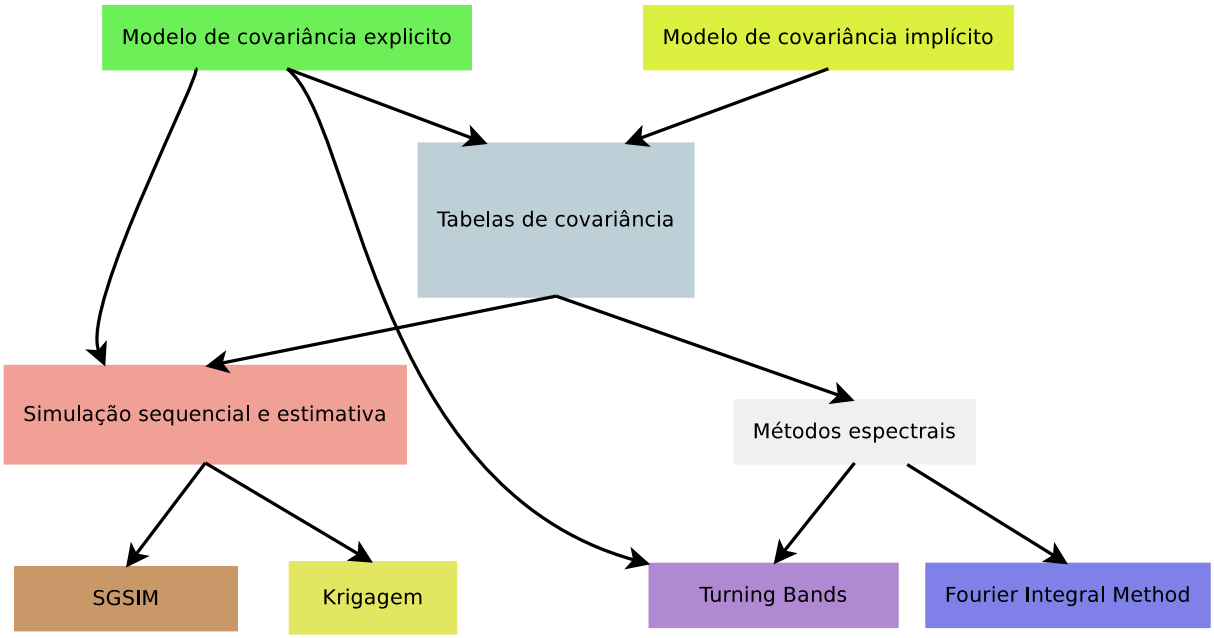
1.3 Metodologia

Uma nova plataforma de *software* conhecida, o *AR2GAS*, é utilizada como base para o desenvolvimento de versões de alta performance de algoritmos de geoestatística tradicionais e novos.

Como não há solução universal em computação distribuída, alguns algoritmos foram alterados ou reescritos (como é o caso do SGSIM e TBSIM), ou ainda foram implementados a partir de sua definição (como foi o caso do FIM).

Na Fig. 2, é mostrado o relacionamento lógico entre os temas que serão explorados nesse trabalho. No conjunto de *softwares* desenvolvidos, as tabelas de covariâncias exercem um papel fundamental na construção de uma metodologia ágil de modelagem, já que elas abstraem os detalhes do método utilizado para se construir o modelo de covariância. Isso

Figura 2 – Inter-dependência entre os temas abordados neste trabalho.



permite que diferentes tipos de métodos implícitos ou explícitos possam ser utilizados para se construir essas tabelas. Muitos métodos de construção de modelos de covariância são extremamente paralelizáveis. Os métodos espectrais são outro componente fundamental, já que eles permitem que técnicas de paralelização sejam plenamente aplicadas sem a necessidade de criação de estratégias de sincronização. Pode-se dizer que os métodos espectrais, do ponto de vista computacional, são uma técnica para transformar o problema de simulação geoestatística em um problema *embaraçosamente paralelo* [12]. O processo de otimização e identificação de modelo covariância usando *busca tabu* combinado com *ensemble simulation* é usado na construção de modelos implícitos.

1.4 Contribuições da tese

Este trabalho produziu:

- novas versões HPC de diversos algoritmos fundamentais em geoestatística;
- novas metodologias para geração de tabelas de covariância e densidade espectral;
- nova técnica de modelagem evolutiva que permite o aprimoramento de soluções tradicionais por meio da incorporação de informação local via validação cruzada.

[12] Curiosamente, os métodos espectrais se fundamentam em transformações dos dados, de forma que o problema de simulação se transforme em um conjunto de problemas independentes no espaço, cujas soluções são combinados posteriormente para se construir a solução final.

1.5 Organização da tese

Este trabalho é dividido em 5 capítulos:

- O Capítulo 1 é a introdução, onde é mostrado o problema proposto, as metas e objetivos, a metodologia e a contribuição da tese;
- O Capítulo 2 é dedicado a uma apresentação sucinta da transformada de Fourier, do algoritmo de Transformada Rápida de Fourier e mostra como essa técnica é aplicada na aceleração da computação de covariâncias;
- O Capítulo 3 apresenta um algoritmo de modelagem semi-automática de tabelas de covariância;
- O Capítulo 4 apresenta dois algoritmos de simulação espectral que serão implementados no pacote de simulação espectral HPC;
- O Capítulo 5 apresenta técnicas de modelagem geoestatística usando aprendizado de máquina;
- O Capítulo 6 apresenta as conclusões desse trabalho e trabalhos futuros.

2 Transformada de Fourier

Em (FIGUEIREDO, 2000) é apresentado um interessante problema matemático, que, transcrito de forma simplificada, pode ser apresentado como:

Considere-se a equação diferencial, a seguir:

$$\frac{d^2}{dt^2}y + 2\frac{d}{dt}y + y = \sin(t). \tag{2.1}$$

Agora, supõe-se que é possível a realização das seguinte operações:

- Seja $D \times y = \frac{d}{dt}y$ e $D^2 \times y = \frac{d^2}{dt^2}y$;
- Aplicando-se o "operador" D na eq. (2.1), obtém-se:

$$\begin{aligned} D^2y + 2Dy + y &= sin(t) \\ y(D^2 + 2D + 1) &= sin(t) \\ y = \frac{sin(t)}{D^2 + 2D + 1}&(???). \end{aligned} \tag{2.2}$$

Esse absurdo matemático simplificaria muito o problema de resolução de equações diferenciais, não? Afinal, o problema de se resolver uma equação como a (2.1) seria reduzido ao problema de se resolver uma equação algébrica, que é muito mais fácil.

Retornando-se ao mundo real, embora os passos anteriores sejam completamente *nonsense* segundo a teoria matemática, eles motivam uma interessante questão: "Existe alguma ferramenta matemática que atuaria como um *portal* entre dois mundos simplificando significativamente problemas difíceis, como a resolução de equações diferenciais?".

Essa ferramenta existe, embora não seja exatamente tão simples como os passos absurdos que foram apresentados no começo do texto. Trata-se da *Transformada de Fourier*. Conforme é visto a seguir, utilizando-se essa transformada, equações diferenciais são "algebrizadas" e operações complexas, como a convolução entre duas funções [1], podem ser mais eficientemente computadas.

A transformada de Fourier, essencialmente, é uma mudança de variáveis mais sofisticada que altera a natureza das operações que precisam ser executadas, de forma que a capacidade de computação de quem resolve um problema é amplificada. Mais que

[1] A convolução entre duas funções f e g é definida como $f * g = \int_{-\infty}^{+\infty} f(\lambda)g(t-\lambda)d\lambda$.

isso, por meio da transformada de Fourier, é possível desenvolver métodos automáticos, mecânicos e *brainless* para se resolver problemas que exigiriam grande habilidade e domínio de técnicas mais complexas.

Devido a sua importância, na definição do *Método Integral de Fourier* e na construção eficiente de variogramas, tabelas de covariância e outras estruturas, esse capítulo é dedicado à apresentação de alguns resultados fundamentais sobre Transformada de Fourier.

Figura 3 – Jean-Baptiste Joseph Fourier.

2.1 A série de Fourier

Jean-Baptiste Joseph Fourier (FOURIER, 1822) (visto na Fig. 3) demonstrou que muitas funções contínuas e periódicas $f(t)$ (com período $2T$) podem ser escritas como uma soma de funções trigonométricas. Essa soma é chamada de *Série de Fourier* (2.3) (FIGUEIREDO, 2000), com termos a_k e b_k.

$$\begin{aligned} f(t) &= \frac{a_0}{2} + \sum_{k=1}^{+\infty} a_k \cos(\frac{2k\pi t}{T}) + b_k \sin(\frac{2k\pi t}{T}), \\ a_k &= \frac{2}{T}\int_{t_0}^{t_0+T} f(t)\cos(\frac{2k\pi t}{T})dt, k \geq 0 \\ b_k &= \frac{2}{T}\int_{t_0}^{t_0+T} f(t)\sin(\frac{2k\pi t}{T})dt, k \geq 1 \end{aligned} \tag{2.3}$$

A eq. (2.3) pode ser simplificada utilizando-se a identidade de Euler $e^{i\theta} = \cos(\theta) +$

$i\sin(\theta)$ e introduzindo-se um novo termo c_k.

$$\begin{aligned} c_k &= \frac{a_k - ib_k}{2} \\ c_k &= \frac{1}{T}\int_{t_0}^{t_0+T}[f(t)\cos(\frac{2\pi k}{T}) - if(t)\sin(\frac{2\pi k}{T})]dt \\ c_k &= \frac{1}{T}\int_{t_0}^{t_0+T}[f(t)\cos(\frac{-2\pi k}{T}) + if(t)\sin(\frac{-2\pi k}{T})]dt \\ c_k &= \frac{1}{T}\int_{t_0}^{t_0+T} f(t)e^{\frac{-i2\pi k}{T}}dt \\ f(t) &= \sum_{k=-T}^{T} c_k e^{\frac{i2\pi kt}{T}} \end{aligned} \tag{2.4}$$

A eq. (2.4) é a forma complexa da série de Fourier, cujas propriedades são apresentadas na subseção a seguir.

2.1.1 Propriedades da Série de Fourier

Nessa subseção, algumas propriedades fundamentais das séries de Fourier são apresentadas, onde $x(t)$ e $y(t)$ são funções contínuas periódicas, com período T, frequência fundamental $w_0 = 2\pi/T$ e coeficientes de Fourier a_k e b_k, respectivamente:

1. Linearidade: A combinação linear $z(t) = Ax(t) + By(t)$ de $x(t)$ e $y(t)$ gera uma série de Fourier cujos coeficientes c_k são iguais a $c_k = Aa_k + Bb_k$.

2. Deslocamento temporal: A série de Fourier de $x(t - t_0)$ possui coeficientes $c_k = a_k e^{-jkw_0t_0}$.

3. Deslocamento de frequência: Os coeficientes do sinal $e^{jMw_0t}x(t)$ são dados por $c_k = a_{k-M}$.

4. Conjugação: Os coeficientes de $x^*(t)$ são dados por $c_k = a^*_{-k}$.

5. Inversão temporal: Os coeficientes de $x(-t)$ são $c_k = a_{-k}$.

6. Escalonamento temporal: Os coeficientes do sinal periódico $x(\alpha t)$, com $\alpha > 0$ e período T/α, são $c_k = a_k$.

7. Convolução periódica: Os coeficientes de $z(t) = \int_0^T x(\tau)y(t-\tau)d\tau$ são $c_k = Ta_kb_k$

8. Multiplicação: Os coeficientes de $z(t) = x(t)y(t)$ são $c_k = \sum_{l=-\infty}^{+\infty} a_l b_{k-l}$

9. Diferenciação: Os coeficientes de $\frac{dx(t)}{dt}$ são $c_k = jkw_0a_k$.

10. Integração: Os coeficientes de $z(t) = \int_{-\infty}^{t} x(\tau)d\tau$ (com valor finito e periódica só se $a_0 = 0$) são $c_k = \frac{1}{jkw_0}a_k$.

11. Conjugado simétrico de funções reais: Os coeficientes de uma função real $z(t) = x(t)$ são $c_k = a^*_{-k}$.

12. Coeficientes de funções reais e pares: Os coeficientes de uma função real e par $x(t)$ são reais e pares, ou seja, $a_k = a_{-k}$.

13. Coeficientes de funções reais e impares: Os coeficientes de uma função real e impar [2] $x(t)$ são puramente imaginários e impares, ou seja, $a_k = -a_{-k}$.

14. Decomposição em termos pares e impares de um função real: Os coeficientes das funções $x_p(t) = \mathbb{PAR}[x(t)]$ e $x_i(t) = \mathbb{IMPAR}[x(t)]$ de uma função $x(t)$ real são $ap_k = \text{Real}[a_k]$ e $ai_k = j\text{Im}[a_k]$. Ou seja, os coeficientes de Fourier de uma função real $x(t)$ podem ser usados para decompor-la em suas partes par e impar.

15. Identidade de Parseval de funções periódicas: $\frac{1}{T}\int_0^T |x(t)|^2 dt = \sum_{k=-\infty}^{+\infty} |a_k|^2$

A identidade de Parseval também pode ser vista como o principio da conservação de energia. No desenvolvimento do algoritmo de simulação espectral usando-se transformada de Fourier, as propriedades mais utilizadas são o Teorema da Convolução (item 7) e as propriedades das funções reais.

Além disso, pode-se considerar as propriedades da transformada de Fourier e de sua versão discreta como extensões das propriedades apresentadas nessa subseção.

2.2 Construindo a Transformada de Fourier

Seja $\xi_k = 2k\pi/T$, $h = \pi/T$ e

$$\begin{aligned} f(t) &\approx \sum_{k=-\infty}^{+\infty} c_k e^{i\xi_k t} \\ c_k &= \frac{1}{T}\int_{-T/2}^{T/2} f(t)e^{-i\xi_k t}dt. \end{aligned} \tag{2.5}$$

Introduzindo-se a expressão

$$c(\xi_k) = \frac{1}{\sqrt{2\pi}}\int_{-T/2}^{T/2} f(t)e^{-i\xi_k t}dt, \tag{2.6}$$

[2] Um função $x(t)$ é par, caso a igualdade $x(-t) = x(t)$ seja satisfeita. E, $x(t)$ é impar, se $x(-t) = -x(t)$.

pode-se escrever:

$$f(t) \approx \frac{1}{\sqrt{2\pi}} \sum_{k=-\infty}^{+\infty} h c_k e^{i\xi_k t}. \tag{2.7}$$

Ao fazer $T \to +\infty$, temos que $h \to 0$ e a eq. (2.7) adquire o aspecto formal de uma soma de Riemann. Isto é, a eq. (2.7) converge para a integral:

$$\begin{aligned} f(t) &\approx \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} c(\xi) e^{i\xi t} d\xi \\ c(\xi) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(t) e^{-i\xi t} dt \end{aligned} \tag{2.8}$$

O par de equações (2.8) são, respectivamente, a *Transformada de Fourier* e a *Transformada Inversa de Fourier*.

A Transformada de Fourier é uma ferramenta fundamental de cálculo, com diversas propriedades que permitem a simplificação de problemas difíceis como a resolução de equações diferenciais ordinárias [3].

Por exemplo, seja a equação:

$$y'' + 2\alpha y' + \alpha^2 y = x(t), \tag{2.9}$$

onde $\alpha \in \mathbb{C}$ e $x(t)$ é uma função que possui transformada de Fourier.

Uma propriedade muito importante da transformada de Fourier diz que:

$$\begin{aligned} &\mathscr{F}\left[f^{(n)}(t)\right] = (wi)^n F(\xi) \\ &w = 2\pi\xi \\ &F(\xi) = \mathscr{F}\left[f(t)\right], \end{aligned} \tag{2.10}$$

onde $F(\xi)$ é usado para representar a transformada da função $f(t)$ e $\mathscr{F}[\cdot]$ é o operador transformada de Fourier.

Logo, a eq. (2.9) pode ser transformada na seguinte equação algébrica no espaço de Fourier:

$$-w^2 Y(\xi) + 2\alpha w i Y(\xi) + \alpha^2 Y(\xi) = X(\xi). \tag{2.11}$$

[3] Trata-se de uma forma "correta" de se realizar os passos absurdos mostrados na introdução desse capítulo.

Resolvendo-se a equação algébrica (2.11):

$$\begin{aligned} &-w^2Y(\xi) + 2\alpha wiY(\xi) + \alpha^2 Y(\xi) = X(\xi), \\ &Y(\xi) = \frac{X(\xi)}{\alpha^2 + 2\alpha wi - w^2}. \end{aligned} \tag{2.12}$$

Antes de finalizar a solução da eq. (2.12), é necessária a apresentação de outra propriedade importante das Transformadas de Fourier, o *Teorema da Convolução*:

$$\mathscr{F}\left[f(t) * g(t)\right] = \mathscr{F}\left[\int_{-\infty}^{\infty} f(\lambda)g(t-\lambda)d\lambda\right] = F(\xi)G(\xi) \tag{2.13}$$

O Teorema da Convolução permite que a eq. (2.12) seja escrita como uma convolução:

$$\begin{aligned} &\mathscr{F}^{-1}\left[Y(\xi)\right] = \mathscr{F}^{-1}\left[X(\xi)\right] * \mathscr{F}^{-1}\left[\frac{1}{\alpha^2 + 2\alpha wi - w^2}\right] \\ &y(t) = \mathscr{F}^{-1}\left[Y(\xi)\right] \\ &x(t) = \mathscr{F}^{-1}\left[X(\xi)\right] \\ &h(t) = \mathscr{F}^{-1}\left[\frac{1}{\alpha^2 + 2\alpha wi - w^2}\right] = u(t)te^{-\alpha t} \\ &y(t) = h(t) * x(t) \\ &y(t) = [u(t)te^{-\alpha t}] * x(t) \\ &y(t) = \int_0^{+\infty} \lambda e^{-\alpha\lambda} x(t-\lambda)d\lambda \end{aligned} \tag{2.14}$$

A função $u(t)$, na eq. (2.14), é definida como:

$$u(t) = \begin{cases} 1, & \text{se } t \geq 0 \\ 0, & \text{outros valores} \end{cases}. \tag{2.15}$$

A função $h(t) = u(t)t$ é a *Função de Transferência* do sistema, ela transforma o problema de resolução de uma equação diferencial em um problema de resolução de uma integral de convolução. Essa integral de convolução pode ser resolvida utilizando-se o Teorema de Convolução.

É importante lembrar que as definições apresentadas podem ser extendidas para o caso multi-dimensional. No caso $\mathbb{C}^n$, a transformada de Fourier é definida como:

$$\begin{aligned} f(t) &= \int_{\mathbb{C}^n} c(\xi)e^{i<\xi,t>}dt \\ c(\xi) &= \int_{\mathbb{C}^n} f(t)e^{-i<\xi,t>}dt \\ t = (t_1, t_2, ..., t_n), \xi &= (\xi_1, \xi_2, ..., \xi_n), < a, b >= \textstyle\sum_{k=1}^{n} a_k b_k \end{aligned} \tag{2.16}$$

Para o entendimento deste trabalho, os conceitos mais importantes são o Teorema da Convolução e a versão discreta da transformada de Fourier, que é computada utilizando-

se a Transformada Rápida de Fourier (*Fast Fourier Transform* - FFT) (COOLEY; TUKEY, 1965) (RADER, 1968).

Existem muitas propriedades da Transformada de Fourier que permitem a simplificação e aceleração de cálculos, conforme pode ser visto em (BOASHASH, 2003). A seguir, são mostrados alguns resultados fundamentais, que serão usados durante a apresentação do Método Integral de Fourier e do algoritmo para geração eficiente de variogramas, tabelas de covariância e outras estatísticas.

2.2.1 Propriedades da Transformada de Fourier

A seguir, as propriedades de funções aperiódicas [4] $x(t)$ e $y(t)$, com transformadas de Fourier $X(jw)$ e $Y(jw)$, respectivamente, são apresentadas:

1. Linearidade do operador: A transformada de $z(t) = Ax(t) + By(t)$ é dada por $Z(jw) = AX(jw) + BY(jw)$.
2. Deslocamento temporal: A transformada de $z(t) = x(t - t_0)$ é dada por $Z(jw) = e^{-jwt_0}$X(jw).
3. Deslocamento em frequência: A transformada de $z(t) = e^{jw_0t}x(t)$ é dada por $Z(jw) = X(j(w - w_0))$.
4. Conjugação: A transformada de $z(t) = x^*(t)$ é $Z(jw) = X^*(-jw)$.
5. Reversão temporal: A transformada de $z(t) = x(-t)$ é $Z(jw) = X(-jw)$.
6. Escalonamento na frequência e no tempo: A transformada de $z(t) = x(\alpha t)$ é $Z(jw) = \frac{1}{|\alpha|}X(\frac{jw}{\alpha})$.
7. Convolução: A transformada de $z(t) = x(t) * y(t)$ é $Z(jw) = X(jw)Y(jw)$.
8. Multiplicação: A transformada de $z(t) = x(t)y(t)$ é $Z(jw) = \frac{1}{2\pi}X(jw) * Y(jw)$.
9. Diferenciação: A transformada de $z(t) = \frac{dx(t)}{dt}$ é $Z(jw) = jwX(jw)$.
10. Integração: A transformada de $z(t) = \int_{-\infty}^{t} x(\tau)d\tau$ é $Z(jw) = \pi X(0)\delta(w) + \frac{1}{jw}X(jw)$.
11. Diferenciação em frequência: A transformada de $z(t) = tx(t)$ é $Z(jw) = j\frac{dX(jw)}{dw}$.
12. Conjugado simétrico de uma função real: $X(jw) = X^*(-jw)$.

[4] É importante ressaltar que uma função aperiódica pode ser vista como uma função $f(t)$, com período $T \to +\infty$.

13. Simetria de uma função real e par: A transformada de uma função real e par $x(t)$ também é real e par.

14. Simetria de uma função real e impar: A transformada de uma função real e impar $x(t)$ é puramente imaginária e impar.

15. Decomposição em componentes pares e ímpares de uma função real: As transformada de Fourier das funções $x_p(t) = \mathbb{PAR}[x(t)]$ e $x_i(t) = \mathbb{IMPAR}[x(t)]$ de uma função $x(t)$ real são $X_p(jw) = \text{Real}[X(jw)]$ e $X_i(jw) = \text{Im}[X(jw)]$.

16. Identidade de Parseval: $\int_{-\infty}^{+\infty} |x(t)|^2 dt = \frac{1}{2\pi} \int_{-\infty}^{+\infty} |X(jw)|^2 dw$

Como pode-se notar, as propriedades da transformada de Fourier são versões assintóticas das propriedades das séries de Fourier. Além disso, as propriedades das transformadas das funções reais são fundamentais quando se trabalha com densidade espectral, já que a função de covariância é uma função real par. Usando-se as propriedades definidas anteriormente, é imediato concluir que a densidade espectral (transformada de Fourier da função covariância) também será uma função real e par. Esse fato é explorado no algoritmo de modelagem implícita que é apresentado no Cap. 3.

Da mesma forma, as propriedades da transformada de Fourier podem ser estendidas para a sua versão discreta, conforme é visto na seção a seguir.

2.3 A Transformada Discreta de Fourier

Em aplicações práticas, a versão discreta da transformada de Fourier (Transformada Discreta de Fourier - *Discrete Fourier Transform* - DFT) é mais utilizada. A transformada discreta de Fourier, em $\mathbb{C}^1$, é definida da seguinte forma:

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi k n i}{N}} \\ x_n &= \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi k n i}{N}} \end{aligned} \tag{2.17}$$

Da mesma forma que na versão contínua, a definição dada pela eq. (2.17) pode ser extendida para $\mathbb{C}^d$, o caso d-dimensional.

$$\begin{aligned} X_k &= \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \ldots \sum_{n_d=0}^{N_d-1} x_n e^{\frac{-2\pi k_1 n_1 i}{N_1} - \frac{2\pi k_2 n_2 i}{N_2} - \ldots - \frac{2\pi k_d n_d i}{N_d}} \\ x_n &= \frac{1}{N_1 N_2 \ldots N_d} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \ldots \sum_{k_d=0}^{N_d-1} X_k e^{\frac{2\pi k_1 n_1 i}{N_1} + \frac{2\pi k_2 n_2 i}{N_2} - \ldots + \frac{2\pi k_d n_d i}{N_d}} \\ k &= (k_1, k_2, \ldots, k_d), n = (n_1, n_2, \ldots, n_d) \end{aligned} \tag{2.18}$$

A DFT possui as mesmas propriedades da versão contínua. No caso da DFT, por exemplo, o teorema da convolução é expresso como:

$$DFT\left[x_n * y_n\right] = DFT\left[\sum_{m=-\infty}^{\infty} x_m y_{n-m}\right] = X_k \cdot Y_k \tag{2.19}$$

O operador $\cdot$ é o produto ponto-a-ponto. Ou seja, utilizando a versão DFT, o tempo para se computar uma convolução entre dois vetores de tamanho n reduz de $O(n^2)$ [5] para $O(n)$ mais o tempo para se computar as transformadas, $O(n\log(n))$ [6].

Outros teoremas da versão contínua também podem ser mapeados em suas versões DFT, como pode ser visto em (BOASHASH, 2003).

Implementar a DFT diretamente a partir de sua definição resulta em um algoritmo $O(n^2)$. Mas, felizmente, em 1965, Cooley e Tukey (COOLEY; TUKEY, 1965) publicaram um artigo que apresentou um dos mais importantes algoritmos da ciência da computação, a *Transformada Rápida de Fourier*. Esse algoritmo explora a simetria da equação que define a DFT para reduzir o tempo de computação para $O(n\log(n))$.

Nesse trabalho, o resultado mais utilizado será a eq. (2.19). Já que a covariância espacial pode ser obtida facilmente a partir de uma convolução, como pode ser visto a seguir.

A covariância espacial é definida como:

$$C_{zw}(\boldsymbol{h}) = \frac{1}{N(\boldsymbol{h})}\sum_{\alpha=1}^{N(\boldsymbol{h})} z(\boldsymbol{u}_\alpha)w(\boldsymbol{u}_\alpha+\boldsymbol{h}) - m_{z_{-\boldsymbol{h}}}m_{w_{+\boldsymbol{h}}}, \tag{2.20}$$

onde $m_{z_{-\boldsymbol{h}}}$ e $m_{w_{+\boldsymbol{h}}}$ denotam, respectivamente a média dos valores da cauda e da cabeça do par de variáveis z e w, $N(\boldsymbol{h})$ denota o número de pares que podem ser encontrado a um passo $\boldsymbol{h}$. Aplicando a transformada de Fourier e o teorema da convolução em (2.20), obtém-se:

$$\begin{aligned} &DFT\left[C_{zw}(\boldsymbol{h})\right] = DFT\left[\frac{1}{N(\boldsymbol{h})}\sum_{\alpha=1}^{N(\boldsymbol{h})} z(\boldsymbol{u}_\alpha)w(\boldsymbol{u}_\alpha+h) - m_{z_{-\boldsymbol{h}}}m_{w_{+\boldsymbol{h}}}\right] \\ &s_{zw}(k) = Z_k \cdot W_k - DFT\left[m_{z_{-\boldsymbol{h}}}m_{w_{+\boldsymbol{h}}}\right] \end{aligned} \tag{2.21}$$

A eq. (2.21) define a *densidade espectral* $s_{zw}(k)$ no índice k (no caso discreto, considera-se $h = (h_x, h_y, h_z) = (n_x d_x, n_y d_y, n_z d_z)$, onde d_x, d_y e d_z são as dimensões da

[5] $O(f(n))$ é uma notação usada em ciência da computação para indicar que o tempo de execução de um algoritmo cresce de forma proporcional a $f(n)$ conforme o tamanho da entrada n aumente. Isso permite analisar de forma qualitativa como a performance de um algoritmo degrada com diferentes tamanhos de entrada. Por exemplo, um algoritmo $O(n^2)$ pode demandar vários segundos para executar uma tarefa que um algoritmo $O(n)$ ou $O(n\log(n))$ executaria em milésimos de segundo.

[6] A DFT pode ser computada eficientemente usando-se a transformada rápida de fourier (FFT)

célula do *grid*). Essa equação pode ser computada em $O(n\log(n))$, com $n = N_x N_y N_z$ e N_x, N_y e N_z sendo o número de células em cada dimensão, já que a etapa mais pesada é a computação da transformada de z e w, e $m_{z_{-h}} m_{w_{+h}}$ podem ser computados em $O(n\log(n))$, usando-se as transformadas $DFT\left[m_{z_{-h}}\right] = M_z \cdot \dot{I}$ e $DFT\left[m_{w_{+h}}\right] = I \cdot \dot{M}_w$ (onde I é transformada da função $i(\boldsymbol{u})$ que vale 1 se há informação na posição $\boldsymbol{u}$ ou 0, caso contrário).

Logo, a função covariância $C_{zw}(\boldsymbol{h})$ pode ser definida como:

$$C_{zw}(\boldsymbol{h}) = DFT^{-1}\left[Z_k \cdot W_k\right] - DFT^{-1}\left[M_z \cdot \dot{I}\right] DFT^{-1}\left[I \cdot \dot{M}_w\right] \tag{2.22}$$

De maneira similar, é possível se obter expressões para se computar em $O(n\log(n))$ outras funções como variograma, variograma cruzado, correlograma, entre outros, como pode ser visto em (MARCOTTE, 1996).

Nas sub-seções seguintes, são apresentadas propriedades da transformada discreta de Fourier e o algoritmo para computação de DFTs em $O(n\log(n))$.

2.3.1 Propriedades da Transformada Discreta de Fourier

As propriedades da transformada discreta de Fourier são, basicamente, as mesmas da série de Fourier e da transformada de Fourier. A seguir, considera-se x_n e y_n sinais discretos periódicos com período N e frequência fundamental $w_0 = 2\pi/N$ com DFTs X_k e Y_k, respectivamente:

1. Linearidade do operador: A DFT do sinal discreto $z_n = Ax_n + By_n$ é igual à $Z_k = AX_k + BY_k$.
2. Deslocamento temporal: A DFT do sinal $z_n = x_{n-n_0}$ é $Z_k = e^{-jkw_0n_0}X_k$.
3. Deslocamento de frequência: A DFT do sinal $z_n = e^{jMw_0n}x_n$ é $Z_k = X_{k-M}$.
4. Conjugação: A DFT do sinal $z_n = x_n^*$ é $Z_k = X_{-k}^*$
5. Inversão temporal: A DFT do sinal $z_n = x_{-n}$ é $Z_k = X_{-k}$.
6. Escalonamento temporal: A DFT do sinal $z_{m,n} = \begin{cases} x_{n/m}, & \text{se } n \text{ é múltiplo de } m, \\ 0, & \text{se } n \text{ não é múltiplo de m} \end{cases}$ é $Z_k = \frac{1}{m}X_k$, um sinal discreto com período mN.
7. Convolução periódica: A DFT da convolução, no período N, $z_n = \sum_{r=0}^{N} x_r y_{n-r}$ é $Z_k = NX_kY_k$.
8. Multiplicação: A DFT do sinal $z_n = x_n y_n$ é $Z_k = \sum_{r=0}^{N} X_r Y_{k-r}$.

9. Primeira diferença: A DFT do sinal $z_n = x_n - x_{n-1}$ é $Z_k = (1 - e^{-jw_0 k})X_k$.
10. Soma móvel: A DFT do sinal $z_n = \sum_{r=-\infty}^{n} x_r$, caso seja finito e $X_0 = 0$, então a transformada é igual $Z_k = \frac{1}{1 - e^{-jkw_0}} X_k$.
11. Conjugado simétrico de funções reais: A DFT do sinal x_n real é dada por $Z_k = X^*_{-k}$.
12. Sinal real e par: A DFT de um sinal real e par x_n é também real e par.
13. Sinal real e impar: A DFT de um sinal real e impar y_n é puramente imaginária e impar.
14. Decomposição de sinais reais em partes reais e impares: As DFTs dos sinais $xp_n = \mathbb{PAR}[x_n]$ e $xi_n = \mathbb{IMPAR}[x_n]$ de x_n real são $Xp_k = \text{Real}[X_k]$ e $Xi_k = j\text{Im}[X_k]$.
15. Identidade de Parseval: $\frac{1}{N}\sum_{n=0}^{N} |x_n|^2 dt = \sum_{k=0}^{N} |X_k|^2$.

A identidade de Parseval pode ser usada para mostrar que, por exemplo, quando aplica-se a transformada de Fourier em um vetor de covariância, o valor da densidade espectral s no ponto 0 é igual a variância do vetor x_n. A propriedade da convolução periódica é utilizada para acelerar a computação de muitas estatísticas, como covariância, variograma, correlação, entre outras (já que essas métricas podem ser reescritas como convoluções). Essas propriedades, descritas anteriormente, serão utilizadas durante a implementação dos geradores de tabelas de covariância e simulação espectral.

Na seção seguinte, é apresentado o algoritmo de transformada rápida de Fourier que explora, principalmente, os resultados de simetria, convolução e escalonamento para construir métodos para se computar, em $O(N \log(N))$ passos, a DFT de um sinal discreto x_n, com tamanho N.

2.3.2 A Transformada Rápida de Fourier

A Transformada Rápida de Fourier (*Fast Fourier Transform* - FFT) não é uma transformada nova, trata-se apenas de um algoritmo que explora a simetria da definição da transformada discreta de Fourier para acelerar a sua computação de $O(N^2)$ passos para $O(N \log N)$ passos, onde N é o tamanho do vetor. Além disso, o mesmo algoritmo usado na transformada direta pode ser usado na inversa, bastando trocar o termo o termo $e^{-2\pi i}$ por $e^{2\pi i}$ e dividir o resultado pelo tamanho N do vetor.

Existem diversos algoritmos FFT, cada um com propriedades que o torna mais adequado para um determinado tipo de entrada. Por exemplo, existem algoritmos, como o primeiro algoritmo FFT descrito por Cooley-Tukey (a versão *radix-2* [7] é apresentada

[7] *Radix-2* significa que o tamnaho do vetor precisa ser potência de 2.

no Algoritmo 1) [8], que são eficientes quando o vetor de entrada x possui um tamanho N que é uma potência de 2, ou seja, N pode ser escrito como $N = 2^m$. Mas, na prática, implementações de alta performance de algoritmos FFT demandam técnicas matemáticas e computacionais muito mais sofisticadas, como utilizar mais instruções de ponto flutuante que operam em múltiplos dados simultaneamente (*SIMD - Single Instruction Multiple Data*) e técnicas mais avançadas de teoria dos números e álgebra. Como pode ser visto em (JOHNSON; FRIGO, 2008), uma implementação que utilize os algoritmos FFT mais modernos consegue reduzir o tempo de execução em fatores entre 5 à 40 vezes, quando comparamos com o algoritmo de Cooley-Tukey clássico, já que a performance da FFT tende a se deteriorar significativamente quando o tamanho N do vetor de dados aumenta. Além disso, a complexidade do algoritmo aumenta significativamente quando o tamanho do vetor não é potência de 2, já que é preciso definir uma estratégia de cálculo que decomponha o problema em pedaços menores que são resolvidos recursivamente.

Apesar do problema de se computar FFT não ser trivial, para casos onde N não é potência de 2, o princípio que inspira os diversos algoritmos é simples e é apresentado a seguir.

Considera-se a DFT descrita, a seguir, na eq. (2.23) .

$$X_k = \sum_{n=0}^{N-1} x_n e^{(-2\pi nki)/N}, \tag{2.23}$$

onde X_k é a DFT do vetor $x = \{x_0, x_1, ..., x_{N-1}\}$, com tamanho N e $k = 0, 1, ..., N-1$.

A forma $radix - 2$ é a mais simples do algoritmo de Cooley-Tukey (COOLEY; TUKEY, 1965), essa forma explora a igualdade descrita em (2.24).

Sem perda de generalidade, a eq. (2.23), pode ser decomposta em uma soma de termos pares e ímpares, como visto em (2.24).

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{\{-2\pi(2m)ki\}/N} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{\{-2\pi(2m+1)ki\}/N} \tag{2.24}$$

Fazendo-se $P_k = \sum_{m=0}^{N/2-1} x_{2m} e^{\{-2\pi(2mk)i\}/N}$ e $I_k = \sum_{m=0}^{N/2-1} x_{2m+1} e^{\{-2\pi(2m)ki\}/N}$, a eq. (2.24) pode ser reescrita como:

$$X_k = P_k + e^{-(2\pi ki)/N} I_k \tag{2.25}$$

A eq. (2.25) mostra a decomposição da eq. (2.23) em seus termos pares e impares.

[8] Segundo (HEIDEMAN; JOHNSON; BURRUS, 1985), Gauss apresentou versões de algoritmos FFT para $N = 3^n$ e $N = 6^n$ (GAUSS, 1866).

Entrada: Um vetor de dados $x = \{x_0, x_s, x_{2s}, ..., x_{(N-1)s}\}$, o tamanho N do vetor de dados e passo de incremento s. N deve ser uma potência de 2, ou seja, $N = 2^m$.

saída *A DFT X do vetor x.*
$X_0, X_1, ..., X_{N-1} \leftarrow fft(x, N, s)$:
início
  se $N = 1$ **então**
    $X_0 \leftarrow x_0$;
    retorna X ;
    senão
      // Os primeiros $N/2$ termos são a parte P_k da DFT do vetor x. Para computar P_k computa-se a DFT do sub-vetor $(x_0, x_{2s}, x_{4s}, .., x_{N-2})$.
      $X_0, ..., X_{N/2-1} \leftarrow fft(x, N/2, 2s)$;
      // Os últimos $N/2$ termos são a parte I_k da DFT do vetor x. O sub-vetor $x + s$ é definido como $x + s = (x_s, x_{s+2s}, x_{s+4s}, ..., x_{N-1})$.
      $X_{N/2}, ..., X_{N-1} \leftarrow fft(x + s, N/2, 2s)$;
      para $k = 0...N/2 - 1$ **faça**
        $t \leftarrow X_k$;
        $X_k \leftarrow t + e^{-2\pi ik/N} X_{k+N/2}$;
        $X_{k+N/2} \leftarrow t - e^{2\pi ik/N} X_{k+N/2}$;
      fim
      retorna X ;
    fim
  fim
fim

Algoritmo 1: Algoritmo de Cooley–Tukey (FFT- versão *radix-2*) (COOLEY; TUKEY, 1965) (JOHNSON; FRIGO, 2008) aplicado na computação da DFT X de vetores x com tamanho $N = 2^m$. A função recursiva $fft(x, N, s)$ realiza a computação descrita pelo algoritmo. Para utilizar esse algoritmo, o usuário deve chamar a função fft, considerando um valor inicial de s igual a 1.

Algo importante, que deve ser observado, é que

$$P_{k+N/2} = \sum_{m=0}^{N/2-1} x_{2m} e^{\{-2\pi(2m(k+N/2))i\}/N} = \sum_{m=0}^{N/2-1} x_{2m} e^{\{-2\pi(2mk)i\}/N - 2m\pi i} = P_k \tag{2.26}$$

e, da mesma forma,

$$I_{k+N/2} = \sum_{m=0}^{N/2-1} x_{2m+1} e^{\{-2\pi(2m(k+N/2))i\}/N} = \sum_{m=0}^{N/2-1} x_{2m+1} e^{\{-2\pi(2mk)i\}/N - 2m\pi i} = I_k. \tag{2.27}$$

Ou seja, as eqs. (2.26) e (2.27) demonstram uma importante propriedade de periodicidade dos termos I_k e P_k que permite que a eq. (2.25) seja reescrita como:

$$X_k = \begin{cases} P_k + e^{-2\pi ki/N} I_k, & \text{se } 0 \leq k < N/2, \\ P_{k-N/2} + e^{-2\pi ki/N} I_{k-N/2}, & \text{se } N/2 \leq k < N. \end{cases} \tag{2.28}$$

O termo $e^{-2\pi ki/N}$ é chamado de *fator de giro* da eq. (2.28). E, este termo obedece a eq. (2.29).

$$e^{-2\pi i(k+N/2)/N} = e^{-2\pi ki/N} e^{-\pi i} = -e^{-2\pi ki/N} \tag{2.29}$$

Logo, a eq. (2.28) pode ser transformada, considerando $0 \leq k < N/2$, na eq. (2.30).

$$\begin{aligned} X_k &= P_k + e^{-2\pi ki/N} I_k \\ X_{k+N/2} &= P_k - e^{-2\pi ki/N} I_k \end{aligned} \tag{2.30}$$

Se N é uma potência de 2, $N = 2^l$, a eq. (2.30) pode ser aplicada recursivamente $\log(N) = l$ vezes. Além disso, pela eq. (2.30), pode-se concluir que, para computar os N termos da DFT X_k, precisa-se computar $N/2$ termos P_k e I_k (que, por sua vez, podem ser computados utilizando-se a mesma eq. (2.30)). Logo, o número total de operações realizadas é $O(2N/2 + 4(N/4) + ... + 2^l(N/2^l)) = O(lN) = O(N \log(N))$.

O Algoritmo 1 é a implementação mais simples de uma FFT, conhecida como Algoritmo de Cooley–Tukey versão *radix-2*, mas ele só funciona nos casos em que N é potência de 2. Os termos P_k e I_k da eq. (2.30), também, são DFTs e, em cada nível da recorrência definida na eq. (2.30), pode-se aplicar o algoritmo mais apropriado para se computar a DFT específica. No caso geral, para vetores, com tamanho N, que não sejam potências de 2, será preciso decompor a DFT X_k em sub-transformadas, de modo que cada uma tenha um método apropriado para se obter o resultado. Essa decomposição não é trivial, e, na prática, para se obter um bom resultado é preciso se combinar diferentes tipos de algoritmos FFT, como pode ser visto na implementação da biblioteca FFTW, descrita em (JOHNSON; FRIGO, 2008). Alguns autores chamam essa decomposição de *construção do plano de execução da transformada*. A performance ótima de um algoritmo FFT é um problema de otimização que depende, essencialmente, da decomposição em fatores de N.

Uma versão mais geral do Algoritmo de Cooley-Tukey (COOLEY; TUKEY, 1965), é obtida ao se decompor o tamanho N do vetor x em $N = N_1 N_2$, onde N_1 e N_2 são número inteiros não necessariamente primos entre si [9]. Para uma melhor performance, o ideal é fatorar N em termos inteiros próximos de $\sqrt{N}$. Reescrevendo-se a eq. (2.23), fazendo

[9] Ou seja, o maior divisor comum entre N_1 e N_2 é 1.

$k = N_2k_1 + k_2$ e $n = N_1n_2 + n_1$, obtém-se

$$
\begin{aligned}
X_{N_2k_1+k_2} &= \sum_{n=0}^{N-1} x_n e^{-2\pi n(N_2k_1+k_2)i/N}, \\
X_{N_2k_1+k_2} &= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{N_1n_2+n_1} e^{-2\pi(N_1n_2+n_1)(N_2k_1+k_2)i/(N_1N_2)}, \\
X_{N_2k_1+k_2} &= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{N_1n_2+n_1} e^{-2\pi(n_2/N_2+n_1/N)(N_2k_1+k_2)i}, \\
X_{N_2k_1+k_2} &= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{N_1n_2+n_1} e^{-2\pi((N_2k_1+k_2)n_2/N_2+(N_2k_1+k_2)n_1/N)i}, \\
X_{N_2k_1+k_2} &= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{N_1n_2+n_1} e^{-2\pi(k_1n_2+k_2n_2/N_2+k_1n_1/N_1+k_2n_1/N)i}, \\
X_{N_2k_1+k_2} &= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{N_1n_2+n_1} e^{-2\pi(k_2n_2/N_2)i} e^{-2\pi(k_1n_1/N_1)i} e^{-2\pi(k_2n_1/N)i}, \\
X_{N_2k_1+k_2} &= \sum_{n_1=0}^{N_1-1} \left[e^{-2\pi(k_2n_1/N)i}\right] \left[\sum_{n_2=0}^{N_2-1} x_{N_1n_2+n_1} e^{-2\pi(k_2n_2/N_2)i}\right] e^{-2\pi(k_1n_1/N_1)i}.
\end{aligned}
\tag{2.31}
$$

Pela eq. (2.31), o problema de se computar a DFT de um vetor x com tamanho $N = N_1N_2$ é decomposto em N_1 DFTs $\sum_{n_2=0}^{N_2-1} x_{N_1n_2+n_1} e^{-2\pi(k_2n_2/N_2)i}$. As novas DFTs podem ser computadas utilizando-se alguma das seguintes abordagens:

1. Caso o tamanho seja potência de 2, o Algoritmo 1 pode ser usado;
2. Caso o tamanho seja par, pode-se recorrer a simplificação dada na eq. (2.28);
3. Caso o tamanho seja um número composto [10], pode-se utilizar novamente a simplificação dada na eq. (2.31) e, em seguida, o algoritmo apropriado;
4. Caso o tamanho seja primo, um algoritmo mais sofisticado precisará ser usado, como o algoritmo de Rader (RADER, 1968) ou de Winograd (WINOGRAD, 1978).

É importante lembrar que, para cada N_1, define-se a versão *radix-*N_1 do algoritmo de Cooley-Tukey. Por exemplo, o Algoritmo 1 é a versão *radix-2* do algoritmo de Cooley-Tukey.

O problema de se computar a DFT de um vetor x com tamanho N, número primo, é mais desafiante. Nesse caso, o Algoritmo de Rader (RADER, 1968) ou o de Winograd (WINOGRAD, 1978) precisa ser utilizado, algoritmos que, diferentemente do Algoritmo 1, se baseiam em resultados mais sofisticados de álgebra e teoria dos números. No caso do algoritmo de Rader, o problema de se computar a Transformada de Fourier é transformado no problema de se computar a convolução de dois vetores menores (que, por sua vez, pode ser eficientemente resolvido utilizando-se outras transformadas de Fourier).

[10] Ou seja, um número que pode ser decomposto em fatores primos.

Caso N seja primo, pela teoria dos números, o conjunto de índices $I = \{1, 2, ..., N-1\}$ forma uma grupo finito com multiplicação módulo N [11] . Em outras palavras, qualquer índice $n \in I$ pode ser escrito como $n = g^q \mod N$ para um único $q \in \{0, 1, ..., N-1\}$. Além disso, no grupo multiplicativo módulo N, o inverso multiplicativo [12] $k = g^{-p} \mod N$ também é bem definido, se N é primo, $k \in \{0, 1, ..., N-1\}$ e para cada n existe um único k associado e vice-versa (ou seja, existe uma bijeção entre o conjunto I e ele mesmo, gerada pela operação inverso multiplicativo). Ou seja, a DFT descrita na eq. (2.23) pode ser escrita como

$$\begin{aligned} X_0 &= \sum_{n=0}^{N-1} x_n \\ X_{k=g^{-p}} &= x_0 + \sum_{q=0}^{N-2} x_{n=g^q} e^{-2\pi i/N g^{-(p-q)}}, \end{aligned} \tag{2.32}$$

com $p = 0, 1, .., N-2$ e $q = 0, 1, ..., N-2$.

Pode-se observar, que a expressão $\sum_{q=0}^{N-2} x_{n=g^q} e^{-2\pi i/N g^{-(p-q)}}$ define uma convolução entre as sequências

$$\begin{aligned} a_q &= x_{g^q} \text{ e} \\ b_q &= e^{-2\pi i/N g^{-q}}. \end{aligned} \tag{2.33}$$

Então, pode-se aplicar o teorema da convolução para se computar X_k, usando a_q e b_q, definidos em (2.33). Além disso, a decomposição vista na eq. (2.28) pode ser usada para se simplificar a computação das DFTs das sequências a_q e b_q. Caso, apareça uma DFT da parte par ou impar, com tamanho impar ou primo, é preciso definir qual é a melhor estratégia a ser adotada, se deve-se aplicar novamente a transformação em problema de convolução dadas nas eqs. (2.32) e (2.33), a decomposições em fatores do algoritmo de Cooley-Tukey ou algum outro algoritmo que tenha melhor performance em um determinado tamanho de vetor.

O maior desafio, quando se implementa uma API para se computar FFTs (JOHNSON; FRIGO, 2008), é desenvolver algoritmos que estimem o custo de computação de um determinada estratégia e encontrem a melhor estratégia de resolução de um problema de DFT. Já que, como pode ser visto no algoritmo de Rader e de Cooley-Tukey, por exemplo, a estratégia fundamental é dividir e conquistar. A DFT de um vetor de tamanho N é decomposta em problemas de DFT menores que podem ser resolvidos da melhor forma possível utilizando-se uma determinada estratégia de resolução.

[11] Em aritmética finita, um grupo multiplicativo módulo N é um conjunto finito de inteiros $I = \{1, 2, ..., N-1\}$ onde a operação $x \times y \mod N$ é definida, com $x \in I$, $y \in I$ e $xy \mod N \in I$. $x \mod N$ é o resto inteiro da divisão de x/N.

[12] O inverso multiplicativo r de um inteiro m no módulo N é um valor que satisfaz a igualdade $mr \mod N = 1$.

Um resultado importante, que pode ser constatado observando-se os algoritmos anteriores, é que transformadas de vetores com tamanho potência de 2 são muito mais eficientemente computadas do que outros tamanhos. Além disso, o caso para vetores de tamanho primo é computacionalmente muito mais desafiante, já que o problema é transformado numa convolução de duas sequências menores que é, por sua vez, resolvido também aplicando-se algum algoritmo de FFT. Embora, a ordem do algoritmo continue $O(N \log(N))$, o número de operações inteiras e de ponto flutuantes, que são caras computacionalmente, aumenta significativamente. A natureza recursiva do algoritmo FFT é vital para que uma implementação seja eficiente, já que a cada nível de recorrência (onde, tem-se um vetor a ser transformado com tamanho menor que o original e com propriedades diferentes) é preciso se utilizar um método diferente.

A biblioteca FFTW, desenvolvida por Frigo e Matteo (JOHNSON; FRIGO, 2008), é utilizada para se computar as FFTs, nos algoritmos implementados nesse trabalho. FFTW foi escolhida por causa de sua performance, generalidade (as funções funcionam eficientemente para vetores de qualquer tamanho), suporte sólido, além de ser a solução industrial padrão para esse tipo de problema.

3 Construindo-se tabelas de covariância

No capítulo 2, foi apresentada a Transformada de Fourier que é uma ferramenta matemática que permite transformar problemas complexos em um versão simplificada em outro espaço. O Teorema da Convolução foi apresentado e foi comentada sua importância na aceleração da computação de covariâncias.

Neste capítulo, é apresentado um método para se construir tabelas de covariância a partir dos dados brutos. Essas tabelas de covariâncias substituem modelos de covariância tradicionalmente usados em métodos como Krigagem, SGSIM. Trata-se de uma metodologia em que o modelo de covariância é definido implicitamente por um matriz que fornece todas covariâncias em qualquer lugar da grade de simulação.

A principal vantagem de se utilizar uma tabela de covariância gerada a partir dos dados brutos é o fato de não ser mais necessário utilizar um precioso tempo na modelagem dos variogramas: problema que se torna muito mais complexo conforme o número de dimensões e variáveis aumentam. Embora, não seja possível reutilizar o modelo não-paramétrico em softwares que esperem um modelo parametrizado para covariância (efeito pepita, tipo de estrutura, alcances). O algoritmo de modelagem de tabelas de covariância pode ser estendido para o caso multivariado. Tudo graças ao importante Teorema de Bochner (LOOMIS, 1954), um resultado matemático fundamental que permite que funções definidas positivas sejam facilmente construídas [1] no espaço de Fourier.

Pode-se ver nesse capítulo como a alteração do ponto de vista de um problema simplifica signficativamente o esforço necessário na resolução.

3.1 Covariância espacial, tabela de covariância e densidade espectral

Correlação e covariância espacial são métricas que medem o quanto duas variáveis são relacionadas entre si, de acordo com o afastamento espacial dessas amostras entre si. No caso, essas métricas tentam quantificar o quanto o comportamento de uma variável impacta no comportamento da outra, conforme duas amostras são aproximadas ou afastadas.

Relembrando-se o que foi visto no Cap. 2. A covariância espacial é dada pela eq.

[1] Uma exigência da maioria dos algoritmos de estimativa e simulação é que o modelo de covariância seja definido positivo.

(3.1).

$$C_{zw}(\boldsymbol{h}) = \frac{1}{N(\boldsymbol{h})} \sum_{\alpha=1}^{N(\boldsymbol{h})} z(\boldsymbol{u}_\alpha) w(\boldsymbol{u}_\alpha + \boldsymbol{h}) - m_{z_{-\boldsymbol{h}}} m_{w_{+\boldsymbol{h}}}, \tag{3.1}$$

E, por sua vez, aplicando-se a transformada de Fourier em (3.1), obtém-se a densidade espectral, dada na eq. (3.2).

$$s_{zw}(k) = Z_k \cdot W_k - DFT\left[m_{z_{-\boldsymbol{h}}} m_{w_{+\boldsymbol{h}}}\right] \tag{3.2}$$

A correlação espacial $\rho_{zw}(\boldsymbol{h})$ é obtida dividindo-se a eq. (3.1) por $C_{zw}(0)$.

Na prática, trabalha-se, geralmente, com uma versão discreta de $C_{zw}(\boldsymbol{h})$, a *tabela de covariâncias* $C_{zw}(i, j, k)$, no caso 3D, ou $C_{zw}(i, j)$, no caso 2D. Cada célula (i, j, k) corresponde a um *lag* $h = (i\Delta_x, j\Delta_y, k\Delta_z)$, onde os Δ_l, com $l = x, y, z$, são a discretização espacial, nas direções x, y e z, respectivamente, do *grid* de estimativa ou simulação.

A computação das tabelas de covariância $C_{zw}(i, j, k)$, aplicando-se diretamente a eq. (3.1), demanda $O(N^2)$ operações, já que para cada *lag* é preciso visitar todos nós para se computar covariância $C_{zw}(i, j, k)$. Isso é algo proibitivo na prática, por isso, a computação da densidade espectral s_{zw} (3.2) e, posteriormente, de sua inversa (C_{zw}), é o método mais adequado. Como foi visto no Cap. 2, a convolução, a DFT e a DFT inversa são operações que podem ser computadas em $O(N \log(N))$.

Em (MARCOTTE, 1996), pode-se encontrar métodos para se computar rapidamente outras estatísticas, além da covariância, como variogramas, pseudo-variograma cruzado, entre outros.

Um dos maiores problemas, durante o estágio de preparação dos parâmetros para uma estimativa ou simulação, é encontrar um modelo de covariância que mais se adeque aos modelos experimentais obtidos a partir das amostras coletados em campo. Esse não é um problema trivial, por isso, na próxima seção serão apresentadas diferentes abordagens para a construção de uma tabela de covariância admissível para ser usada pelos algoritmos de estimativa e simulação.

3.2 Uma visão geral das abordagens para se construir tabelas de covariância a partir dos dados

Existem duas diferentes abordagens para o problema de se computar uma tabela de covariância ou correlação espacial para um conjunto de amostras:

- Usando o espaço de dados e busca-se encontrar uma matriz que satisfaça os critérios de positividade.

- Via abordagem espectral, onde é utilizada a Transformada de Fourier da matriz de covariância experimental, para se obter uma matriz próxima admissível. A abordagem espectral utiliza o fato de que, no espaço de Fourier, é muito mais fácil serem satisfeitos os critérios de positividade, graças ao teorema de Bochner (YAO; JOURNEL, 1998).

Além disso, pode-se dividir as estratégias de calibração de uma matriz de covariância em duas classes: métodos explícitos e métodos implícitos. Os métodos explícitos constroem, a partir da tabela de covariância experimental, uma expressão matemática explicita para um modelo de covariância que melhor se ajusta a covariância experimental computada a partir dos dados. Enquanto, os métodos implícitos expressam o modelo de covariância numericamente, não fornecendo uma expressão matemática, ou seja, os valores de covariância em determinado *lag* é obtido a partir de uma tabela (com interpolação sendo usada, caso um determinado *lag* não esteja presente).

Essencialmente, o método explícito baseia-se em uma pesquisa em um espaço de funções, de forma que a melhor combinação linear de funções definidas é obtida. Mais precisamente, dado um conjunto de N funções positivas definidas, procura-se a melhor combinação linear dessas funções tal que seja feito o melhor ajuste aos dados de covariância experimental (LARRONDO; NEUFELD; DEUTSCH, 2003). Em (PYRCZ; DEUTSCH, 2006) pode ser encontrado um conjunto de funções que podem ser usadas para se ajustar geometrias mais complexas. Lembrando que a combinação linear de modelos de covariância é ainda uma função definida positiva e pode ser usada como modelo de covariância (DEUTSCH; JOURNEL, 1998).

Enquanto o método explícito é relativamente bem definido, o método implícito pode ser implementado de diferentes formas, como pode ser visto a seguir:

- Via busca iterativa, no espaço dos dados: (HIGHAM, 2002) apresenta uma solução iterativa que busca a matriz de correlação mais próxima da matriz de correlações experimentais, computadas a partir dos dados brutos. (QI; SUN, 2006) apresenta uma versão que converge mais rapidamente que o algoritmo de Higham, utilizando-se o método de Newton (BERTSEKAS, 1999). (YIN; ZHANG, 2013) apresentam técnicas baseadas em análise de gradientes. A abordagem iterativa é bem rica e podemos encontrar outros trabalhos como, por exemplo: em (XIONG; ZOLOTOV; HE, 2007), é apresentada uma solução que utiliza análise convexa para calibrar o melhor modelo de correlação, e em (MISHRA, 2004) é apresentada uma solução que utiliza otimização clássica para minimizar o erro dado por uma métrica que avalia o quanto a solução está próxima de uma matriz positiva definida. A literatura utilizando

a abordagem iterativa é ampla, já que existem muitas técnicas de otimização que se ajustam muito bem ao problema de se encontrar um candidato que mais se aproxime de um determinado alvo dada uma métrica de performance;

- Utilizando-se inteligência computacional, no espaço de dados: Tabelas de covariância e correlação também são estruturas muito utilizadas nos campos de aprendizado de máquina e análise de grandes volumes de dados (*Big Data*). Entre as diferentes técnicas de otimização utilizando-se inteligência computacional, aqui cita-se duas: Evolução diferencial e mistura de modelos com filtragem. Embora ainda seja uma solução iterativa, a evolução diferencial (STORN; PRICE, 1997) é uma técnica de inteligência computacional que consegue, de forma quase não paramétrica, encontrar um boa solução satisfazendo algum critério de performance, como visto em (MISHRA, 2007). Em (HOFFBECK; LANDGREBE, 1996), são apresentadas técnicas de mistura de modelos e classificação para se construir uma tabela de correlação admissível.

- Aplicando-se um algoritmo de suavização no espaço de Fourier: Os critérios de positividade são muito mais difíceis de serem atendidos no espaço original dos dados do que no espaço de Fourier. Em (YAO; JOURNEL, 1998) (YAO, 1998a), é apresentado um importante algoritmo de suavização da densidade espectral, a versão no espaço de Fourier da tabela de covariância, que simplifica significativamente a modelagem implícita de um dado conjunto de dados. Além disso, devido à natureza das operações realizadas, que são, essencialmente, simples médias em uma janela de dados específica, e a natureza desacoplada do problema de suavização, pode-se obter facilmente implementações de alta performance a partir dessa abordagem. Lembrando-se que a FFT pode ser implementada utilizando GPU (MORELAND; ANGEL, 2003) (CHEN; CUI; MEI, 2010) ou HPC tradicional (DMITRUK et al., 2001) (TAKAHASHI, 2005).

Na próxima seção, é feito um maior detalhamento da abordagem de construção de tabelas de covariância utilizando-se o espaço de Fourier.

3.3 Construção de Tabelas de Covariâncias utilizando-se o espaço de Fourier

O Teorema de Bochner (BOCHNER, 1939) é um resultado fundamental para a construção de uma metodologia para suavizar tabelas de covariância experimental (YAO; JOURNEL, 1998). Esse teorema assegura que se uma função discreta s no espaço discreto de Fourier satisfizer um conjunto de propriedades, sua transformada discreta de Fourier inversa $DFT^{-1}[s]$ é uma função positiva definda. As propriedades que devem ser satisfeitas são:

- $\sum_{k \in \mathbb{Z}^d} s(k) = C(0)$
- s é simétrica.
- $s(k)$ é real positivo $\forall k \in \mathbb{Z}^d$.

$\mathbb{Z}^d$ é o espaço de índices d-dimensional, por exemplo, elementos de $\mathbb{Z}^2$ são $\{(1,1),(1,3),(5,6)...\}$.

Na prática, quando é construída uma função de covariância aproximada $\tilde{C}(\boldsymbol{h})$ a partir dos dados, o resultado é geralmente uma matriz esparsa com muitos dados faltantes e com muitos artefatos. Obviamente, essa matriz pode não satisfazer as exigências de positividade para ser utilizada em algoritmos de simulação e estimativa.

Por isso, é preciso aplicar um algoritmo de suavização, de forma que o modelo resultante seja positivo definido. A solução tradicional é ajustar o modelo utilizando-se um conjunto de funções positiva definidas. Mas, esse processo é muito trabalhoso, já que o esforço de modelagem cresce exponencialmente conforme o número de dimensões aumenta. Para reduzir a complexidade, a modelagem espectral se torna atraente, já que as exigências de positividade no espaço de Fourier são muito mais fáceis de serem atendidas do que no espaço dos dados. Com base nisso, em (YAO; JOURNEL, 1998) é proposto o seguinte algoritmo de suavização:

i Por definição, a funçao de covariância experimental $\tilde{C}(\boldsymbol{h})$ é simétrica.

ii Suavize $\tilde{C}(\boldsymbol{h})$ utilizando médias móveis. $C^s(\boldsymbol{h}) = \sum_{k \in \Omega(\boldsymbol{h})} \tilde{C}(k)\lambda_k$. Onde Ω é definido como $\Omega = \{k \in \mathbb{Z}^d : acos(| < \boldsymbol{u}_h, \boldsymbol{u}_k > |/(||\boldsymbol{u}_h||||\boldsymbol{u}_k||)) < \delta_\theta, ||\boldsymbol{u}_h|| - ||\boldsymbol{u}_k|| < \delta_r\}$, $\delta_\theta > 0$ e $\delta_r > 0$ são, respectivamente, a tolerância angular e radial, $\lambda_k = np(k)/\left[(1 + ||\boldsymbol{u}_h - \boldsymbol{u}_k||)\lambda\right]$, $\lambda = \sum_k np(k)/(1 + ||\boldsymbol{u}_h - \boldsymbol{u}_k||)$ e $np(k)$ é o número de pares utilizados para construir a covariância $\tilde{C}(k)$.

iii Gere a densidade espectral experimental suavizada $s^s(k) = DFT\left[C^s(h)\right]$

iv Normalize o espectro s^s utilizando a expressão $s'(k) = s^s(k)\tilde{C}(0)/(||k||\beta)$, $\beta = \sum_j s^s(j)/||j||$. Isso garante que $\sum_k s'(k) = \tilde{C}(0)$.

v Enquanto ainda houver valores negativos em $s'(k)$ utilize médias móveis com janelas de tamanho variando de 1 até o tamanho do domínio inteiro. No caso 2D, suavize $s'(i,j)$ usando a equação $s'(i,j) = \frac{1}{m_1 m_2} \sum_{l_1=-m_1/2}^{m_1/2} \sum_{l_2=-m_2/2}^{m_2/2} s'(i+l_1, j+l_2)$, m_1 e m_2 são os tamanhos da janela em cada direção. Quanto menor o tamanho da janela melhor, já que uma janela muito grande gera uma espectro muito suave, o

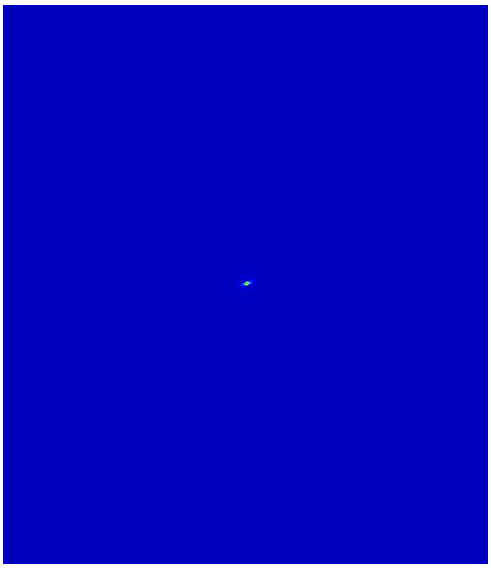
que resulta em uma tabela de covariâncias de baixa qualidade [2]. Caso $s'(k)$ seja um valor negativo bem próximo de zero, pode-se forçar esse valor como 0.

vi Agora tem-se $s'(k)$ normalizado, simétrico e com valores positivos. Se a DFT inversa é aplicada, obtém-se a tabela de covariâncias $C(i, j, k)$ desejada.

O passo (ii) do algoritmo anterior descreve um leque com tolerância angular $\delta\theta$ e tolerância radial $\delta\rho$ em torno do ponto a ser estimado ou suavizado. O peso λ_k é usado para se priorizar informações provenientes de covariâncias computadas com maior número de pares de amostras.

A fig. 4 ilustra uma densidade espectral suavizada utilizando-se o algoritmo apresentado nesta seção. A fig. 5 apresenta uma ampliação de 900% em torno do centro da densidade espectral, permitindo uma melhor visualização de sua forma. E, a fig. 6 mostra a tabela de covariância gerada a partir da transformada de Fourier inversa da densidade espectral mostrada na fig. 4. Essa tabela de covariância foi gerada à partir da base de dados *Walker Lake*.

Figura 4 – Exemplo de densidade espectral suavizada.



3.4 Algumas considerações

A qualidade da tabela de covariâncias gerada pelo algoritmo de Yao depende fortemente da quantidade de dados fornecida para se construir a tabela de covariância experimental. Por usar médias móveis para interpolar os dados faltantes, tanto na primeira

[2] Embora, uma janela muito pequena possa resultar na persistência de artefatos e na criação de um modelo mais ruidoso. Nesse trabalho, usa-se verificação visual para identificar o tamanho adequado da janela de suavização. Mas, um importante problema é definir uma métrica que permita definir o tamanho de janela ótimo que consiga balancear suavização e precisão do modelo.

Figura 5 – Visualização ampliada em 900% do centro da densidade espectral ilustrada na fig. 4.

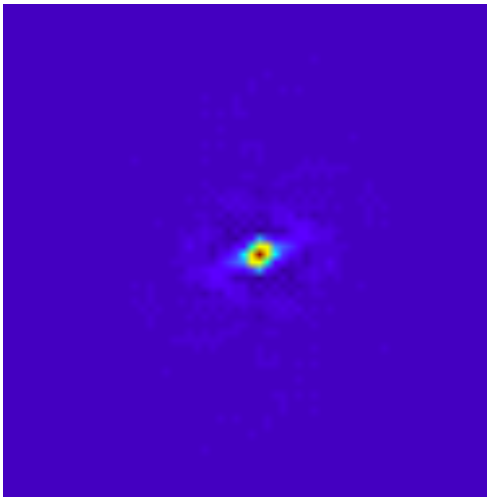
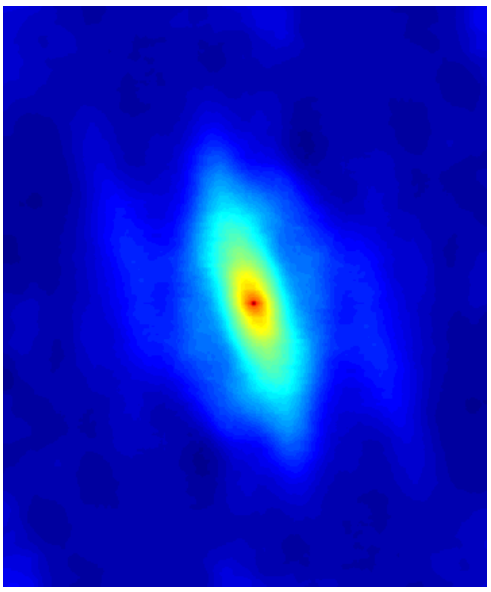


Figura 6 – Exemplo de tabela de covariância suavizada.



versão da tabela de covariância quanto na densidade espectral, pode ocorrer uma suavização excessiva. Para minimizar esse problema, pode-se aplicar uma correção de escala no modelo multiplicando-se todos valores por um fator de correção. Ou seja, caso a covariância $C(0)$ no *lag* $(0, 0, 0)$ seja diferente da variância esperada dos dados σ^2, multiplica-se todos valores da tabela de covariância pelo fator de correção de escala $\sigma^2/C(0)$.

Uma das principais vantagens de se usar tabelas de covariância é o fato dessa ferramenta implicitamente lidar com as variações da covariância de acordo com a geometria dos dados. Isso poupa um significativo tempo de modelagem do geomodelador.

Nos casos em que há poucos dados para se construir a tabela de covariância experimental, deve-se utilizar alguma técnica heurística para se imputar informação adicional: como, por exemplo, explicitamente se imputar um modelo de covariância usando-se uma combinação de funções admissíveis. Caso contrário, a qualidade da modelo gerado

será baixa e este modelo não poderá ser usado em uma estimativa ou simulação futura.

Como pode ser visto em (YAO; JOURNEL, 1998), o teorema de Bochner também permite uma simplificação na geração de modelos de covariância em problemas multivariados, reduzindo-se inclusive a quantidade de modelagens necessárias. A modelagem de covariância no espaço de Fourier, no caso multivariado, é mais simples que o tradicional modelo linear de corregionalização (JOURNEL; HUIJBREGTS, 1978). Já que a dependência espacial no domínio da frequência é removida.

4 Simulação espectral: Turning Bands e Método Integral de Fourier

O caminho aleatório impõe uma série de desafios para a paralelização dos algoritmos de simulação sequencial, como pode ser visto em (RASERA; MACHADO; COSTA, 2015). Afinal, o fato do caminho aleatório ser previamente definido impõe uma sequência de passos para a execução da simulação, o que gera a necessidade de se adicionar alguma lógica de sincronização ou partição do espaço a ser simulado. Isso gera um aumento de complexidade e pode ter impactos negativos na performance computacional. Como foi discutido no Cap. 1, os algoritmos de simulação espectral oferecem uma alternativa que simplifica bastante o desenvolvimento de versões paralelas e de alta performance.

Neste capítulo, são apresentados dois algoritmos espectrais: o *Turning Bands* (LANTUÉJOUL, 2013; MATHERON, 1973) e o Método Integral de Fourier (FIM - *Fourier Integral Method*) (YAO, 1998b). Ambos permitem que a simulação seja realizada sem a necessidade de uma sincronização de passos, algo que simplifica bastante o desenvolvimento de versões de alta performance desses algoritmos.

O FIM realiza a simulação do espaço de Fourier. Nesse algoritmo, os dados são transformados via FFT, e as simulações não-condicionais são geradas perturbando-se a fase dos dados. Após a retro-transformação, o condicionamento é feito através da krigagem do resíduo. Este algoritmo utiliza diretamente a densidade espectral gerada a partir dos dados, quer seja usando-se modelagem explícita (via modelos positivo definidos e heurísticas definidas pelo modelador) ou modelagem implícita (usando-se um algoritmo de suavização, como o descrito em (YAO; JOURNEL, 1998)).

De forma similar ao FIM, o *Turning Bands* utiliza uma transformação do problema de simulação em 2D ou 3D em N simulações 1D. Basicamente, pode-se entender esse algoritmo como um técnica para transformar a simulação geo-estatística em simulações de processos estocástico (onde algoritmos específicos podem ser utilizados para cada tipo de modelo de covariância). Além disso, esse método utiliza uma técnica de descorrelação espacial que quebra o espaço de simulação em um conjunto de linhas independentes que podem ser simuladas individualmente. Essa descorrelação espacial facilita o desenvolvimento de versões HPC do algoritmo. As simulações em cada linha são combinadas para se gerar a simulação final. O *Turning Bands* é fundamentado no *Teorema do Limite Central* (LEMONS; LANGEVIN, 2002).

Nas seções seguintes, é apresentado um maior detalhamento desses dois algoritmos. Além disso, é mostrado como a densidade espectral e a tabela de covariância, gerada

utilizando-se a técnica descrita no Cap. 3 ou por modelagem explícita, podem ser utilizadas conjuntamente com o FIM e com o *Turning Bands*.

4.1 *Turning Bands*

O *Turning Bands* é membro de uma família de algoritmos que reduzem o problema de simulação de uma função aleatória Gaussiana em um problema de simulação de variáveis aleatórias independentes, os *Métodos Espectrais*. Este algoritmo transforma a simulação de uma função de covariância $C(\boldsymbol{h})$ em M simulações de processos estocásticos independentes com covariância $C_\theta(r)$ (onde θ é uma direção e $C_\theta(r)$ é chamada de *representação espectral da covariância* $C(\boldsymbol{h})$ na direção θ e distância r).

Para se gerar as simulações, M linhas são criadas em diferentes direções θ em uma esfera unitária padrão em $\mathbb{R}^D$ [1]. Em seguida, simulações 1D são computadas em cada uma dessas direções e os resultados são combinados para se gerar a simulação. Cada tipo de função de covariância $C(\boldsymbol{h})$ é simulado utilizando-se um algoritmo específico (EMERY; LANTUÉJOUL, 2006). Além disso, se uma função de covariância $C(\boldsymbol{h})$ pode ser escrita como uma combinação linear de modelos mais simples, esses modelos podem ser simulados independentemente e os resultados são combinados linearmente.

Como as simulações 1D em cada nó são completamente independentes entre si, essas simulações podem ser paralelizadas sem qualquer técnica de sincronização. O problema de paralelização se torna *embaraçosamente paralelo*, já que pode-se livremente distribuir as simulações de cada nó entre diferentes *threads* e processos.

Muitos métodos espectrais são baseados nos seguintes resultados e definições (LANTUÉJOUL, 2013):

- Definição de uma função aleatória Gaussiana: Uma função aleatória é Gaussiana se qualquer combinação linear de suas variáveis obedece uma distribuição Gaussiana.

- Caracterização da distribuição espacial de uma função aleatória Gaussiana: A distribuição espacial de uma função aleatória Gaussiana é totalmente caracterizada por seu valor médio e sua função de covariância.

- Vetores Gaussianos independentes: Dois vetores Gaussianos [2] são independentes se e somente se são descorrelacionados [3].

[1] Um bola ou esfera unitária em $\mathbb{R}^d$ é definida como um conjunto de pontos S_d, onde para quaisquer dois pontos $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$ a relação $\Delta(\boldsymbol{u}, \boldsymbol{v}) \leq 1$ se mantém. Δ é um função distância em $\mathbb{R}^d$.

[2] Isto é, um vetor cujos componentes possuem distribuição gaussiana.

[3] Isto é, dois vetores $V = [V_1, ..., V_n]$ e $U = [U_1, ..., U_n]$ são descorrelacionados se a correlação entre eles é 0. Ou seja, são vetores gerados por processos aleatórios independentes.

- Combinação linear de variáveis aleatórias independentes e identicamente distribuídas (I.I.D.): Seja Y_1, Y_2, ..., Y_n uma sequencia de variáveis aleatórias independentes e identicamente distribuídas (I.I.D.), isto é, variáveis aleatórias descorrelacionadas com a mesma função de distribuição de probabilidade δ_Y. Se suas médias são finitas, então pela lei dos grandes números pode-se dizer que a média $(Y_1 + Y_2 + ... + Y_n)/n$ converge para um valor próximo da média m da distribuição δ_Y. Em outras palavras,

$$\lim_{n \Rightarrow +\infty} \frac{Y_1 + Y_2 + ... + Y_n}{n} = m. \tag{4.1}$$

  Caso σ^2 também seja finita, pelo teorema do limite central, têm-se que $(Y_1 + Y_2 + ... + Y_n)/n$ tende a ser Gaussiano em torno da média de $Y_1, Y_2, ..., Y_n$. Então, pode-se dizer que a distribuição da função $Z_n = (Y_1 + Y_2 + ... + Y_n)/n$ converge para

$$\lim_{n \to +\infty} P\left\{ \frac{\frac{Y_1 + ... + Y_n}{n} - m}{\frac{\sigma}{\sqrt{n}}} < y \right\} = G(y), \tag{4.2}$$

  onde $G(y)$ é a função de distribuição Gaussiana padrão.

Em essência, esses resultados afirmam que a simulação de qualquer função aleatória Gaussiana pode ser reduzida na simulação de sequências de variáveis aleatórias independentes e identicamente distribuídas (I.I.D.). Isso é um resultado fundamental, mas não é suficiente, pois não responde algumas questões fundamentais listadas a seguir:

- Como gerar a sequência de variáveis aleatórias I.I.D. associada a uma função aleatória Gaussiana específica?
- Como simular uma variável aleatória com função de covariância $C_\theta(r)$?
- Quantas simulações de processos estocásticos são necessárias para que o método convirja para distribuição Gaussiana esperada?

Alguns algoritmos foram desenvolvidos para responder essas questões como o método espectral, método da diluição, método da tesselação e o *Turning Bands* (LANTUÉJOUL, 2013; MATHERON, 1973). Neste trabalho, o foco será dado ao *Turning Bands*.

Como foi dito anteriormente, o *Turning Bands* é uma técnica que reduz o problema de se simular uma função aleatória gaussiana com função de covariância $C(h)$ em um problema de processo estocástico, onde várias simulações $1D$ de processos com covariância $C_\theta(r)$ são realizadas. Onde θ indica um parâmetro de direção. Esse método pode ser visto como uma generalização do método espectral (LANTUÉJOUL, 2013). A única diferença entre o *Turning Bands* e o método espectral é que este último utiliza exclusivamente funções cosseno para construir a representação espectral das covariâncias.

4.1.1 Alguns resultados

A relação entre a simulação de processos estocásticos independentes e a simulação da função aleatória Gaussiana original é expressa por

$$z^{(n)}(\boldsymbol{u}) = \frac{1}{\sqrt{n}} \sum_{k=1}^{n} Z_k(< \boldsymbol{u}, \theta_k >), \boldsymbol{u} \in \mathbb{R}^d \tag{4.3}$$

onde $(\theta_n, n \in \mathbb{N})$ é uma sequência de direções em uma esfera unitária S_d e $(Z_k, k \in \mathbb{N})$ é uma sequência de processos estocásticos independentes com covariância C_{θ_k}, $\rho_k =< \boldsymbol{u}, \theta_k >$ é a projeção do vetor $\boldsymbol{u}$ na direção θ_n e $Z_n(\rho_n)$ é a representação espectral da função aleatória Gaussiana $z^{(n)}(\boldsymbol{u})$ na direção θ_k.

Sendo que a covariância $C^{(n)}(\boldsymbol{h})$ de $z^{(n)}(\boldsymbol{u})$ é descrita por

$$C^{(n)}(\boldsymbol{h}) = \frac{1}{n} \sum_{k=1}^{n} C_{\theta_k}(< \boldsymbol{h}, \theta_k >), \tag{4.4}$$

onde $C_{\theta_n}(< \boldsymbol{h}, \theta_n >)$ é a representação espectral da covariância $C^{(n)}(\boldsymbol{h})$. Se a covariância C é isotrópica, isto é, pode ser escrita como $C(\boldsymbol{h}) = C_d(||\boldsymbol{h}||)$ para alguma função escalar C_d definida em $\mathbb{R}^+$, então a relação entre C_1 e C_d é dada por

$$C_d(r) = 2\frac{(d-1)\omega_{d-1}}{d\omega_d} \int_0^1 (1-t^2)^{\frac{d-3}{2}} C_1(tr)dt, \tag{4.5}$$

onde ω_d significa, t é uma variável de integração, o d-volume [4] da esfera unitária em $\mathbb{R}^d$. Se $d = 3$, a equação (4.5) é reduzida para

$$C_3(r) = \int_0^1 C_1(tr)dt \tag{4.6}$$

ou, de forma equivalente,

$$C_1(r) = \frac{d}{dr} r C_3(r). \tag{4.7}$$

Curiosamente, para $d = 2$ a relação é mais complicada

$$C_2(r) = \frac{1}{\pi} \int_0^\pi C_1(rsin(\theta))d\theta \tag{4.8}$$

e

$$C_1(r) = 1 + r \int_0^{\pi/2} \frac{d}{dr} C_2(rsin\theta)d\theta. \tag{4.9}$$

Com base nos resultados apresentados nessa sub-seção, pode-se apresentar o algoritmo *Turning Bands*.

[4] O d-volume ω_d é uma generalização do conceito de área, volume, etc. No caso 2D, por exemplo, ω_2 (a área) do circulo de raio r é igual há πr^2. E, no caso 3D, o volume ω_3 da esfera é igual $\frac{4\pi r^3}{3}$.

4.1.2 O Algoritmo

O *Turning Bands* pode ser escrito como

Entrada: Domínio D da simulação, δ_{θ_n} é uma distribuição onde $\sum_{k=1}^{n} \delta_{\theta_k}$ converge fracamente [a] para ϖ, a distribuição uniforme em S_d (esfera unitária em $\mathbb{R}^d$) e $C(\boldsymbol{h})$ é a função covariância isotrópica dos dados.

saída *A realização* $z^{(n)}(\boldsymbol{u})$.

início

Gere um conjunto de direções $\theta_1, ..., \theta_n$ tal que $\frac{1}{n}\sum_{k=1}^{n} \delta_{\theta_k} \approx \varpi$;

Gere processos estocásticos independentes padrões $X_1, ..., X_n$ com funções covariância $C_{\theta_1}, ..., C_{\theta_n}$;

Compute $z^{(n)}(\boldsymbol{u}) = \frac{1}{\sqrt{n}}\sum_{k=1}^{n} Z_k(< \boldsymbol{u}, \theta_k >)$ para qualquer $\boldsymbol{u} \in D$;

fim

Algoritmo 2: Algoritmo *Turning Bands*.

[a] Isto é, para um n suficientemente grande, a distribuição discreta converge para uma distribuição assintótica.

É importante observar que o algoritmo *Turning Bands* não determina um método para gerar as direções ou como construir processos estocásticos com covariância C_θ. Este algoritmo é somente uma descrição de alto nível dos passos de processamento para se gerar uma simulação. Então, para implementar esse algoritmo é necessário resolver o seguinte conjunto de problemas:

1. Como gerar um conjunto de direções satisfazendo $\frac{1}{n}\sum_{k=1}^{n} \delta_{\theta_k} \approx \varpi$? Ou seja, um conjunto de direções quase uniformemente distribuído na esfera S_d?

2. Como simular um processo estocástico satisfazendo C_{θ_n}?

3. Como condicionar as simulações aos dados?

As próximas sub-seções apresentam soluções para esses problemas.

4.1.3 Geração de direções quase uniformemente distribuídas

Existem três modos principais de se gerar direções θ_n quase uniformemente distribuídas em uma esfera unitária S_d:

- Triangular a superfície da esfera (ZAGAYEVSKIY; DEUTSCH, 2016) e utilizar os vértices como pontos para se definir as direções. Essa operação pode ser custosa dependendo do número de linhas desejadas e conforme o número de dimensões aumenta (para três dimensões já se torna um problema complexo).

- Gerar θ_n usando uma distribuição uniforme em S_d. Mas, a convergência de $\frac{1}{n}\sum_{k=1}^{n}\delta_{\theta_k}$ é muito lenta, ou seja, é preciso um número grande de linhas para gerar bons resultados.

- Usar uma sequência quase-aleatória com discrepância fraca [5]. Para $d = 3$, em (FREULON, 1994) é proposta a sequência:

$$\begin{aligned} u_n &= \frac{a_0}{2} + \frac{a_1}{4} + ... + \frac{a_p}{2^{p+1}} \\ v_n &= \frac{b_0}{3} + \frac{b_1}{9} + ... + \frac{b_p}{3^{q+1}} \\ \theta_n &= \left(\cos(2\pi u_n)\sqrt{1-v_n^2}, \sin(2\pi u_n)\sqrt{1-v_n^2}, v_n\right) \end{aligned} \tag{4.10}$$

onde $a_i = 0, 1$, $b_j = 0, 1, 2$ e $n = a_p...a_2a_1a_0 = b_q...b_2b_1b_0 = a_0 + 2a_1 + ... + 2^p a_p = b_0 + 3b_1 + ... + 3^q b_q$. Em outras palavaras, a_i e b_j são os dígitos das representações binárias e ternárias de n, respectivamente.

As sequências u_n e v_n são conhecidas como sequências de Van der Corput e possuem aplicações em otimização aleatória. O algoritmo de Freulon produz direções θ_n tão distantes quanto possíveis de $\theta_1, ..., \theta_{n-1}$ e consegue preencher S_d tão rápido quanto possível. Como pode-se ver nas Figs. 7, 8 e 9, usando-se 1000 bandas é possível gerar um conjunto denso de pontos preenchendo a esfera S_d.

Na prática, o algoritmo de Freulon assegura uma rápida convergência para o algoritmo *Turning Bands*. Geralmente, para grandes conjuntos de dados, usando-se de 1000 a 2000 direções, é possível gerar bons resultados. Para gerar outras configurações de bandas, pode-se aplicar uma rotação aleatória.

[5] Isto é, uma sequência de pontos que preenche de forma quase uniforme a esfera.

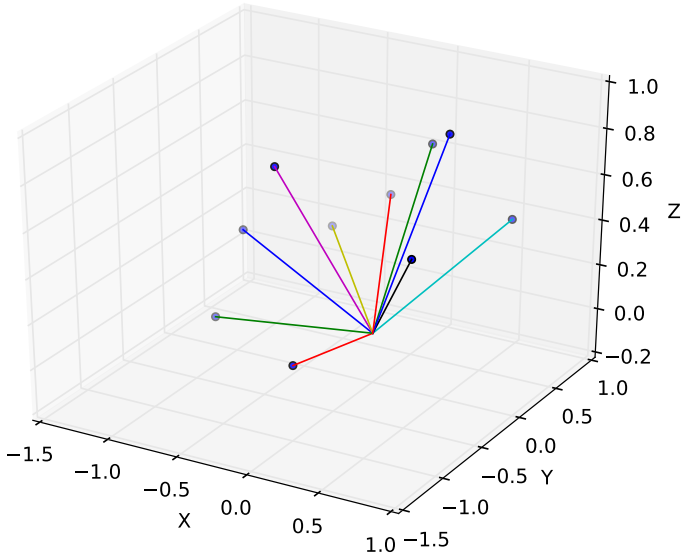


Figura 7 – 10 bandas geradas utilizando-se o algoritmo de Freulon.

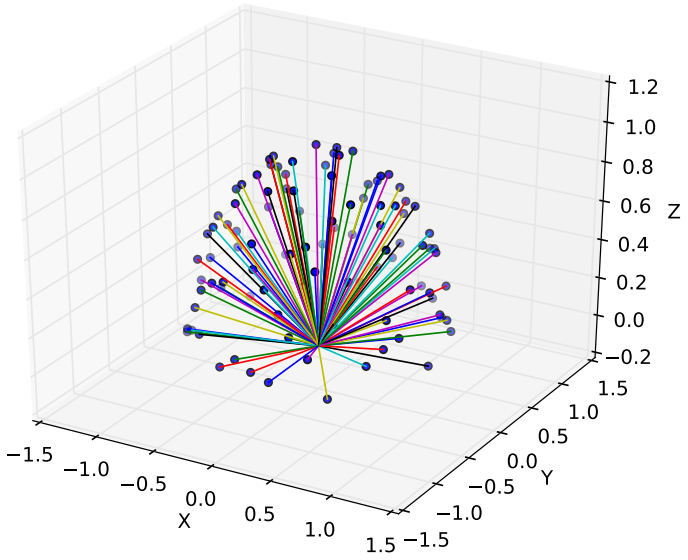


Figura 8 – 100 bandas geradas usando-se o algoritmo de Freulon.

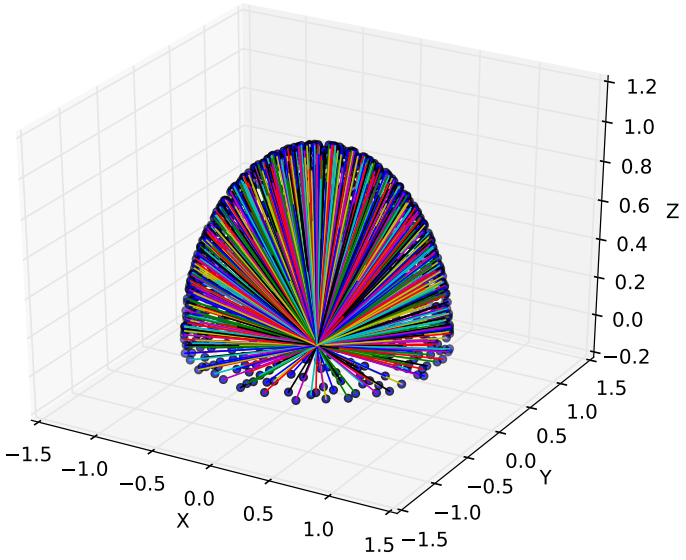


Figura 9 – 1000 bandas geradas utilizando-se o algoritmo de Freulon.

É importante observar que o *Turning Bands* é um algoritmo de simulação que não depende de uma malha regular definida como outros algoritmos de simulação espectral baseados em pertubação da matriz de densidade espectral (Método Integral de Fourier, por exemplo).

4.1.4 Geração das simulações unidimensionais

Tendo-se gerado as bandas que preenchem uniformemente a esfera unitária, agora, é necessário gerar as simulações unidimensionais em cada uma dessas bandas. Nessa sub-seção, são apresentados alguns algoritmos para simular certos tipos de covariâncias unidimensionais C_1 de forma eficiente.

Da mesma forma que o *Turning Bands* não especifica um algoritmo para se gerar as bandas, não há uma determinação específica de qual algoritmo deve ser usado em cada banda para se realizar as simulações de processos estocásticos com covariância C_1. Por isso, na prática, para cada banda e modelo de covariância utiliza-se o algoritmo que de forma mais eficiente consiga realizar a simulação estocástica. Lembrando-se que a simulação de cada modelo pode ser realizada de forma independente e no final os resultados são linearmente combinados. Em (EMERY; LANTUÉJOUL, 2006) são apresentados 15 algoritmos para se gerar simulações de 15 tipos diferentes de modelos de covariância. No caso geral, para se simular um modelo de covariância genérico pode-se usar o método

integral de Fourier 1D para se realizar a simulação da linha (ZAGAYEVSKIY; DEUTSCH, 2016).

Neste trabalho, são apresentadas técnicas para gerar simulações estocásticas dos três modelos mais utilizados: Efeito pepita, Esférico e Gaussiano.

- Para gerar um processo estocástico com uma covariância que é a representação espectral do efeito pepita, com variância c, é suficiente gerar um número aleatório com distribuição Gaussiana $N(0, c)$ com média 0 e variância c.

- O algoritmo para se gerar o processo estocástico associado à covariância esférica

$$C_3(r) = c(1 - \frac{3r}{2a} + \frac{r^3}{2a^3})1_{0 \leq r \leq a} \tag{4.11}$$

  com *sill* c e alcance (fator de escala) a é

  1. Escolha de forma uniformemente aleatória um deslocamento no intervalo $[0, a[$ e divida o eixo em intervalos com comprimento a.
  2. Dentro de cada intervalo, gere uma função linear com igual chance de ser crescente ou decrescente; o sinal do coeficiente é independente de um intervalo para outro.
  3. O valor do coeficiente angular da função linear é $2\sqrt{\frac{3c}{n}}$, onde n é o número de bandas.

- Pode-se utilizar o método espectral contínuo para gerar o processo estocástico associado à covariância Gaussiana

$$C_3(r) = c \exp\left\{-\left(\frac{r}{a}\right)^2\right\} \tag{4.12}$$

  com *sill* c e fator de escala a (alcance, na prática, $a\sqrt{3}$). O algoritmo é o seguinte:

  1. Gere um vetor Gaussiano T e uma fase aleatória uniforme ϕ definida em $[0, 2\pi[$. Em $\mathbb{R}^3$, o vetor Gaussiano pode ser computado como $T = (t_1, t_2, t_3)$, onde $t_i, i = 1, 2, 3$ é um número aleatório Gaussiano com distribuição $N(0, 1)$.
  2. O processo estocástico é simulado usando-se a equação $Z_\theta(\boldsymbol{u}) = \sqrt{2c}\cos(2\pi(< \boldsymbol{u}, T > +\phi))$, onde $\boldsymbol{u} \in \mathbb{R}^3$.

A fig. 10 ilustra uma simulação 1D da representação espectral da covariância esférica. E, a fig. 11 mostra uma simulação não-condicional 2D utilizando-se um modelo esférico.

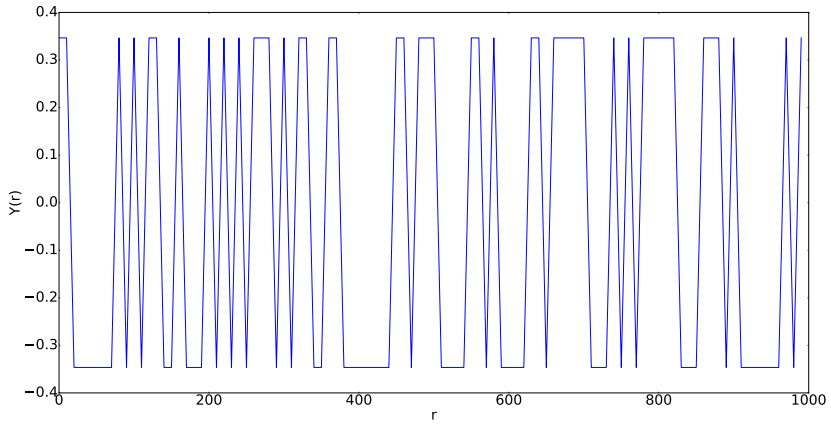


Figura 10 – Simulação do processo estocástico $Y(r)$, em uma direção ρ, associado com a covariância esférica, com $c = 10$ e $a = 10$. r é a coordenada na banda de direção ρ.

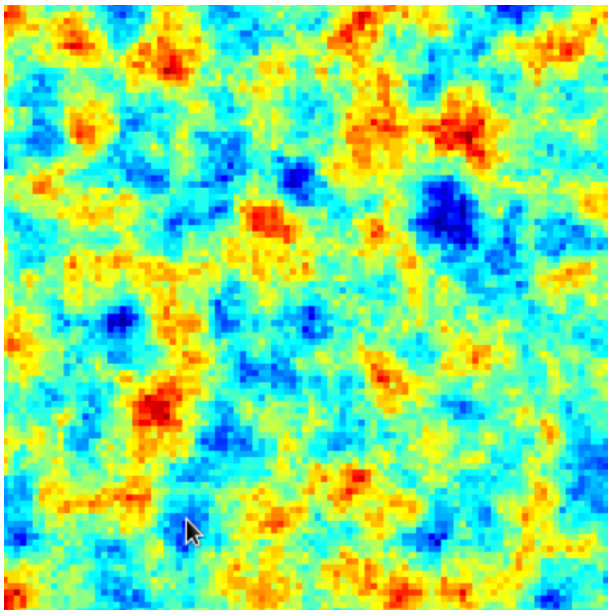


Figura 11 – Simulação não condicional usando-se uma covariância esférica com $c = 10$, $a = 10$ e 1000 bandas.

A fig. 12 ilustra uma simulação 1D da representação espectral da covariância gaussiana. E, a fig. 13 mostra uma simulação não-condicional 2D utilizando-se um modelo gaussiano.

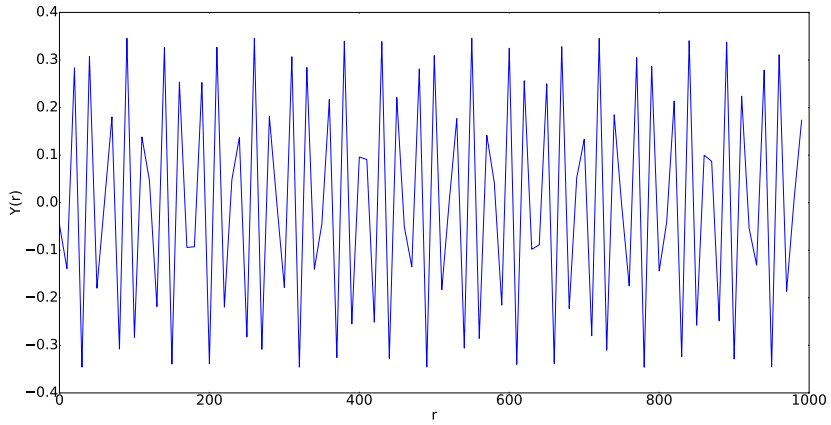


Figura 12 – Simulação de um processo estocástico $Y(r)$, em uma direção ρ, associado a uma covariância Gaussiana com $c = 10$ e $a = 10$. r é a coordenada na banda de direção ρ.

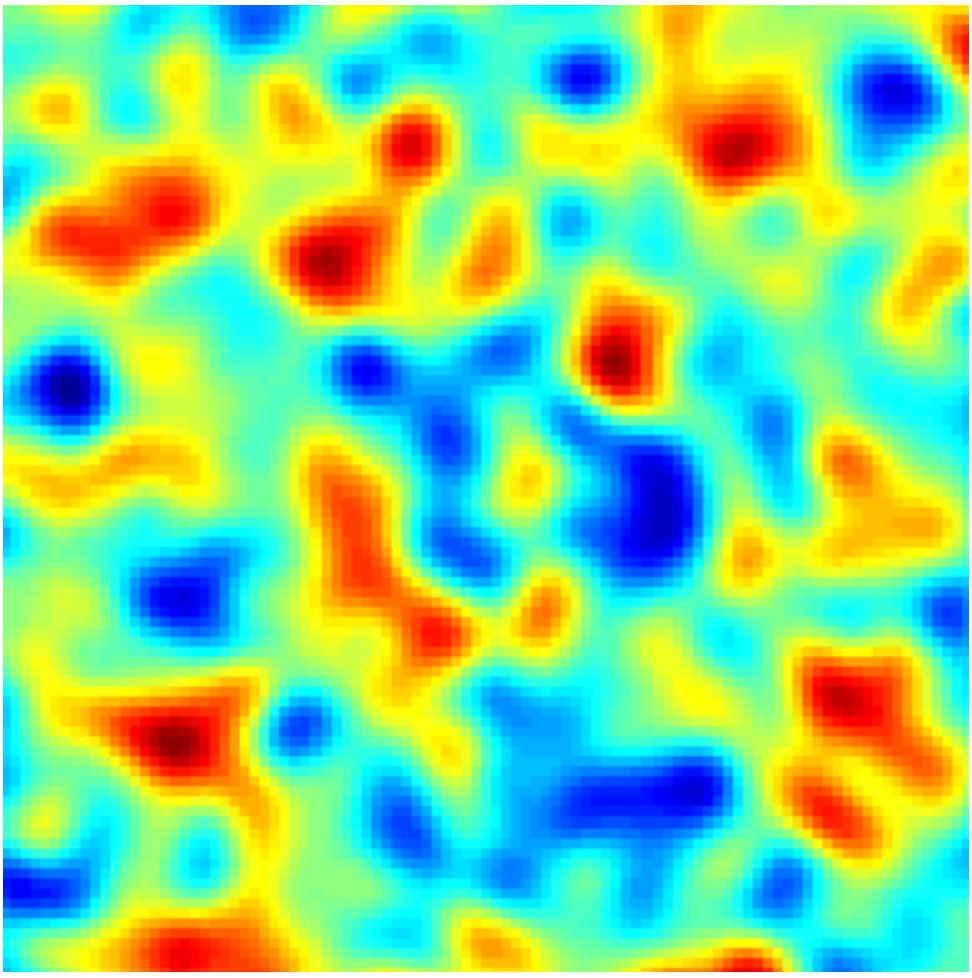


Figura 13 – Simulação não condicional de um modelo Gaussiano com $c = 10$, $a = 10$ e 1000 linhas.

Tendo-se gerado as bandas e as simulações estocásticas independentes em cada banda, agora é necessário condicionar as simulações aos dados. Na seção 4.2, é mostrado como realizar essa condicionamento. Essa técnica é utilizada tanto no *Turning Bands* quanto no Método Integral de Fourier.

Na sub-seção a seguir, são feitas algumas considerações operacionais sobre o *Turning Bands*.

4.1.5 Algumas considerações sobre o *Turning Bands*

Do ponto de vista operacional, considerando-se só os parâmetros de entrada, o *Turning Bands* é muito parecido com algoritmos de simulação tradicionais como a Simulação Sequencial Gaussiana. A única diferença é o parâmetro *número de linhas (bandas)* que exerce um papel fundamental no algoritmo.

A qualidade (velocidade de convergência) do *Turning Bands* é diretamente relacionada ao número de linhas e ao algoritmo utilizado para se gerar as bandas. Na prática, utilizando-se o algoritmo de Freulon, um número de linhas maior que 1000 é suficiente para se gerar bons resultados em 3D (EMERY; LANTUÉJOUL, 2006). Quando o número de linhas não é apropriado, artefatos são gerados e claramente visíveis. As figs. 14, 15 e 16 ilustram o impacto do número de linhas na qualidade da simulação.

Caso o modelo de covariância seja anisotrópico, com anisotropia definida pela matriz T, é preciso transformar as coordenadas do ponto u a ser simulado usando-se a equação $v = Tu$. Isto é, precisa-se projetar o ponto num espaço isotrópico, antes de se realizar a simulação.

O *Turning Bands* é um algoritmo *shared nothing*, isto é, não há interdependência entre os nós para se gerar resultados - ao contrário do SGSIM. Isso permite que uma versão HPC eficiente seja facilmente implementada. Essa é uma característica fundamental dos métodos espectrais, a facilidade de se criar versões HPC, já que todos efetuam uma estratégia de desacoplamento que permite a realização independente e simultânea de diversas etapas do processo de simulação.

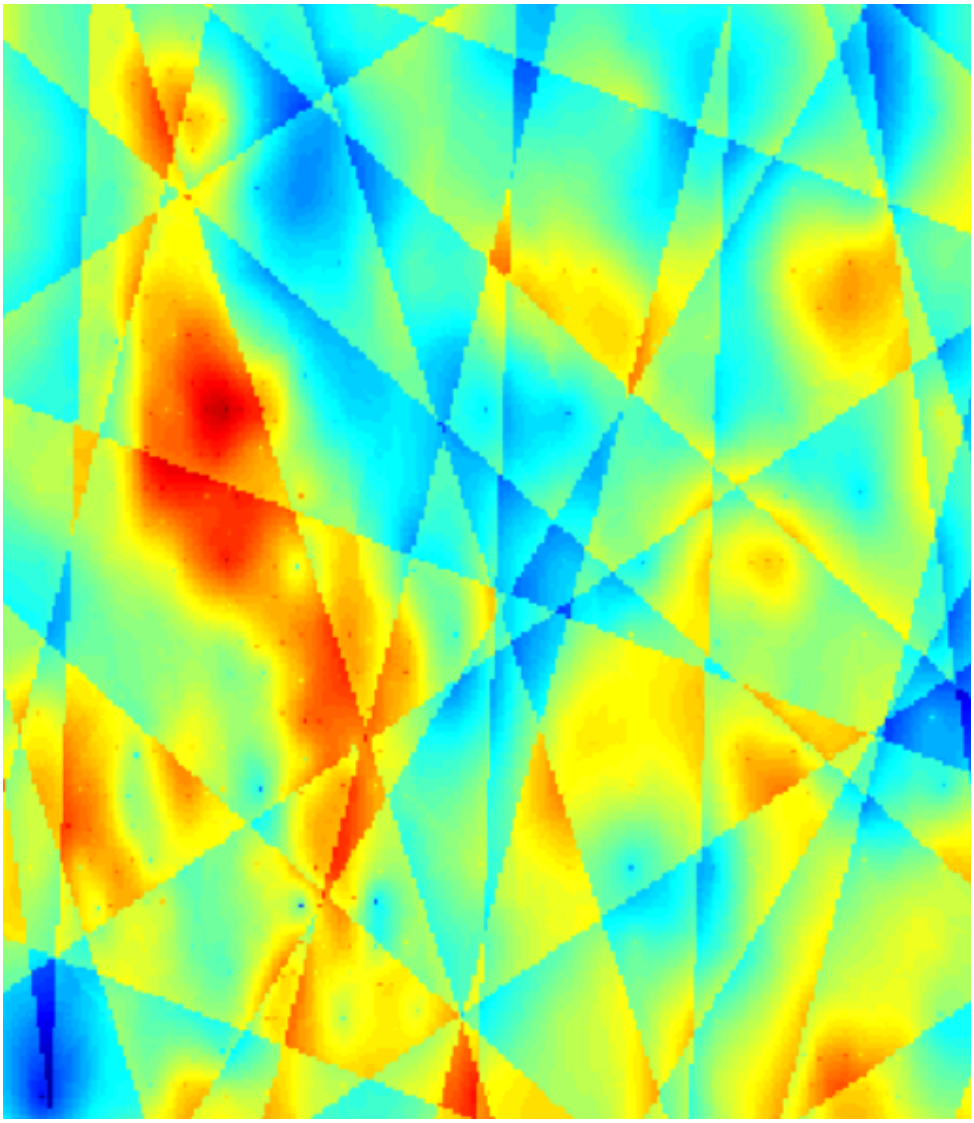


Figura 14 – Simulação condicional do banco de dados *Walker Lake* com número de linhas igual a 10. Pode-se notar a presença clara de artefatos nessa imagem.

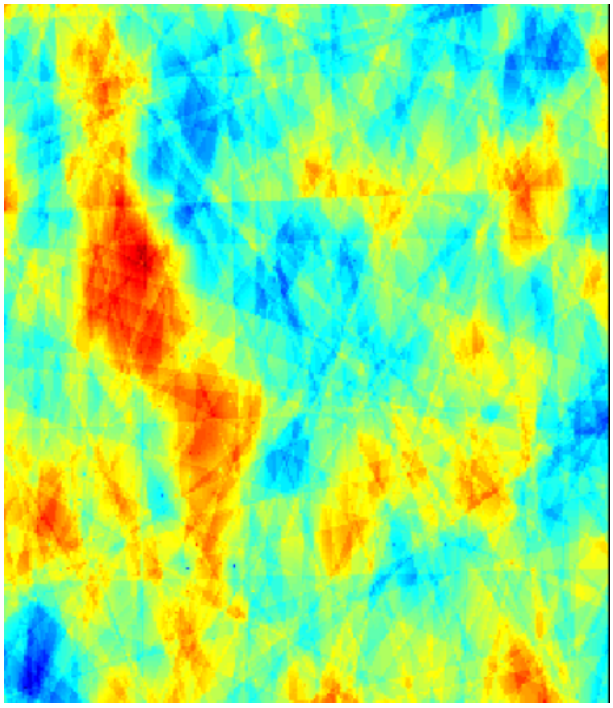


Figura 15 – Simulação condicional do banco de dados *Walker Lake* com número de linhas igual a 100.

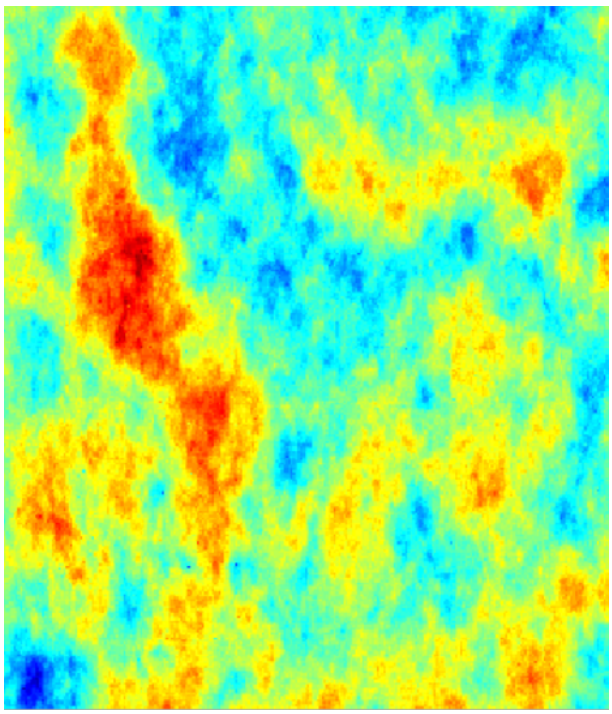


Figura 16 – Simulação condicional do banco de dados *Walker Lake* com número de linhas igual a 1000.

4.2 Condicionando as simulações aos dados

Tanto o *Turning Bands* quanto o Método Integral de Fourier realizam primeiramente uma simulação não condicional utilizando-se uma função de covariância C e, após isso, os dados são usados em uma etapa de condicionamento. Basicamente, o condicionamento nada mais é do que uma krigagem do resíduo. Isso é uma característica fundamental quando se deseja implementar uma versão HPC de um algoritmo espectral, já que a krigagem e a simulação estocástica nas linhas ou pontos são processos *embaraçosamente paralelos*.

O condicionamento aos dados é realizado efetuando-se os seguintes passos:

1. Simule uma função aleatória gaussiana (usando *Turning Bands*, Método Integral de Fourier ou outro tipo de algoritmo de simulação), com média 0 e covariância C, no domínio D e nos pontos de condicionamento CD. Seja $\{z(\boldsymbol{u}) : \boldsymbol{u} \in D\}$ e $\{z(\boldsymbol{v}) : \boldsymbol{v} \in CD\}$ os valores gerados, respectivamente.
2. Estime por krigagem $z^*(\boldsymbol{u}) = \sum_{\boldsymbol{v}\in\Omega(\boldsymbol{u})} \lambda_{\boldsymbol{v}}(\boldsymbol{u})z(\boldsymbol{v})$ para cada $\boldsymbol{u} \in D$.
3. Retorne $(z(\boldsymbol{u}) + y^*(\boldsymbol{u}) - z^*(\boldsymbol{u}) = z(\boldsymbol{u}) + \sum_{\boldsymbol{v}\in\Omega(\boldsymbol{u})} \lambda_{\boldsymbol{v}}(\boldsymbol{u})[y(\boldsymbol{v}) - z(\boldsymbol{v})], \boldsymbol{u} \in D)$.

D é o domínio da simulação, CD é o domínio dos dados condicionantes no espaço gaussiano, $y(\boldsymbol{v})$ são os dados condicionantes, $z(\boldsymbol{u})$ é a simulação não condicional no ponto $\boldsymbol{u}$, $z^*(\boldsymbol{u})$ é a estimava utilizando as simulações $z(\boldsymbol{v})$ nos pontos $v \in CD$, $y^*(\boldsymbol{u})$ é a estimativa utilizando-se os dados e $\lambda_{\boldsymbol{v}}(\boldsymbol{u})$ é o peso de krigagem de $\boldsymbol{v}$ no ponto $\boldsymbol{u}$.

A expressão $z(\boldsymbol{u}) + \sum_{\boldsymbol{v}\in\Omega(\boldsymbol{u})} \lambda_{\boldsymbol{v}}(\boldsymbol{u})[y(\boldsymbol{v}) - z(\boldsymbol{v})]$ é chamada de krigagem do resíduo, é um modo de estimar o erro na posição $\boldsymbol{u}$.

4.3 Método Integral de Fourier (FIM - *Fourier Integral Method*)

No cap. 3, foi visto como utilizar a FFT para construir uma tabela de covariâncias que pode ser usada em algoritmos clássicos de simulação e estimativa. Mas, na literatura, existe um algoritmo que permite a simulação não-condicional no espaço de Fourier (YAO, 1998b), o *Método Integral de Fourier (Fourier Integral Method - FIM)*.

Como no *Turning Bands*, o FIM não exige a criação de um caminho aleatório para se realizar as simulações. Isso permite que o algoritmo seja facilmente adaptado para executar em um ambiente HPC, já que não há necessidade de intercomunicação entre os processos durante a simulação. Ou seja, o FIM permite que vários nós sejam simulados separadamente e ao mesmo tempo. Isso pode resultar em softwares muito mais eficientes que os tradicionais. Afinal, um dos maiores gargalos em qualquer sistema distribuído é o travamento gerado pela necessidade de sincronização de processos.

A ideia central do FIM é perturbar a fase do espectro de densidade usando-se uma distribuição uniforme $U(-\pi, \pi)$. Para dados reais, o espectro de densidade $s(k)$ pode ser escrito como:

$$s(k) = |s(k)| \cos(\phi) - |s(k)| i \sin(\phi), \tag{4.13}$$

onde $|s(k)|$ e ϕ são o módulo e a fase de $s(k)$, respectivamente.

Além disso, pelo Teorema da Convolução (visto no Cap. 2) pode-se ver que, dado $z(\boldsymbol{u})$ (com DFT $Z(k)$), a densidade espectral $s(k)$ pode ser escrita como:

$$\begin{aligned} &s(k) = Z(k) \cdot \dot{Z}(k) = (|Z(k)| \cos(\phi) + i|Z(k)| \sin(\phi)) \cdot \\ &(|Z(k)| \cos(\phi) - i|Z(k)| \sin(\phi)) \\ &s(k) = |Z(k)|^2 \end{aligned} \tag{4.14}$$

Pela eq. (4.14), a densidade espectral nada mais é do que o quadrado de sua magnitude, desconsiderando sua fase.

Logo, para simular uma realização de $z(\boldsymbol{u})$ ($Z(k)$, no espaço de Fourier), basta perturbar a fase do vetor, fazendo $\phi = U(-\pi, \pi)$:

$$Z(k) = \sqrt{s(k)} \left(\cos(\phi) - i \sin(\phi)\right) \tag{4.15}$$

Lembrando-se que a densidade espectral é uma função positiva, então pode-se concluir $z(\boldsymbol{u}) = \dot{z}(-\boldsymbol{u})$, isso implica que só é necessário simular metade do espectro.

Portanto, uma realização $z(\boldsymbol{u})$ do espectro de densidade $s(k)$ é gerada por:

$$z(\boldsymbol{u}) = \mathscr{F}^{-1} \left[\sqrt{s(k)} \left(\cos(\phi) - i \sin(\phi)\right) \right] \tag{4.16}$$

A eq. (4.16) só gera simulações não condicionais. Para condicionar a simulação $z(\boldsymbol{u})$ aos dados $y(\boldsymbol{v})$, é preciso utilizar uma krigagem de resíduos (apresentada na seção 4.2).

Em (YAO, 1998b) é apresentado um algoritmo iterativo para condicionar o espectro aos dados. Mas, além de ser patenteado, o *trade-off* em performance não é tão significativo. Já que a krigagem é um algoritmo extremamente paralelizável e, devido a natureza do FIM, podemos reaproveitar os pesos de krigagem em diferentes simulações.

4.3.1 Algumas considerações sobre o Método Integral de Fourier

O FIM, junto com o algoritmo de Dietrich (DIETRICH; NEWSAM, 1995) e similares, pertence a uma família de algoritmos que permite a geração de simulações não-condicionais diretamente a partir de uma tabela de covariâncias ou densidade espectral. O

algoritmo de Dietrich não utiliza o espaço de Fourier e trabalha diretamente com a tabela de covariância sem demandar qualquer transformação dos dados, mas a complexidade de implementação é superior. Embora, a teoria da Transformada Fourier exija certa sofisticação matemática para ser compreendida, ela permite uma grande simplificação do problema de geração de tabelas de covariância admissíveis (ou seja, funções positivas) a partir dos dados. Sem falar que o FIM é apenas uma consequência direta do Teorema de Convolução, já que ele somente é uma readição da fase, considerando-se que o espectro de densidade preserva a magnitude dos dados.

Caso haja dados amostrais suficientes para se gerar uma boa aproximação da tabela de covariâncias, usando-se algoritmos como o visto no Cap. 3, pode-se poupar um tempo significativo de modelagem.

Mas, é importante lembrar, que o FIM possui a limitação de só poder ser aplicado em malhas regulares, caso seja necessária a simulação em um conjunto de pontos posicionados arbitrariamente no espaço é preciso utilizar técnicas espectrais como o *Turning Bands*. Um fato importante, o FIM 1D pode ser usado para simular uma covariância espectral 1D $C_1(r)$ genérica. Isso permite que tabelas de covariância possam ser utilizadas em combinação com o *Turning Bands*. Em (MANTOGLOU; WILSON, 1982), é encontrado um método para se computar numericamente $C_1(r)$ a partir de uma função de covariância 3D $C(r)$. Essencialmente, precisa-se aproximar a derivada

$$C_1(r) = \frac{d}{dr}[rC(r)]. \tag{4.17}$$

Nas figs. 21 e 22 temos exemplos de realizações não-condicionadas e condicionadas, respectivamente, geradas utilizando-se o FIM. A densidade espectral e a tabela de covariância foram geradas utilizando-se a versão *normal score* da base de dados exaustiva *Walker Lake* (vista na fig. 18). Pode-se notar que a realização gerada pelo FIM (fig. 22) é muito similar à realização gerada pelo *Turning Bands* (fig. 16).

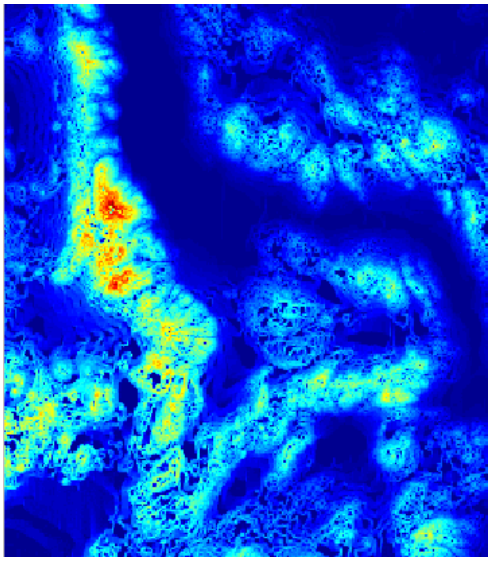


Figura 17 – Base de dados exaustiva *Walker Lake*.

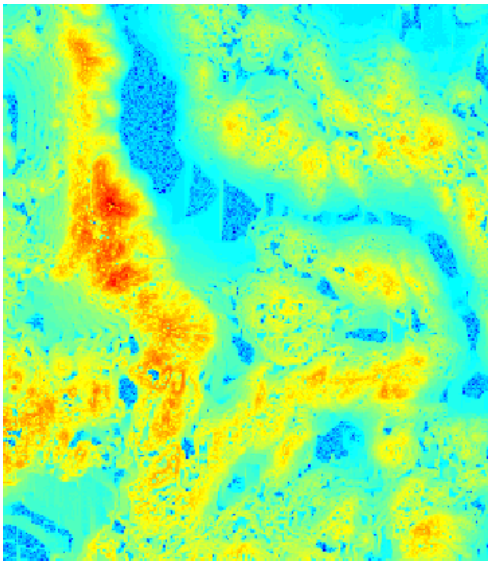


Figura 18 – Base de dados exaustiva *Walker Lake* transformada usando-se *normal score*.

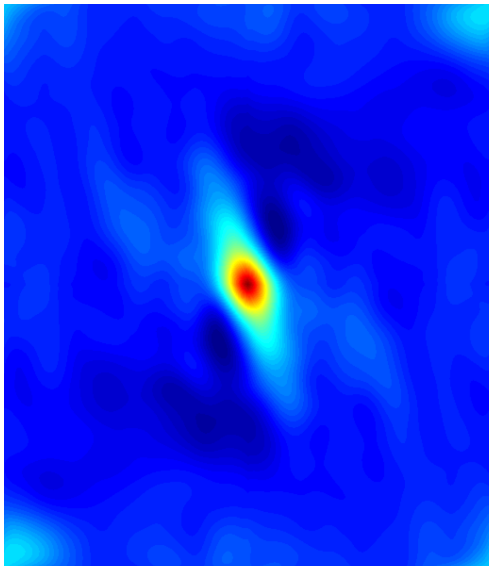


Figura 19 – Tabela de covariância gerada a partir da versão *normal score* da base exaustiva *Walker Lake*.

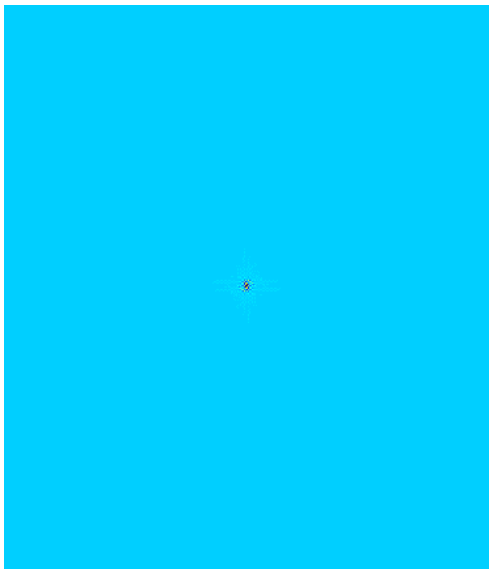


Figura 20 – Densidade espectral gerada a partir da versão *normal score* da base exaustiva *Walker Lake*.

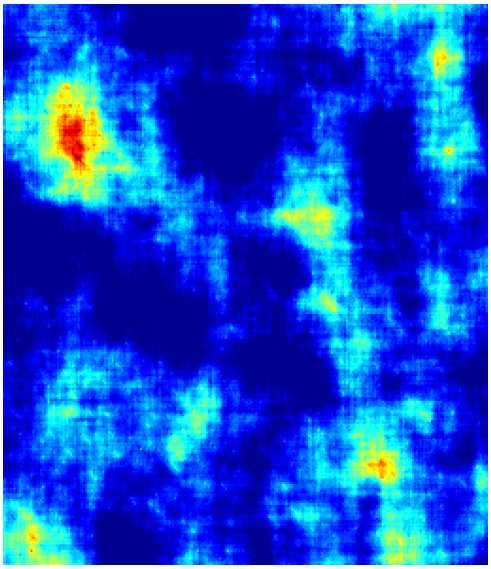


Figura 21 – Simulação não condicional gerada utilizando-se FIM e a densidade espectral da base de dados exaustiva do *Walker Lake*.

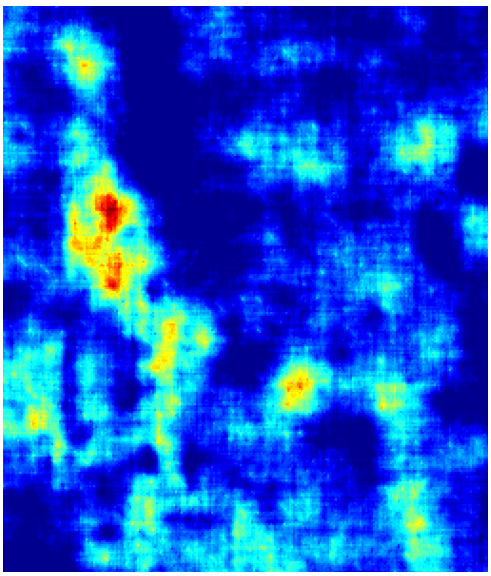


Figura 22 – Simulação condicional gerada utilizando-se FIM e a densidade espectral da base de dados exaustiva do *Walker Lake*.

4.4 Resultados experimentais

Nesta seção, a performance e a qualidade dos resultados simulados pelos algoritmos SGSIM, FIM e TBSIM são analisadas. Em todos cenários, o processo de validação utiliza uma base de dados sintética, criada a partir de uma simulação não-condicional, computada via SGSIM, com os seguintes parâmetros:

- Variograma: $\gamma(h) = 0.3061 + sph(sill = 9.89, ranges = (r1 = 140, r2 = 117, r3 = 62), angles = (azimuth = 127.3, dip = 10.1, rake = 0.))(h)$;

- Distribuição: $\Gamma(\kappa = 1.4, \theta = 2.7)$, média $\mu = 3.78$ e variância $\delta^2 = 10.206$. Uma distribuição gama Γ com parâmetro de forma $\kappa = 1.4$ e escala $\theta = 2.7$.

Todos algoritmos utilizados para se gerar os resultados experimentais, apresentados nesta seção, foram implementados, pelo autor desta tese, na plataforma AR2GAS (*AR2Tech Geostatistics as a Service*), desenvolvida pela AR2Tech (*ADVANCED RESOURCES AND RISK TECHNOLOGY*) [6].

A simulação não-condicional de referência foi realizada em uma malha regular alvo com tamanho de célula $s_x = 2, s_y = 2$ e $s_z = 2$, com 300 células na direção X, 300 células na direção Y, 130 células na direção Z e origem em $(X = 0, Y = 0, Z = 0)$. A Fig. 23 apresenta o sumário estatístico dos dados exaustivos de referência.

Nos experimentos, o condicionamento utilizou 10, 100, 1000 e 5000 amostras extraídos de forma aleatória usando uma distribuição uniforme. A Fig. 24 apresenta a vista de topo dos conjuntos amostrais gerados, a Fig. 25 mostra a seção vertical Oeste-Leste e a Fig. 26 mostra a seção Norte-Sul.

Todos experimentos foram realizados em um computador com processador *Intel Core I7-8550U 8th Gen* de 1,8 GHz, 8 *threads*, 16 GB de memória RAM e 1 TB de disco SSD. A Tabela 1 apresenta o tempo de execução de cada simulador utilizando os quatro conjuntos de amostras. Conforme observado, o FIM foi, em média, $9,74\times$ e $2,05\times$ mais rápido que o SGSIM e TBSIM, respectivamente. Por sua vez, o TBSIM foi $4,74\times$ mais rápido que o SGSIM. Os dados completos de *speed-up* [7] podem ser vistos na Tabela 2 e as Figs. 27 e 28 mostram a performance superior do FIM em todos cenários testados.

[6] Site da AR2Tech: http://ar2tech.com/.
[7] Razão entre o tempo de execução do algoritmo de referência e o algoritmo alvo

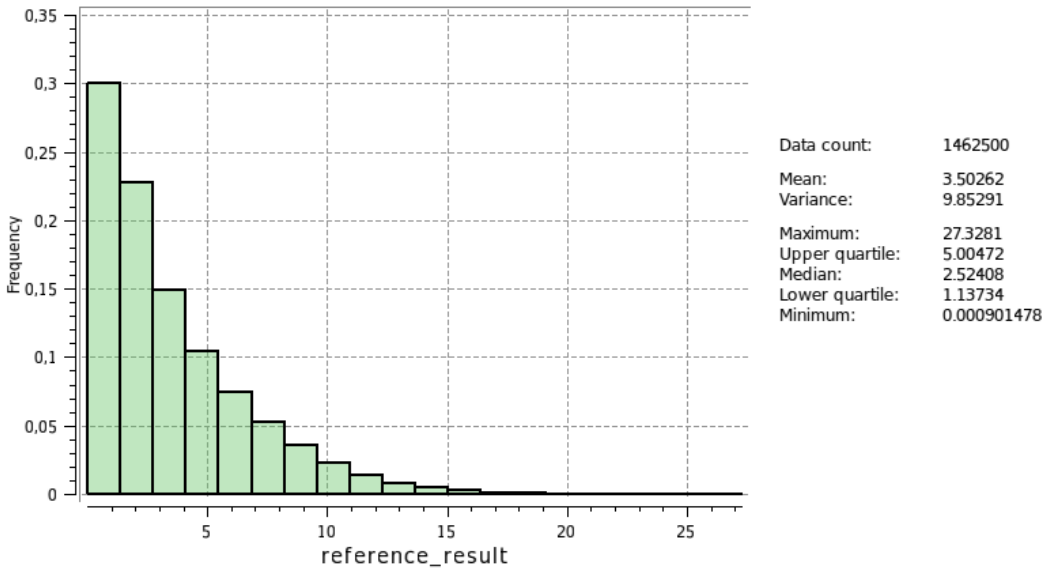
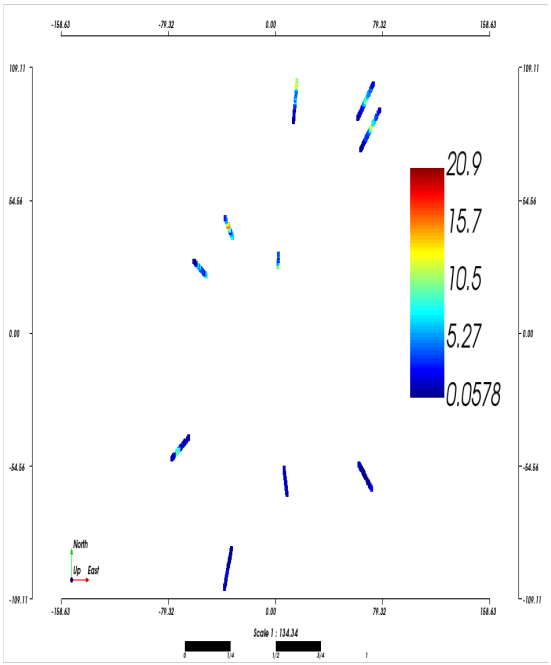
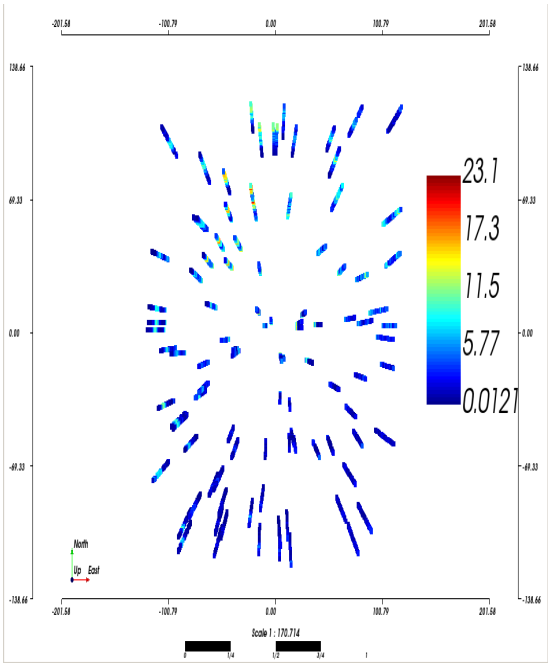


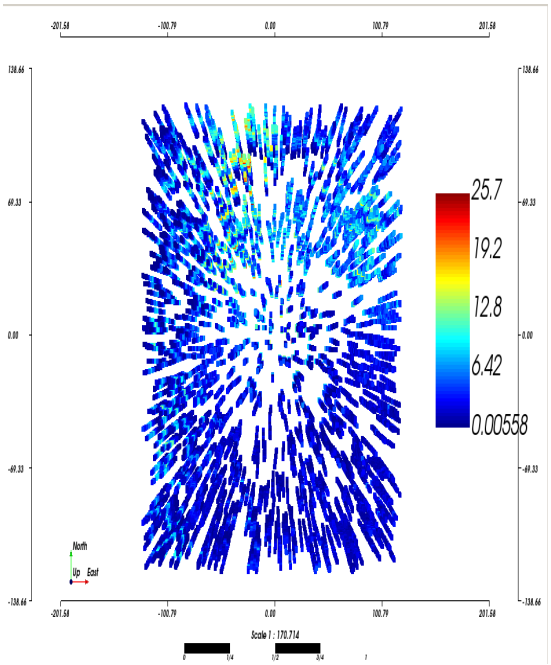
Figura 23 – Distribuição dos dados sintéticos exaustivos usado no teste comparativo.



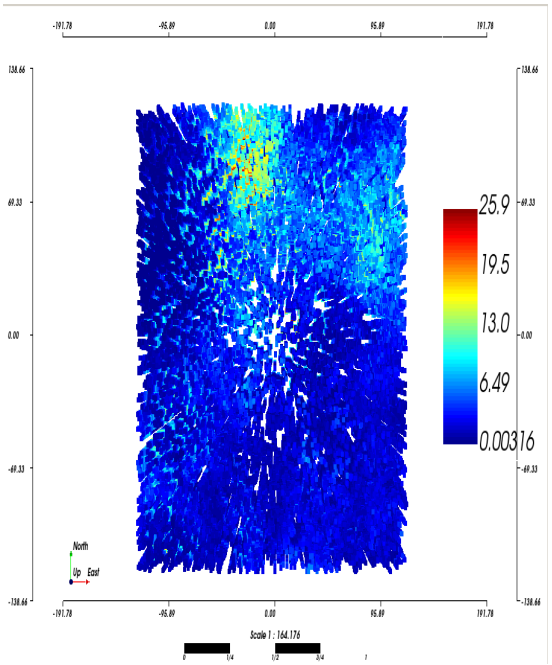
(a) 10 amostras.



(b) 100 amostras.

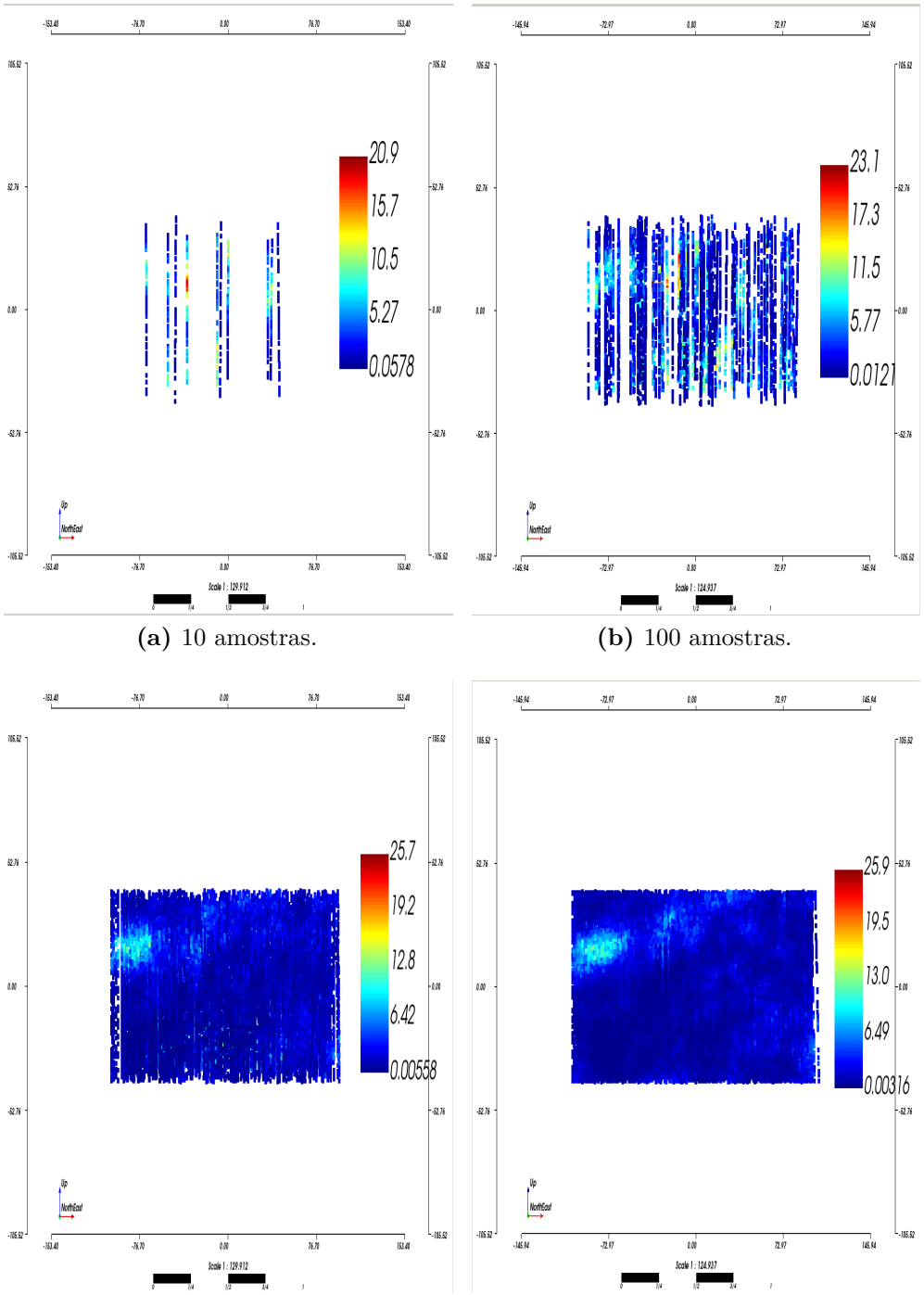


(c) 1000 amostras.



(d) 5000 amostras.

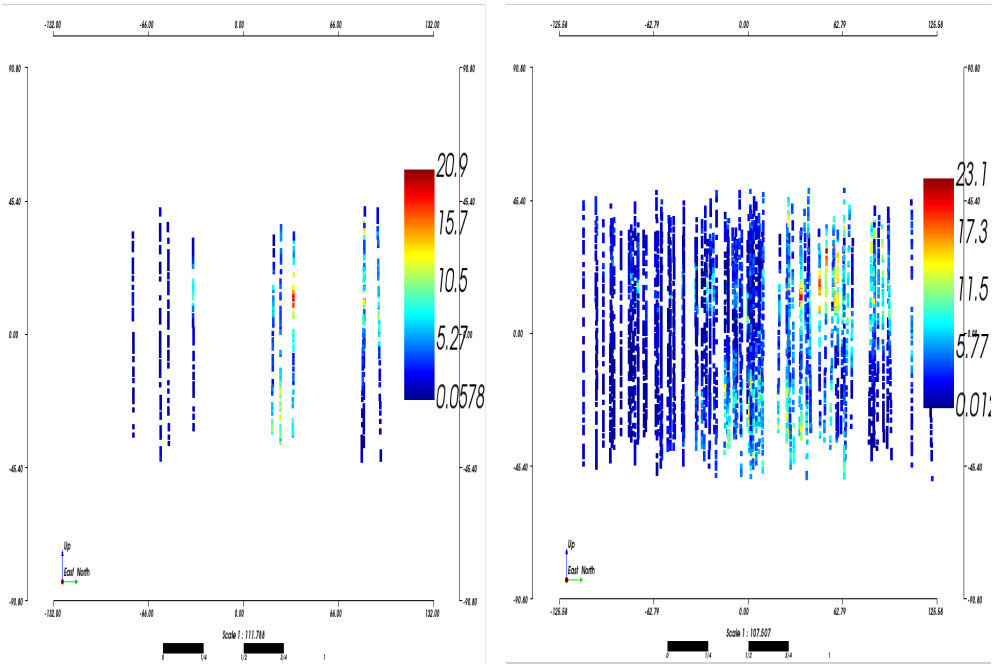
Figura 24 – Vista de topo dos quatro conjuntos de furos de sondagem na malha regular usada nos experimentos.



(a) 10 amostras. **(b)** 100 amostras.

(c) 1000 amostras. **(d)** 5000 amostras.

Figura 25 – Seção vertical Leste-Oeste dos quatro conjuntos de furos de sondagem na malha regular usada nos experimentos.



(a) 10 amostras. **(b)** 100 amostras.

(c) 1000 amostras. **(d)** 5000 amostras.

Figura 26 – Seção vertical Norte-Sul dos quatro conjuntos de furos de sondagem na malha regular usada nos experimentos.

# Drillholes	SGSIM	TBSIM	FIM
10	1486,10	302,67	149,56
100	1519,66	307,61	160,59
1000	1546,53	313,88	158,20
5000	1569,35	369,59	159,86

Tabela 1 – Tempo de execução, em segundos, dos simuladores SGSIM, TBSIM e FIM em cada conjunto de amostras.

# Drillholes	SGSIM/FIM	TBSIM/FIM	SGSIM/TBSIM
10	9,93	2,02	4,90
100	9,46	1,91	4,94
1000	9,77	1,98	4,92
5000	9,81	2,31	4,24

Tabela 2 – *Speed-up* do algoritmo FIM contra SGSIM e TBSIM e TBSIM contra SGSIM.

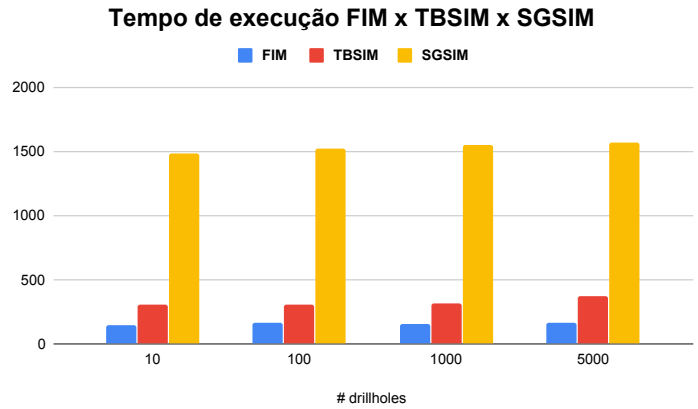


Figura 27 – Gráfico de barras comparativo dos tempos de execução dos simuladores FIM, TBSIM e SGSIM utilizando cada um dos conjuntos de amostra. Nota-se que o FIM apresenta tempo de execução menor que o SGSIM em TBSIM em todos cenários.

# Drillholes	FIM	TBSIM	SGSIM
10	0,743	0,741	0,740
100	0,817	0,819	0,820
1000	0,938	0,936	0,934
5000	0,965	0,961	0,967

Tabela 3 – Comparação dos *Goodness*, eq. (4.18), de cada algoritmo nos cenários com 10, 100, 1000 e 5000 amostras.

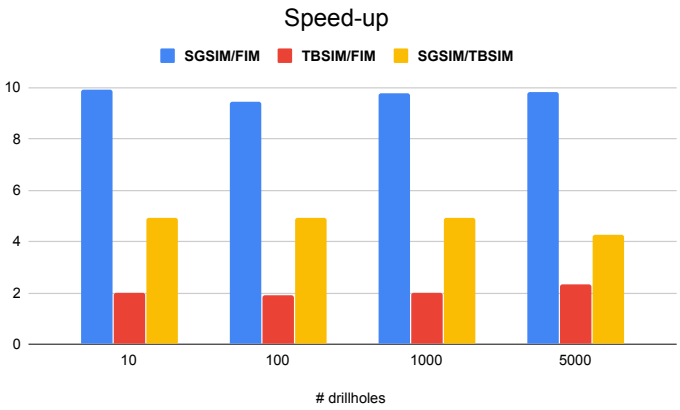


Figura 28 – Gráfico de barras comparativo dos *speed-up* dos simuladores FIM, TBSIM e SGSIM utilizando cada um dos conjuntos de amostra. Nota-se claramente, que o FIM apresenta *speedup* superior ao do TBSIM em todos casos.

Tendo-se mostrado a performance superior do FIM, agora é realizada a análise da qualidade dos resultados gerados por cada algoritmo. A seguir, são apresentados os parâmetros utilizados em cada algoritmo e são comparados os variogramas experimentais, histogramas e *accuracy plot* (GOOVAERTS, 2001) (YAO; YANG; SHAO, 2013) em cada cenário.

Para garantir que todos algoritmos fossem avaliados nas mesmas condições, foram utilizadas a mesma quantidade máxima de dados condicionantes (32) e o mesmo modelo de distribuição e variograma utilizados para se gerar os dados de referência. Além disso, em cada experimento, os ensaios foram realizados utilizando-se cada um dos conjuntos amostrais e foram utilizadas 8 *threads*. Por fim, a semente dos geradores de números aleatórios foi 321913 e foram realizadas 25 realizações. A vizinhança de busca utilizou a anisotropia $ranges = (r1 = 140, r2 = 117, r3 = 62), angles = (azimuth = 127.3, dip = 10.1, rake = 0.)$. O simulador TBSIM utilizou 600 linhas e o simulador SGSIM utilizou 16 nós previamente simulados.

As Figs. 29, 30 e 31 apresentam a flutuação ergódica dos variogramas experimentais, dos experimentos utilizando o simulador FIM, nas direções de maior, média e menor continuidade, respectivamente. Como é esperado, nota-se a convergência dos resultados e

redução da flutuação dos variogramas experimentais em torno do variograma de referência, conforme o número de amostras condicionantes aumenta, isto é, observa-se o fenômeno de redução da incerteza. Na Fig. 32, observa-se o mesmo fenômeno ocorrendo com a flutuação dos histogramas com relação ao histograma de referência. Pode-se observar, na Fig. 33, a satisfatória reprodução de incerteza do simulador FIM, ilustrada pelo gráfico de acurácia. Além disso, nas Figs. 34 e 35, é mostrado que o FIM reproduz a mesma incerteza gerada pelo TBSIM e SGSIM.

Conforme observa-se nas Figs. 36, 37 e 38, as flutuações ergódicas dos variogramas do FIM são similares às obtidas pelo TBSIM. E o mesmo ocorre com os histogramas, vide Fig. 39. Por fim, observando-se as Figs. 40, 41, 42 e 43, pode-se notar que o FIM e TBSIM produzem simulações com variogramas experimentais e histogramas muito similares aos produzidos pelo SGSIM em todos casos testados. A Tabela 3 apresenta a comparação dos *Goodness* de cada algoritmo, pode-se notar que, conforme já observado, todos algoritmos apresentam resultados similares. *Goodness* é definido na eq. (4.18).

$$G = 1 - \int_0^1 [3a(p) - 2][\bar{\xi}(p) - p]dp \tag{4.18}$$

Com:

$$a(p) = \begin{cases} 1, & \text{se } \bar{\xi}(p) \geq p \\ 0, & \text{caso contrário.} \end{cases} \tag{4.19}$$

Onde:

$$\xi(u_i; p) = \begin{cases} 1, & \text{se } F^{-1}(u_i; \frac{1-p}{2}) < z(u_i) \leq F^{-1}(u_i; \frac{1+p}{2}) \\ 0, & \text{caso contrário.} \end{cases} \tag{4.20}$$

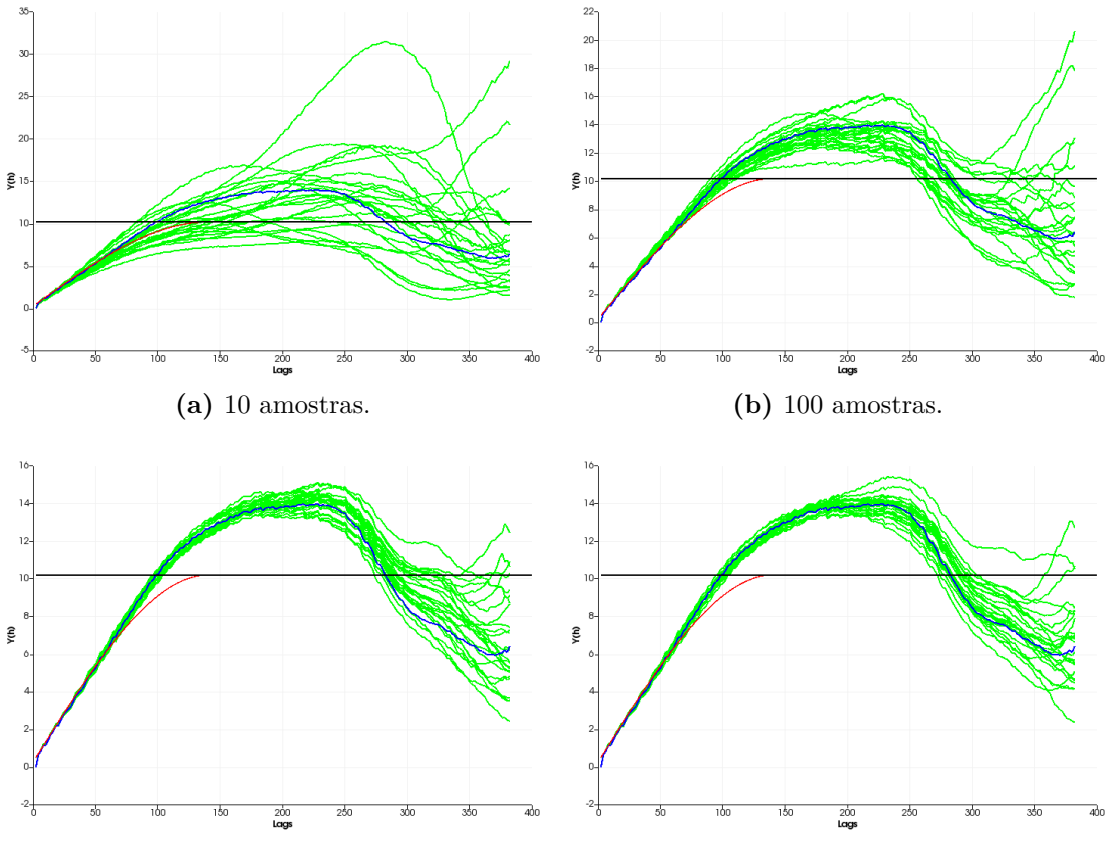
E,

$$\bar{\xi}(p) = \frac{1}{N} \sum_{i=1}^{N} \xi(u_i; p) \ \forall p \in [0, 1]. \tag{4.21}$$

Portanto, com base nos resultados experimentais, pode-se concluir que o FIM além de ser, em média, $2,05\times$ e $9,74\times$ mais rápido que o TBSIM e o SGSIM, respectivamente, produz resultados com qualidade similar ao obtidos por esses algoritmos. Portanto, o FIM é uma excelente alternativa para simulação geoestatística em malhas regulares (cenário muito comum na prática). Já o TBSIM, por não depender de uma malha para realizar a simulação nem depender de um caminho aleatório, é uma excelente alternativa para simulação *gridless* e *streaming*, isto é, simulação sob demanda de qualquer localização.

Essa versão *gridless* é implementada no AR2GAS e fornece um *building block* fundamental para construção de *workflows* de simulação mais sofisticados, como simulação em bloco.

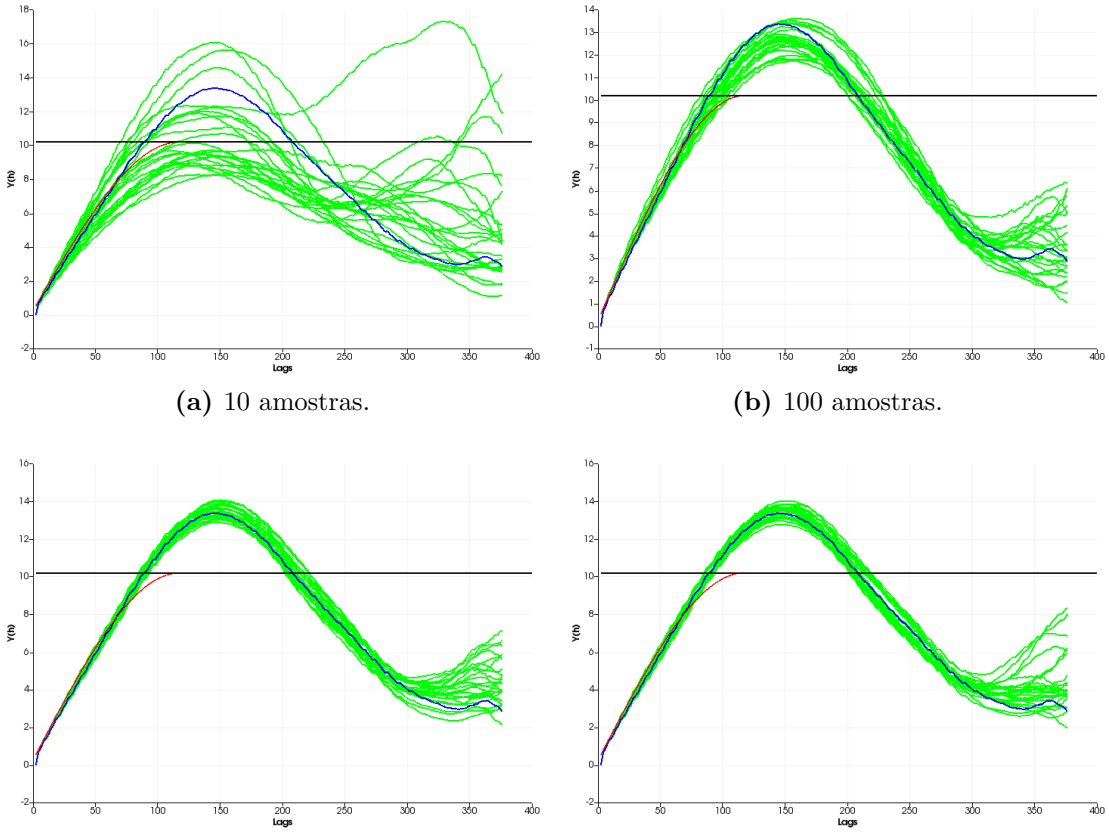
Os *Jupyter notebooks* usados para se gerar os resultados apresentados nessa seção, além dos dados brutos, podem ser encontrados no *link*: https://tinyurl.com/y8olrxxg.



(a) 10 amostras. **(b)** 100 amostras.

(c) 1000 amostras. **(d)** 5000 amostras.

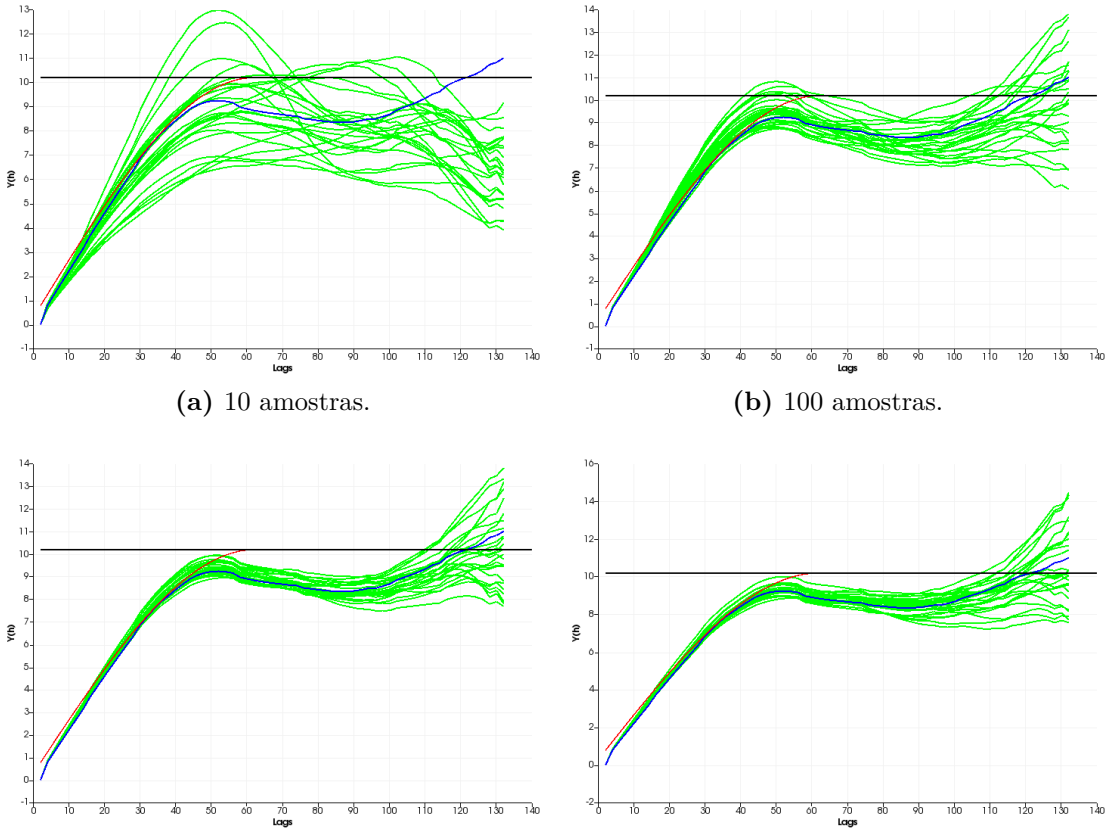
Figura 29 – Flutuação ergódica dos variogramas experimentais, na direção de maior continuidade, das simulações utilizando o simulador FIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em verde, têm-se os variogramas experimentais das simulações FIM, em azul têm-se o variograma experimental dos dados exaustivos de referência e em vermelho têm-se o variograma de referência.



(a) 10 amostras. **(b)** 100 amostras.

(c) 1000 amostras. **(d)** 5000 amostras.

Figura 30 – Flutuação ergódica dos variogramas experimentais, na direção de média continuidade, das simulações utilizando o simulador FIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em verde, têm-se os variogramas experimentais das simulações FIM, em azul têm-se o variograma experimental dos dados exaustivos de referência e em vermelho têm-se o variograma de referência.



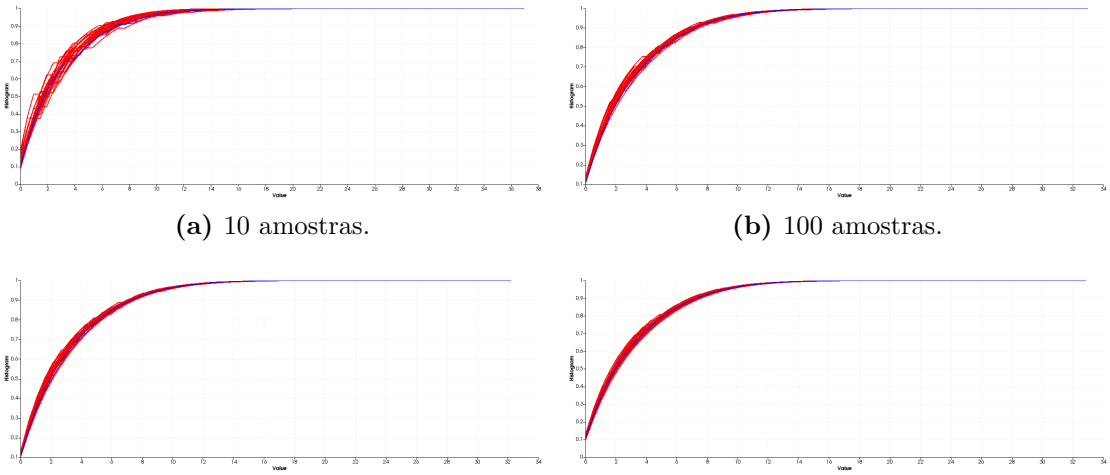
(a) 10 amostras.

(b) 100 amostras.

(c) 1000 amostras.

(d) 5000 amostras.

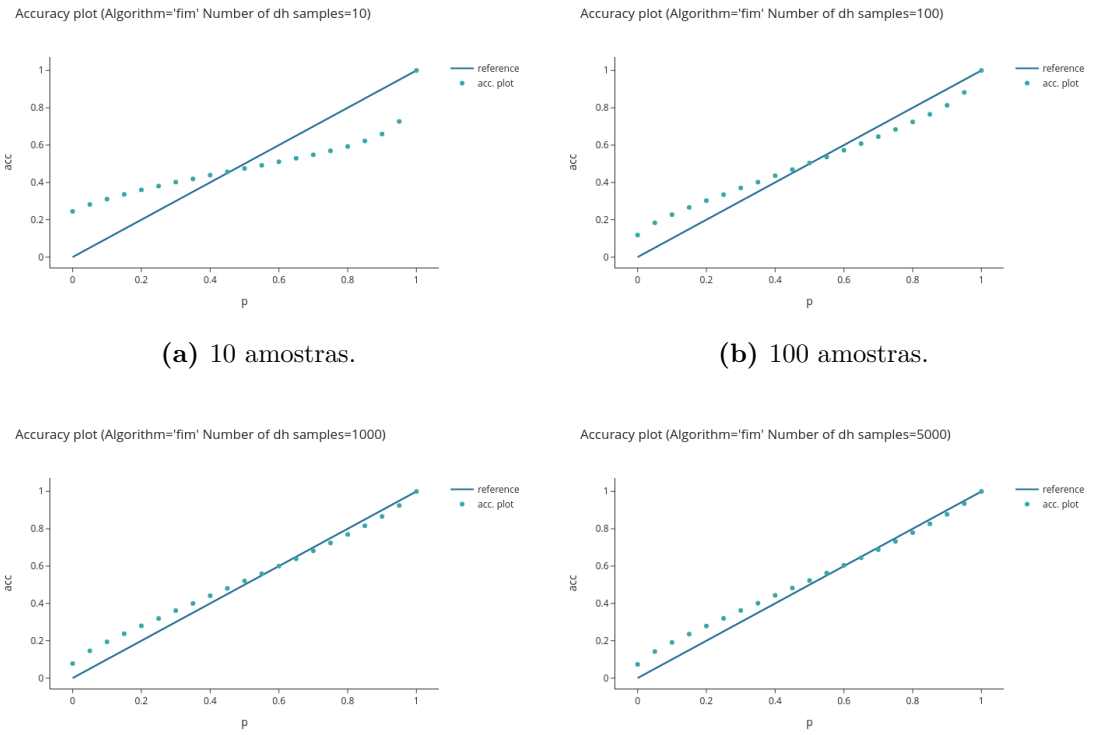
Figura 31 – Flutuação ergódica dos variogramas experimentais, na direção de menor continuidade, das simulações utilizando o simulador FIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em verde, têm-se os variogramas experimentais das simulações FIM, em azul têm-se o variograma experimental dos dados exaustivos de referência e, em vermelho, têm-se o variograma de referência.



(a) 10 amostras. **(b)** 100 amostras.

(c) 1000 amostras. **(d)** 5000 amostras.

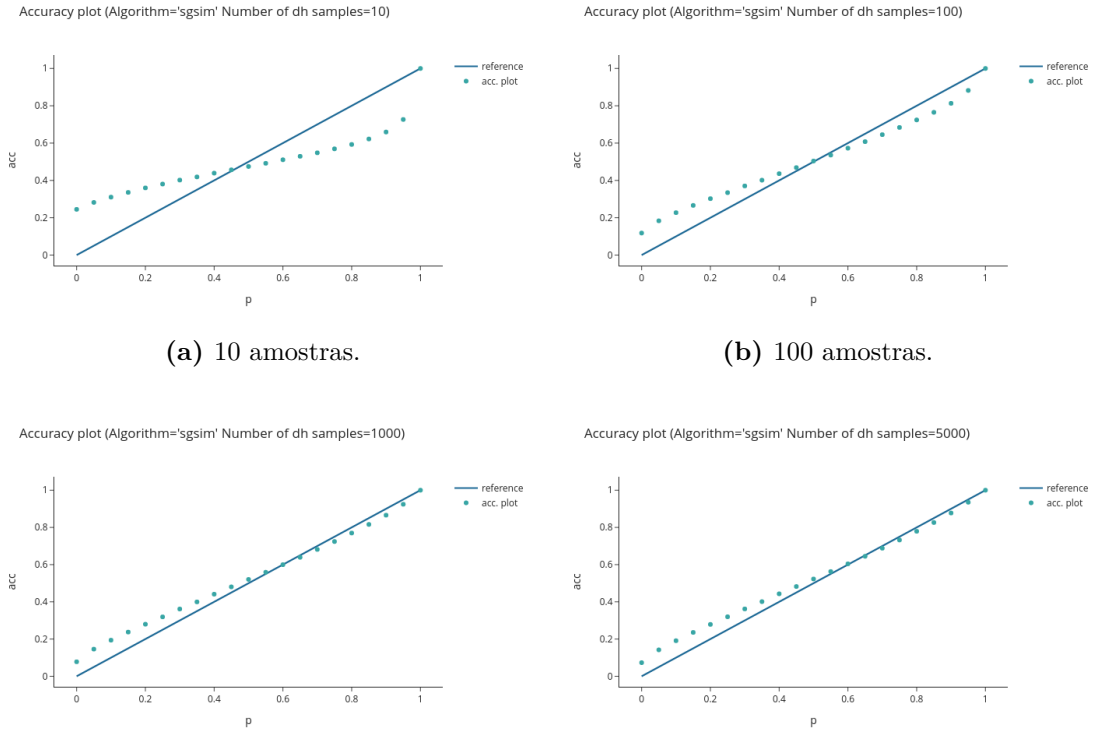
Figura 32 – Flutuação ergódica dos histogramas das simulações utilizando o simulador FIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em vermelho, têm-se os histogramas das simulações FIM e, em azul, têm-se o histograma dos dados exaustivos de referência.



(a) 10 amostras. **(b)** 100 amostras.

(c) 1000 amostras. **(d)** 5000 amostras.

Figura 33 – *Accuracy plots* das simulações utilizando FIM nos cenários com (a) 10, (b) 100, (c) 1000 e (d) 5000 amostras.



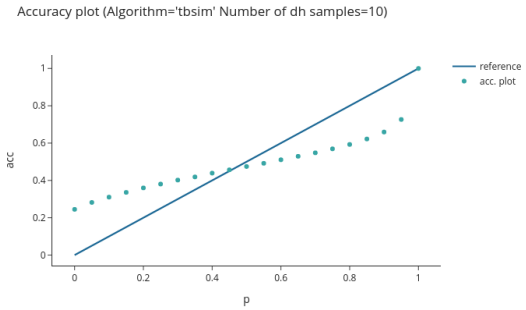
(a) 10 amostras.

(b) 100 amostras.

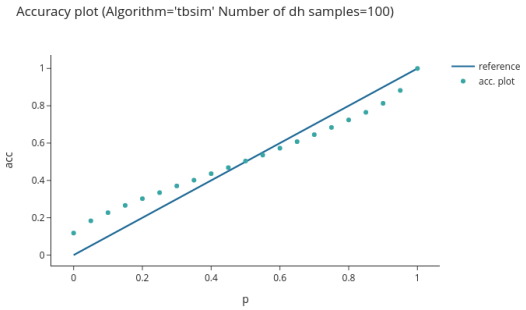
(c) 1000 amostras.

(d) 5000 amostras.

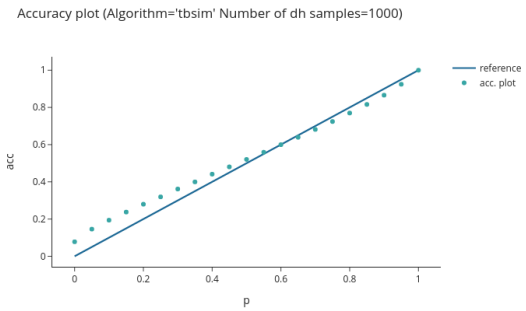
Figura 34 – *Accuracy plots* das simulações utilizando SGSIM nos cenários com (a) 10, (b) 100, (c) 1000 e (d) 5000 amostras.



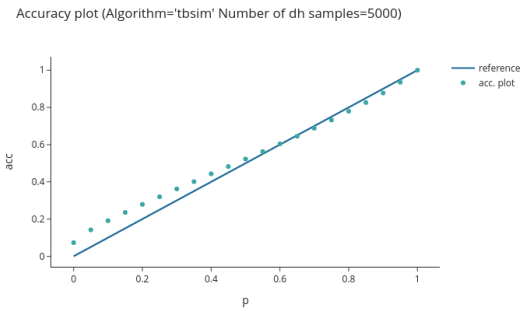
(a) 10 amostras.



(b) 100 amostras.

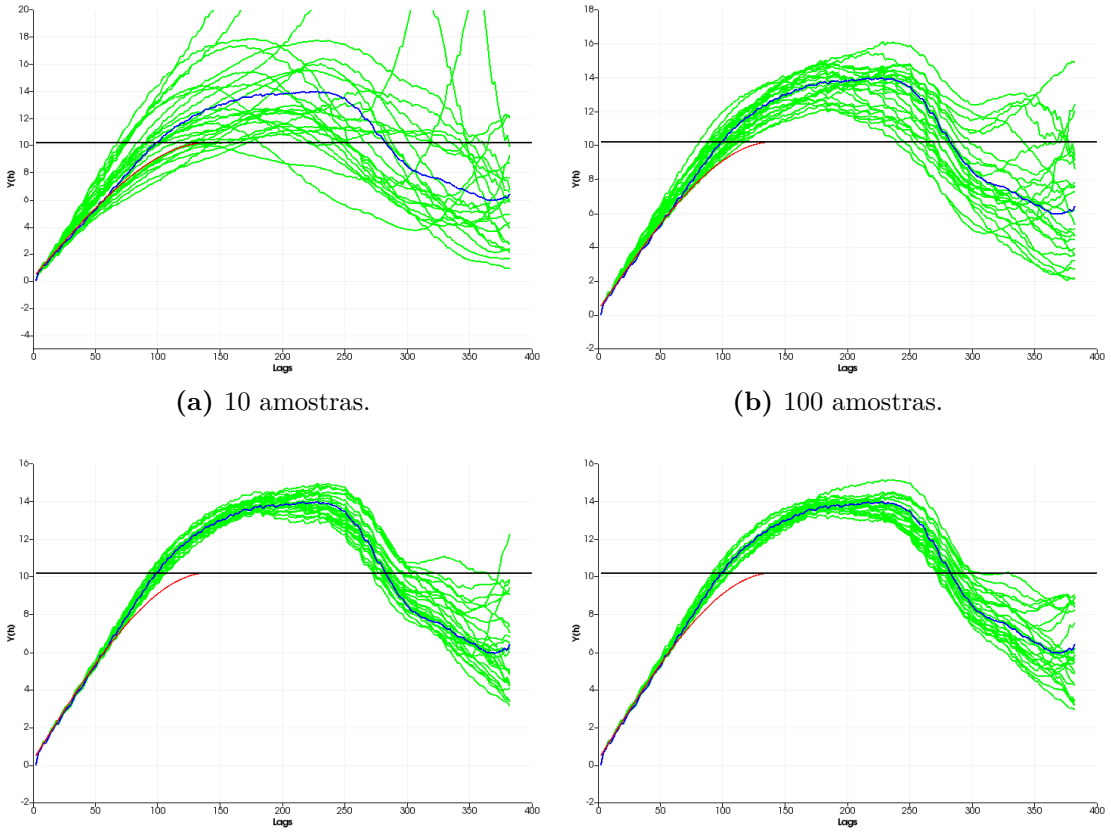


(c) 1000 amostras.



(d) 5000 amostras.

Figura 35 – *Accuracy plots* das simulações utilizando TBSIM nos cenários com (a) 10, (b) 100, (c) 1000 e (d) 5000 amostras.



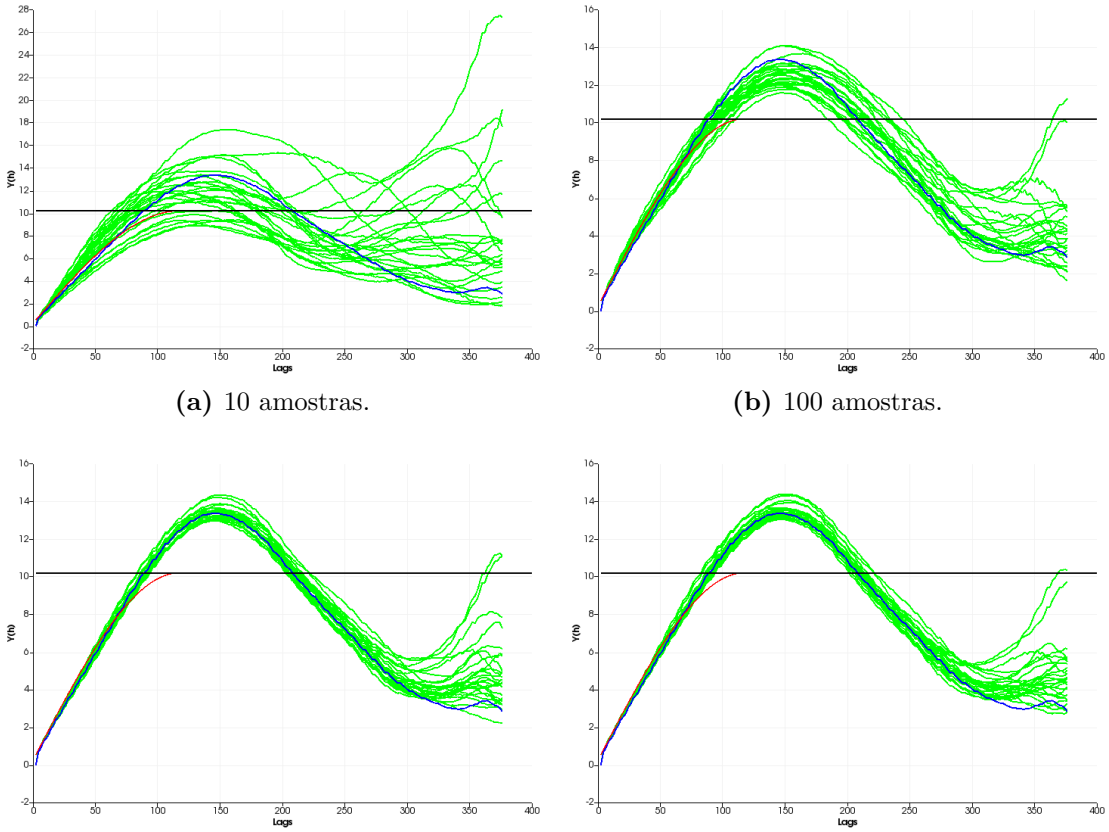
(a) 10 amostras.

(b) 100 amostras.

(c) 1000 amostras.

(d) 5000 amostras.

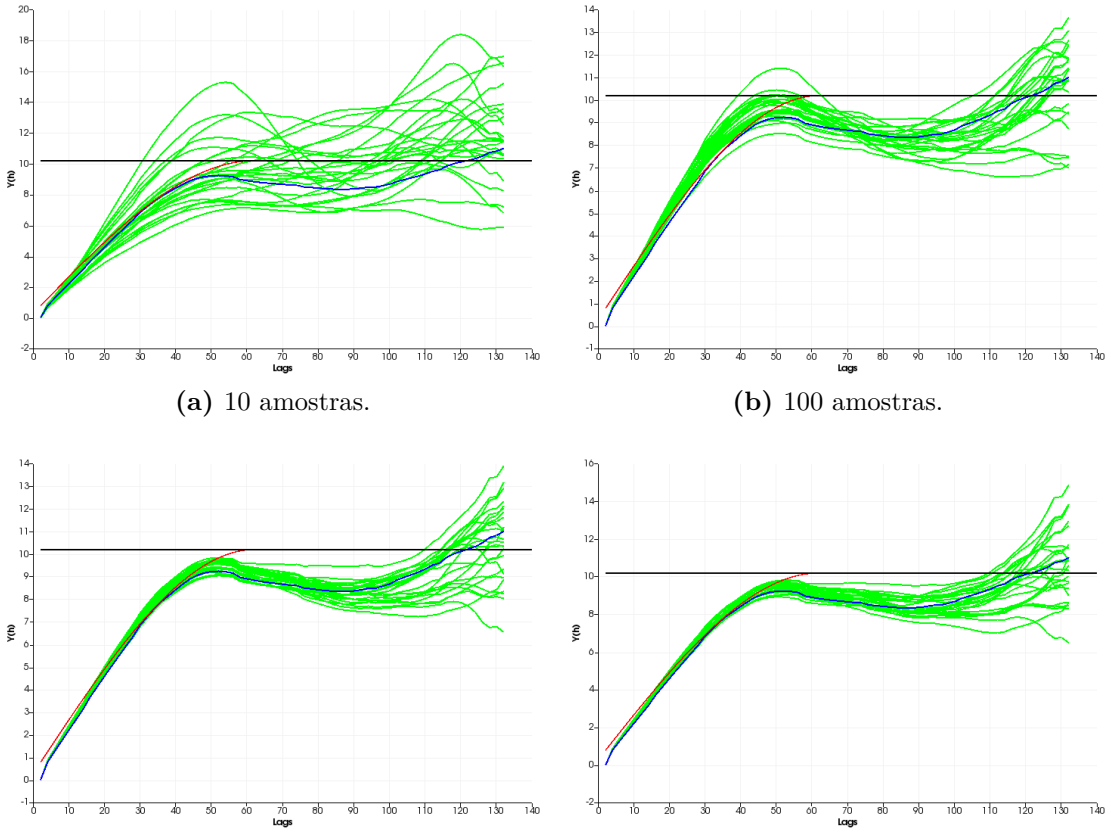
Figura 36 – Flutuação ergódica dos variogramas experimentais, na direção de maior continuidade, das simulações utilizando o simulador TBSIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em verde, têm-se os variogramas experimentais das simulações TBSIM, em azul têm-se o variograma experimental dos dados exaustivos de referência e em vermelho têm-se o variograma de referência.



(a) 10 amostras. **(b)** 100 amostras.

(c) 1000 amostras. **(d)** 5000 amostras.

Figura 37 – Flutuação ergódica dos variogramas experimentais, na direção de média continuidade, das simulações utilizando o simulador TBSIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em verde, têm-se os variogramas experimentais das simulações TBSIM, em azul têm-se o variograma experimental dos dados exaustivos de referência e em vermelho têm-se o variograma de referência.



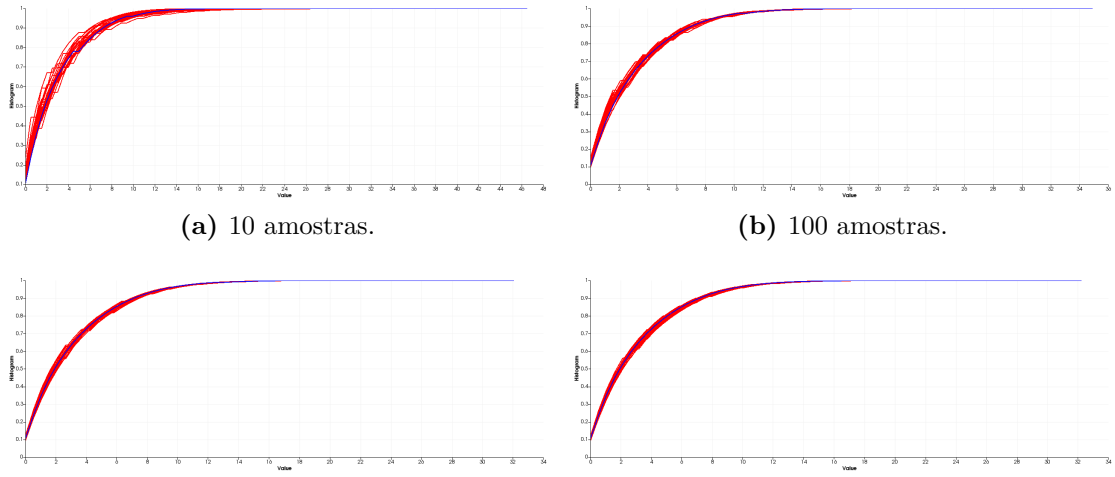
(a) 10 amostras.

(b) 100 amostras.

(c) 1000 amostras.

(d) 5000 amostras.

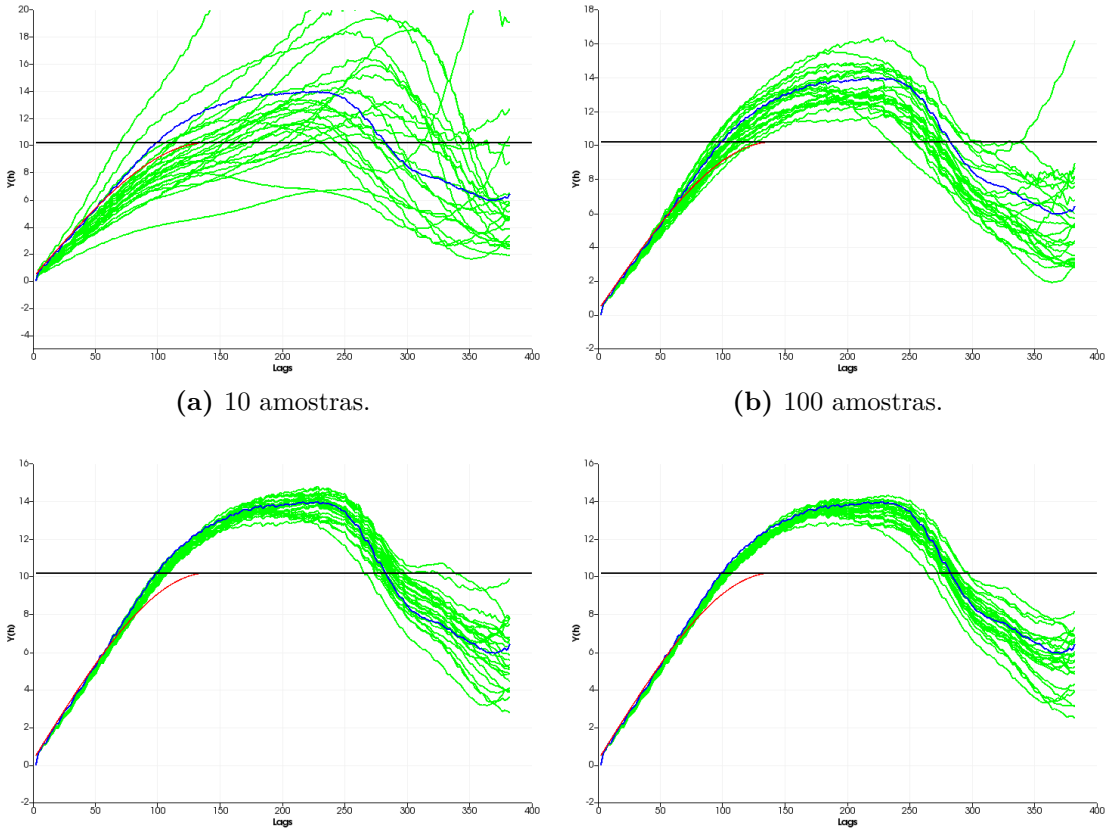
Figura 38 – Flutuação ergódica dos variogramas experimentais, na direção de menor continuidade, das simulações utilizando o simulador TBSIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em verde, têm-se os variogramas experimentais das simulações TBSIM, em azul têm-se o variograma experimental dos dados exaustivos de referência e em vermelho têm-se o variograma de referência.



(a) 10 amostras. **(b)** 100 amostras.

(c) 1000 amostras. **(d)** 5000 amostras.

Figura 39 – Flutuação ergódica dos histogramas das simulações utilizando o simulador TBSIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em vermelho, têm-se os histogramas das simulações TBSIM e, em azul, têm-se o histograma dos dados exaustivos de referência.



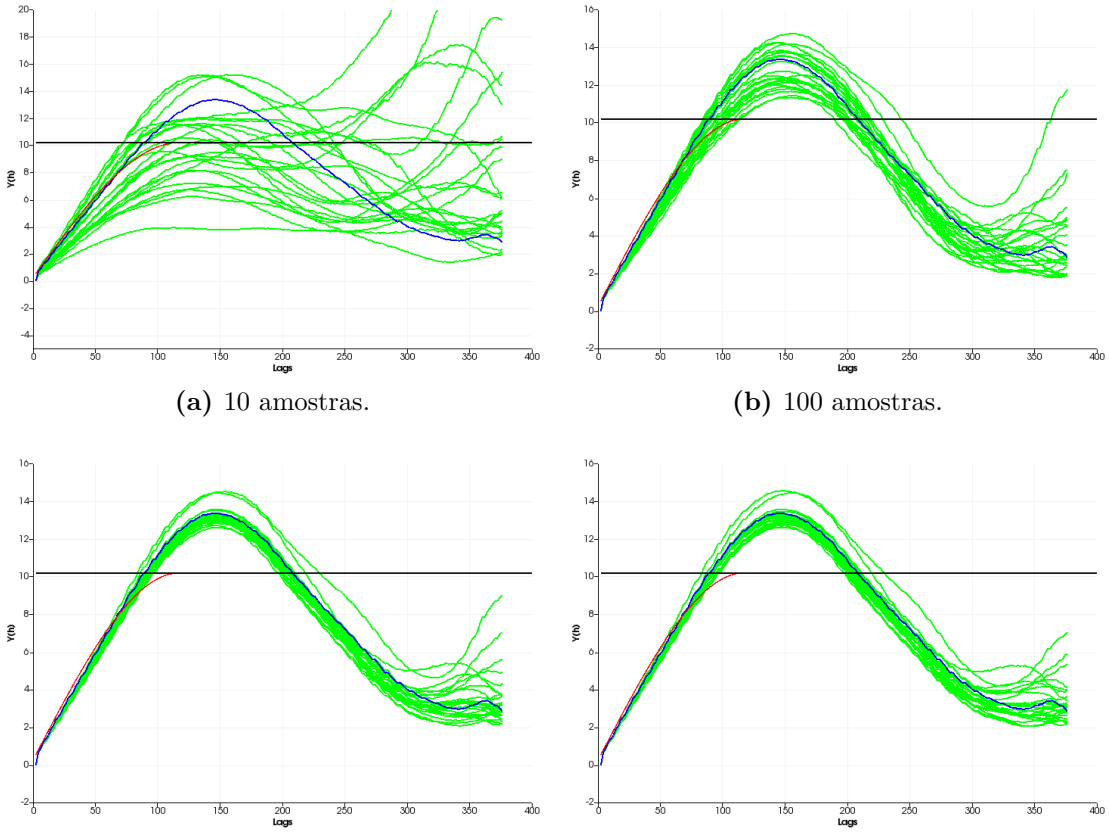
(a) 10 amostras.

(b) 100 amostras.

(c) 1000 amostras.

(d) 5000 amostras.

Figura 40 – Flutuação ergódica dos variogramas experimentais, na direção de maior continuidade, das simulações utilizando o simulador SGSIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em verde, têm-se os variogramas experimentais das simulações SGSIM, em azul têm-se o variograma experimental dos dados exaustivos de referência e em vermelho têm-se o variograma de referência.



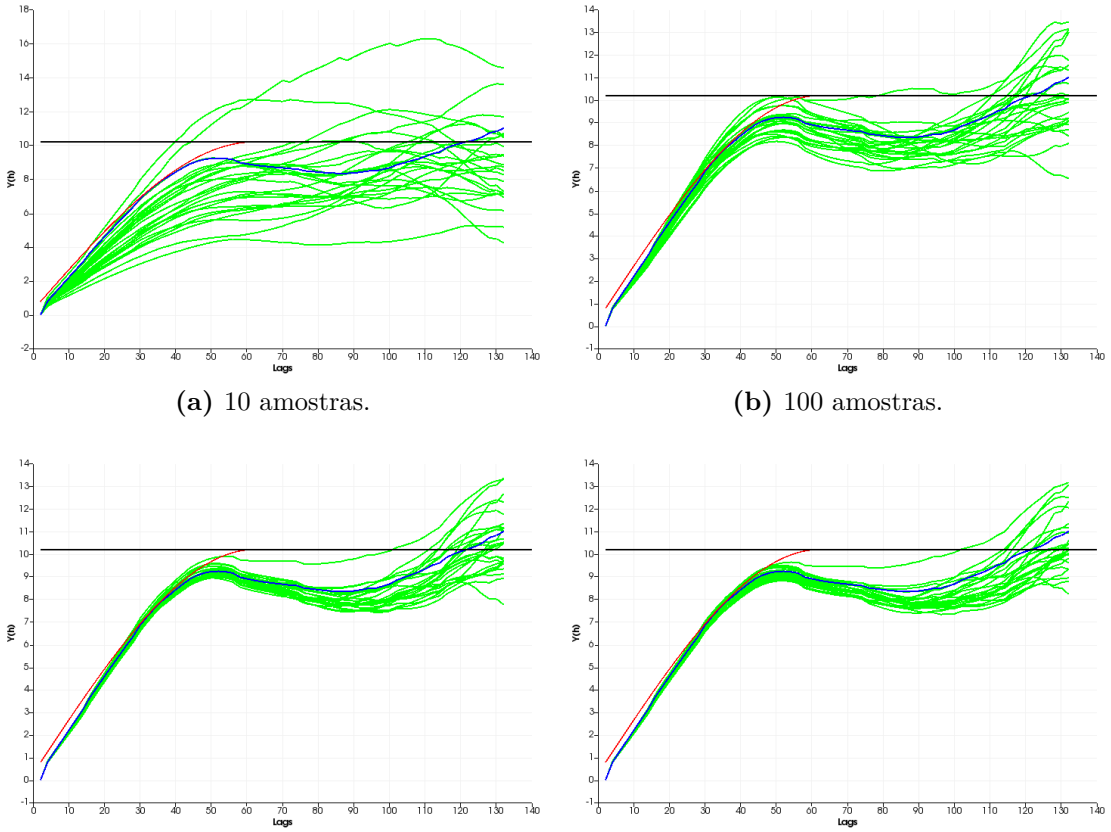
(a) 10 amostras.

(b) 100 amostras.

(c) 1000 amostras.

(d) 5000 amostras.

Figura 41 – Flutuação ergódica dos variogramas experimentais, na direção de média continuidade, das simulações utilizando o simulador SGSIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em verde, têm-se os variogramas experimentais das simulações SGSIM, em azul têm-se o variograma experimental dos dados exaustivos de referência e em vermelho têm-se o variograma de referência.



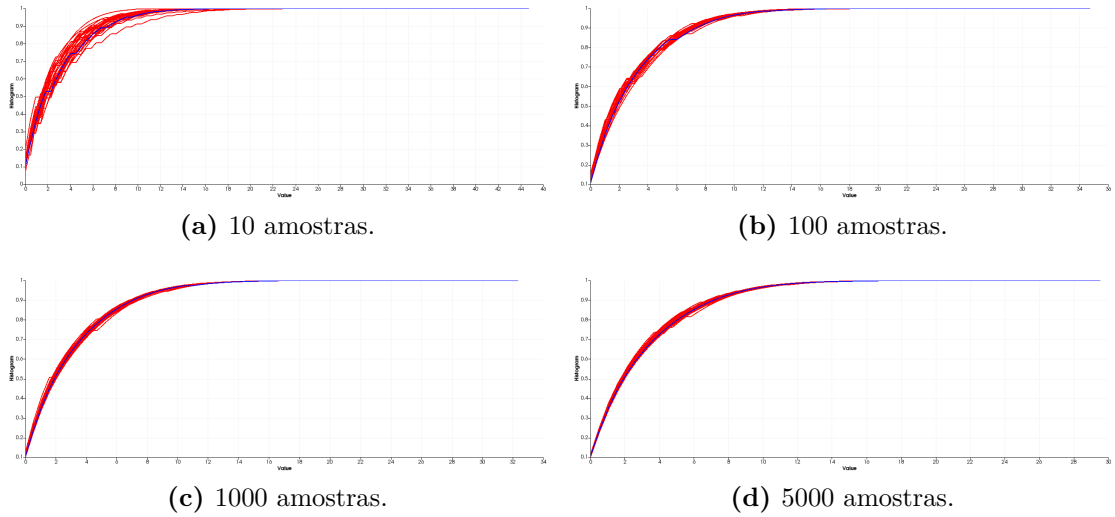
(a) 10 amostras.

(b) 100 amostras.

(c) 1000 amostras.

(d) 5000 amostras.

Figura 42 – Flutuação ergódica dos variogramas experimentais, na direção de menor continuidade, das simulações utilizando o simulador SGSIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em verde, têm-se os variogramas experimentais das simulações SGSIM, em azul têm-se o variograma experimental dos dados exaustivos de referência e em vermelho têm-se o variograma de referência.



(a) 10 amostras. **(b)** 100 amostras.

(c) 1000 amostras. **(d)** 5000 amostras.

Figura 43 – Flutuação ergódica dos histogramas das simulações utilizando o simulador SGSIM com condicionamento utilizando (a) 10 amostras, (b) 100 amostras, (c) 1000 amostras e (d) 5000 amostras. Em vermelho, têm-se os histogramas das simulações SGSIM e, em azul, têm-se o histograma dos dados exaustivos de referência.

4.5 Considerações Finais

A seção 4.4 mostrou a eficiência prática dos métodos de simulação espectral com relação ao clássico algoritmo de simulação sequencial gaussiana. Além de produzir resultados com qualidade similar, os métodos espectrais são trivialmente paralelizáveis e, por não necessitarem de um caminho aleatório, o condicionamento permite que os pesos de krigagem sejam reutilizados. Isso aumenta significativamente a eficiência do paralelismo e reduz o custo computacional, já que o condicionamento é a etapa mais cara do processo de simulação geoestatística.

Apesar de ser mais lento que o FIM, o TBSIM é uma excelente solução para simulação em casos gerais, onde a malha não é necessariamente regular. Além disso, o TBSIM pode ser utilizado em cenários de simulação *gridless*, permitindo que resultados sejam gerados conforme necessários, o que permite que só parte dos dados sejam mantidos na memória RAM. Isso permite que o TBSIM seja utilizado em simulação de larga escala, envolvendo bilhões de nós. Já o FIM, é uma execelente alternativa, em termos de qualidade de resultado e performance, para o problema rotineiro de simulação em malhas regulares.

O algoritmo de simulação sequencial, apesar de mais lento que as alternativas espectrais, é essencial para algoritmos que precisem incorporar informação local, quer seja via anisotropia local, quer seja via *ensemble* utilizando erro de validação cruzada. Conforme visto no Cap. 5, a simulação sequencial fornece uma estrutura algoritmica fundamental para obtenção de forma implícita de informações sobre a distribuição de erro local. Isso

permite a construção de algoritmos mais sofisticados que combinem uma boa reprodução de variogramas e um baixo erro local, reproduzindo com melhor precisão o espaço de incerteza associado ao modelo.

Espera-se que com os resultados desse capítulo, tenha-se mostrado que metodologias espectrais fornecem solução de alta performance e qualidade para o problema de simulação geoestatística.

5 Modelagem geoestatística usando aprendizado de máquina

A evolução da capacidade de processamento e armazenamento de dados viabilizou a implementação de algoritmos de predição que executam mais estágios de tratamento dos dados para calibração de parâmetros, melhorando a capacidade de realização de classificações ou estimativas das infraestruturas de computação moderna. Além disso, esses algoritmos são capazes de evoluir, utilizando novos dados para aprimorar seus parâmetros internos (quer seja pesos numa rede neural, fatores de forma de *Kernels*, pesos de nós vizinhos, entre outros tipos de parâmetros). Essa junção entre algoritmos de otimização de parâmetros e técnicas de regressão e classificação recebeu o nome de aprendizado de máquina (PEDREGOSA et al., 2011).

Obviamente, o desempenho dos modelos gerados por aprendizado de máquina ou mesmo via estatística uni- ou multivariada clássica está extremamente atrelada à qualidade dos dados fornecidos para a calibração do modelo de estimativa ou classificação $z(\boldsymbol{u})$ em um posição $\boldsymbol{u}$ no espaço de entrada [1]. Por isso, foram desenvolvidas técnicas para aprimorar o modo com que os dados são utilizados para "ensinar" o modelo $z(\boldsymbol{u})$ a partir de amostras $y(\boldsymbol{v})$ da realidade. Essas técnicas são divididas em duas principais classes: filtragem e validação cruzada (PYLE, 1999). A filtragem permite a eliminação ou redução do impacto de informação ruidosa que pode prejudicar a calibração dos parâmetros do modelo $z(\boldsymbol{u})$ (é comum, por exemplo, usar algoritmos de classificação para agrupar subconjuntos de amostras com similaridades entre si). Além disso, a validação cruzada é fundamental para que os parâmetros β do modelo $z(\boldsymbol{u};\beta)$ sejam otimizados (ASMUSSEN; GLYNN, 2011). Na validação cruzada, o conjunto de dados de amostras é dividido em duas partes: calibração e teste. O conjunto de calibração é utilizado por algum algoritmo de otimização para se obter os melhores parâmetros β. Já o conjunto de teste é utilizado para se obter uma medida empírica do erro esperado de estimativa do modelo. Além disso, por meio da validação cruzada evita-se que os modelos sofram de *overfitting*, isto é, sejam incapazes de prever com acurácia informações em localizações não amostradas durante o processo da calibração ou incorporem o ruído como informação do modelo.

Epistemologicamente, a validação cruzada pode ser vista como uma implementação prática do método cientifico. Nela, pode-se ver o processo de calibração como o processo de construção de uma teoria que explique um fenômeno observado a partir de amostras da realidade. Os parâmetros calibrados podem ser vistos como a "teoria" construída para se

[1] Em geoestatística $\boldsymbol{u}$ é uma posição no espaço 2D ($\mathbb{R}^2$) ou 3D ($\mathbb{R}^3$) ou uma combinação entre posição espacial e uma ou mais informações secundárias.

explicar o fenômeno. Tendo-se uma teoria inicial, ela precisa ser validada por meio de sua capacidade de prever e/ou explicar novas observações do fenômeno. Uma boa teoria possui uma boa margem de acerto a cada nova observação e deve ser ajustada para explicar novos eventos que aparecerem. Caso uma teoria não sobreviva ao processo constante de validação, ela é descartada e substituída por outra melhor. Esse processo é emulado na validação cruzada, onde se pode criar diferentes conjuntos de calibração e validação de modo que os parâmetros calibrados em um conjunto possam ser validados em um conjunto separado com uma configuração diferente. As validações permitem o descarte de parâmetros que, por mais que sejam ótimos para o conjunto de calibração, podem não ser capazes de generalizar o modelo $z(\boldsymbol{u})$, isto é, são incapazes de explicar amostras não utilizadas na calibração dos parâmetros. Esse processo é recursivo, podendo-se quebrar o conjunto de dados disponíveis em diferentes níveis para se ajustar parâmetros com diferentes complexidades. Por exemplo, no caso clássico da geoestatística, pode-se quebrar o conjunto de amostras em dados usados para computação dos pesos de krigagem, dados usados para calibração dos modelos de covariância, dados usados para calibrar as vizinhanças de busca. Como em geoestatística são comuns problemas com poucos dados disponíveis, esse particionamento pode ser feito de forma não disjunta, isto é dado dois conjuntos de calibração T_1 e T_2, não é necessário que $T_1 \cap T_2 = \emptyset$, basta que $T_1 \neq T_2$ (ou seja, basta que haja uma quantidade suficiente de amostras diferentes em cada conjunto). Esse processo de particionamento de conjunto de dados em k pares de conjuntos de calibração e teste é chamado de validação cruzada *k-Fold* (KOHAVI et al., 1995). Portanto, como é mostrado nas seções seguintes, por meio desse método é possível se obter estimativas mais precisas, uma boa estimativa do erro quadrático (ou variância local), além da redução do viés local.

O foco desse capítulo é a adaptação da metodologia de validação cruzada para se construir algoritmos de estimativa mais robustos e com menos dependência de imputação manual de parâmetros. Por isso, ao longo do texto, assume-se que as amostras já foram devidamente filtradas (isto é, ruídos e erros instrumentais foram removidos) e são representativas do problema sendo estudado. É mostrada na Seção 5.1 uma técnica de otimização de parâmetros baseada no particionamento do conjunto de calibração, o modelo gerado é baseado em *kernels* radiais $\Psi_\beta(\boldsymbol{u}, \boldsymbol{v})$ (em geoestatística, $\Psi_\beta(\boldsymbol{u}, \boldsymbol{v}) = C(\boldsymbol{u} - \boldsymbol{v}; \beta)$, um modelo de covariância com β sendo uma transformação anisotrópica). Esse procedimento apresentado pode ser extendido para outros tipos de modelos não lineares, alterando-se o algoritmo de otimização de parâmetros utilizado.

Como a quantificação da incerteza tem se tornado um problema fundamental em modelagem computacional moderna, é mostrado na Seção 5.5 um modo de incorporar no algoritmo de simulação sequencial à metodologia apresentada substituindo a variância de krigagem pelo erro computado empiricamente por validação cruzada (ASMUSSEN; GLYNN, 2011) e substituindo-se o estimador de krigagem pelo estimador construído através da técnica de particionamento de conjunto de calibração e otimização de erro

empírico apresentados nas Seções 5.1 e 5.2.

Outro problema que pode ser resolvido por meio dos estimadores baseados em validação cruzada é o problema de construção de modelos-base de covariância (KLOECKNER et al., 2019). Na seção 5.6, é mostrada uma aplicação do *ensemble simulator* (KLOECKNER et al., 2019), construído na seção 5.5, para se gerar um modelo base de onde podem ser extraídas tabelas de covariância. Essas tabelas podem ser usadas em algoritmos de simulação e estimativa geoestatística clássica, como pode ser visto no capítulo 3. Uma abordagem alternativa é mostrada na Seção 5.5, com um algoritmo para atualização sob demanda de tabelas de covariância experimentais e um modo de se obter matrizes de covariância locais para uma determinada vizinhança que podem ser utilizadas em krigagens e simulações, isto é, matrizes de covariância positiva definidas. A combinação da técnica de obtenção implícita de matrizes de covariância com a metodologia de *ensemble simulation* permite a criação de um algoritmo de simulação geoestatística que não necessita de uma definição explícita de modelos de covariância, conforme é visto na seção 5.5.

É importante lembrar que as metodologias apresentadas não são uma panaceia, isto é, não são técnicas que podem ser aplicadas sem maiores cuidados para se resolver qualquer problema. A qualidade dos resultados depende de um bom tratamento e processamento das amostras. Além disso, algoritmos de aprendizado de máquina não podem ser vistos como substitutos de algoritmos *clássicos* de geoestatística, muito pelo contrário, por meio das técnicas apresentadas nesse capítulo é possível utilizar algoritmos clássicos como *Estimadores-Base* que podem ter suas estimativas aprimoradas. Outro fator importante é que o conhecimento *a priori* do modelador exerce um papel-chave e pode reduzir significativamente o esforço computacional para se obter bons modelos, além de garantir o realismo dos modelos gerados. Por exemplo, o modelador pode criar manualmente um *Kernel* construído a partir de suas observações e conhecimento e usar esse *Kernel* como estimador base no processo de calibração via validação cruzada.

Ao fim do capítulo, têm-se alguns exemplos de aplicação dos novos algoritmos apresentados. Com o conjunto de metodologias apresentadas torna-se viável a criação de um novo pacote de soluções computacionais que necessitam de uma quantidade menor de parâmetros do modelador. Haja vista que uma parte significativa do tempo é gasto tentando-se obter bons modelos covariância e bons parâmetros de busca.

5.1 Utilizando validação cruzada para melhorar qualidade de modelos

Estimar empiricamente o erro de um modelo é etapa fundamental da modelagem moderna. Por isso, têm-se dedicado significativo esforço no desenvolvimento de metodologias para avaliação da qualidade dos modelos gerados computacionalmente. Uma metodologia

chave é a validação cruzada que consiste em dividir o conjunto de dados disponíveis em dois grupos: calibração (T_1) e teste (T_2). Os dados de calibração são usados na definição de parâmetros do modelo, que terá sua qualidade medida com base na capacidade desse modelo prever as informações presentes no conjunto de teste. O erro de teste é a medida da magnitude da diferença entre o modelo computado e as amostras da realidade não utilizadas na calibração.

Intuitivamente, com uma boa amostragem da realidade, espera-se que o erro de teste convirja para o erro real do modelo (ASMUSSEN; GLYNN, 2011). Um procedimento comum na rotina de modelagem é incorporar novas amostras obtidas ao modelo pré-existente por meio de uma recalibração. A cada nova amostra adicionada é esperado que o erro de teste se reduza. Esse processo de atualização de modelos é emulado em algoritmos de simulação sequencial, além disso, fornece uma importante inspiração para o desenvolvimento de novos algoritmos de simulação *data-driven*. A medida da proximidade entre o modelo e as amostras da realidade é feita por meio de funções de erro.

Funções de erro podem ser usadas para medir parâmetros de qualidade que um modelo precisa seguir para ser um representante realista do fenômeno observado. Por exemplo, caso um modelo precise satisfazer restrições econômicas ou físicas, a função erro precisa atribuir erros maiores para modelos que menos satisfazerem essas restrições. Como o foco deste texto está na apresentação de um algoritmo que utilize a validação cruzada e o conceito abstrato de erro para a construção de modelos, a única função de erro utilizada nos exemplos práticos é a de erro quadrático médio (EQM) (5.1)

$$EQM\{z(\boldsymbol{u}), T\} = \frac{1}{|T|}\sum_{\boldsymbol{v}\in T}(z(\boldsymbol{v}) - y(\boldsymbol{v}))^2, \tag{5.1}$$

onde $z(\boldsymbol{u})$ é o modelo calibrado, $y(\boldsymbol{v})$ é uma função que amostra a realidade (representando o conjunto de amostras), $\boldsymbol{u}$ é um ponto no espaço estudado (no caso 2D é um ponto em $\mathbb{R}^2$ e no caso 3D é um ponto em $\mathbb{R}^3$), $T = \{\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_{|T|}\}$ é o conjunto de localizações utilizadas para o teste do modelo e, por fim, $|T|$ é o tamanho do conjunto T.

O EQM é interessante por ser diferenciável e ser facilmente utilizado em processos de otimização para calibração de parâmetros. Tanto que, essa métrica é utilizada como função objetivo a ser otimizada em diversos estimadores lineares. O estimador de krigagem, inclusive, é construído utilizando-se uma solução ótima nessa métrica (GOOVAERTS, 1997).

Caso o modelo calibrado $z(\boldsymbol{u})$ seja definido pela equação (5.2):

$$z(\boldsymbol{u}) = \sum_{\boldsymbol{v}\in\Omega(\boldsymbol{u},T_1)} \Psi(\boldsymbol{u},\boldsymbol{v})\lambda_v + a, \tag{5.2}$$

onde $\Omega(\boldsymbol{u}, T_1) = \{\boldsymbol{v}_1, \boldsymbol{v}_2, ...\boldsymbol{v}_N\}$, $N = |\Omega(\boldsymbol{u}, T_1)|$, é a vizinhança do ponto $\boldsymbol{u}$ no conjunto de

calibração T_1, $\Psi(\boldsymbol{u}, \boldsymbol{v})$ é alguma função radial de $\boldsymbol{u}$ e $\boldsymbol{v}$, $\lambda_{\boldsymbol{v}}$ é o peso de $\boldsymbol{v}$ na estimativa e a é uma constante livre do estimador. $\Psi(\boldsymbol{u}, \boldsymbol{v})$ é conhecido como o *kernel* do estimador $z(\boldsymbol{u})$ (Em geoestatística, $\Psi(u, v) = C(\boldsymbol{u} - \boldsymbol{v})$). Se definirmos um conjunto $T_2 = \{\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_M\}$, onde $M = |T_2|$ é o tamanho do conjunto T_2, $T_1 \cup T_2 = T$ e $T_1 \cap T_2 = \emptyset$ (ou seja, o conjunto de calibração T é partido em duas partes disjuntas), podemos computar os pesos $\lambda_{\boldsymbol{v}}$ e o termo independente a por meio das sistema de equações lineares (5.3).

$$\begin{cases} \Psi(\boldsymbol{v}_1, \boldsymbol{w}_1)\lambda_1 + \Psi(\boldsymbol{v}_2, \boldsymbol{w}_1)\lambda_2 + ... + \Psi(\boldsymbol{v}_N, \boldsymbol{w}_1)\lambda_N + a = z(\boldsymbol{w}_1) \\ \Psi(\boldsymbol{v}_1, \boldsymbol{w}_2)\lambda_1 + \Psi(\boldsymbol{v}_2, \boldsymbol{w}_2)\lambda_2 + ... + \Psi(\boldsymbol{v}_N, \boldsymbol{w}_2)\lambda_N + a = z(\boldsymbol{w}_2) \\ \dots \\ \Psi(\boldsymbol{v}_1, \boldsymbol{w}_M)\lambda_1 + \Psi(\boldsymbol{v}_2, \boldsymbol{w}_M)\lambda_2 + ... + \Psi(\boldsymbol{v}_N, \boldsymbol{w}_M)\lambda_N + a = z(\boldsymbol{w}_M) \end{cases} . \tag{5.3}$$

O EQM da equação (5.2), com relação ao conjunto T_2 é dado por

$$EQM\{z(\boldsymbol{u}), T_2\} = \frac{1}{M} \sum_{\boldsymbol{w} \in T_2} (\Psi(\boldsymbol{v}_1, \boldsymbol{w})\lambda_1 + ... + \Psi(\boldsymbol{v}_N, \boldsymbol{w})\lambda_N + a - y(\boldsymbol{w}))^2. \tag{5.4}$$

A equação (5.4) pode ser otimizada resolvendo-se a equação (5.5)

$$\nabla EQM\{z(\boldsymbol{u}), T_2\} = \{\frac{\partial EQM}{\partial \lambda_1}, ..., \frac{\partial EQM}{\partial \lambda_N}, \frac{\partial EQM}{\partial a}\} = (0, 0, ..., 0). \tag{5.5}$$

Expandindo a equação (5.5) para cada peso λ_j, tem-se

$$\frac{\partial EQM}{\partial \lambda_j} = \frac{2}{M} \sum_{\boldsymbol{w} \in T_2} (\Psi(\boldsymbol{v}_1, \boldsymbol{w})\lambda_1 + ... + \Psi(\boldsymbol{v}_N, \boldsymbol{w})\lambda_N + a - y(\boldsymbol{w}))\Psi(\boldsymbol{v}_j, \boldsymbol{w}) = 0 \tag{5.6}$$

e

$$\frac{\partial EQM}{\partial a} = \frac{1}{M} \sum_{\boldsymbol{w} \in T_2} (\Psi(\boldsymbol{v}_1, \boldsymbol{w})\lambda_1 + ... + \Psi(\boldsymbol{v}_N, \boldsymbol{w})\lambda_N + a - y(\boldsymbol{w})) = 0, \tag{5.7}$$

com $j = 1, 2, ..., N$. Definindo as constantes $c_{j,k} = \frac{1}{M} \sum_{\boldsymbol{w} \in T_2} \Psi(\boldsymbol{v}_j, \boldsymbol{w})\Psi(\boldsymbol{v}_k, \boldsymbol{w})$ e $b_j = \frac{1}{M} \sum_{\boldsymbol{w} \in T_2} \Psi(\boldsymbol{v}_j, \boldsymbol{w})$, tem-se

$$b_j a + c_{j,1}\lambda_1 + ... + c_{j,N}\lambda_N = \frac{1}{M} \sum_{\boldsymbol{w} \in T_2} y(\boldsymbol{w})\Psi(\boldsymbol{v}_j, \boldsymbol{w}) \tag{5.8}$$

e

$$a + b_1\lambda_1 + ... + b_N\lambda_N = \frac{1}{M} \sum_{\boldsymbol{w} \in T_2} y(\boldsymbol{w}). \tag{5.9}$$

Como $c_{j,k}$ e b_j são constantes, as equações (5.8) e (5.9) podem ser resolvidas via mínimos quadrados. Essas equações são muito poderosas, já que os pesos computados são

a solução que minimiza o erro de validação para o conjunto T_2. Esse procedimento pode ser aplicado recursivamente. Por exemplo, o conjunto de calibração T pode ser quebrado em três partes: T_1 (usado para construir a vizinhança do estimador), T_2 para otimizar os pesos dos vizinhos e T_3 pode ser usado para otimizar um ou mais parâmetros não-lineares usados pelo estimador (como o parâmetro de forma de uma função de base radial ou a anisotropia local de uma função de covariância) (ROQUE; FERREIRA, 2010).

Suponha que $\Psi_\beta(\boldsymbol{u}, \boldsymbol{v})$ seja uma função radial definida por um fator de forma β e seja utilizado como *kernel* do estimador definido pela eq. (5.2). O EQM para o conjunto T_3 desse estimador é dado por

$$\begin{aligned} &EQM\{z(\boldsymbol{u}; \beta; T_2, T_1), T_3\} = \\ &\frac{1}{|T_3|}\{\sum_{\boldsymbol{w} \in T_3} (z(\boldsymbol{w}; \beta; T_2, T_1) - y(\boldsymbol{w}))^2\}, \end{aligned} \tag{5.10}$$

onde $z(\boldsymbol{u}; \beta; T_2, T_1)$ é um estimador com pesos ótimos e parâmetro de forma β, segundo a métrica EQM, computados a partir das equações (5.8) e (5.9). O conjunto T_2 é um subconjunto utilizado para calibração dos pesos e T_1 é o conjunto utilizado para se construir a vizinhança de dados utilizada pelo estimador. Por fim, T_3 é um subconjunto utilizado para se computar o parâmetro de forma β do *kernel* $\Psi_\beta(\boldsymbol{u}, \boldsymbol{v})$. Além disso, $T_1 \cap T_2 = \emptyset$, $T_1 \cap T_3 = \emptyset$, $T_2 \neq T_3$ e $T_1 \cup T_2 \cup T_3 = C$. Ou seja, o conjunto de dados de calibração T é quebrado em três conjuntos distintos e cada conjunto é usado em um passo de calibração diferente. Otimizando a eq. (5.10) segundo a variavel β tem-se:

$$\begin{aligned} &\frac{dEQM\{z(\boldsymbol{u}; \beta; T_2, T_1), T_3\}}{d\beta} = \\ &\frac{1}{|T_3|}\{\sum_{\boldsymbol{w} \in T_3} \frac{d[(z(\boldsymbol{w}; \beta; T_2, T_1) - y(\boldsymbol{w}))^2]}{d\beta}\}. \end{aligned} \tag{5.11}$$

Fazendo $z(\boldsymbol{u}; \beta; T_2, T_1) = \widehat{z}_\beta(\boldsymbol{u})$, $EQM\{\widehat{z}_\beta(\boldsymbol{u})\} = \epsilon(\widehat{z}_\beta)$, $\Psi_\beta = \Psi_\beta(\boldsymbol{u}, \boldsymbol{v})$ e $|T_3| = L$ para melhorar a notação da equação (5.11), tem-se

$$\frac{d\epsilon(\widehat{z}_\beta)}{d\beta} = \frac{1}{L}\{\sum_{\boldsymbol{w} \in T_3} \frac{d[(\widehat{z}_\beta(\boldsymbol{w}) - y(\boldsymbol{w}))^2]}{d\beta}\}. \tag{5.12}$$

Expandindo a equação (5.12)

$$\frac{d\epsilon(\widehat{z}_\beta)}{d\beta} = \frac{1}{L}\{\sum_{\boldsymbol{w} \in T_3} \frac{d[\widehat{z}_\beta(\boldsymbol{w})^2 - 2y(\boldsymbol{w})\widehat{z}_\beta(\boldsymbol{w}) - y(\boldsymbol{w})^2]}{d\beta}\}. \tag{5.13}$$

$$\frac{d\epsilon(\widehat{z}_\beta)}{d\beta} = \frac{1}{L}\{\sum_{\boldsymbol{w} \in T_3} \frac{-2d[y(\boldsymbol{w})\widehat{z}_\beta(\boldsymbol{w})]}{d\beta}\}. \tag{5.14}$$

$$\frac{d\epsilon(\widehat{z}_\beta)}{d\beta} = \frac{-2}{L}\{\sum_{\boldsymbol{w} \in T_3} y(\boldsymbol{w}) \frac{d[\widehat{z}_\beta(\boldsymbol{w})]}{d\beta}\}. \tag{5.15}$$

expandindo a eq. (5.15), tem-se

$$\frac{d\epsilon(\widehat{z}_\beta)}{d\beta} = -\frac{2}{L}\{\sum_{\boldsymbol{w}\in T_3} y(\boldsymbol{w})\frac{d[\Psi_\beta(\boldsymbol{v}_1,\boldsymbol{w})\lambda_1 + ... + \Psi_\beta(\boldsymbol{v}_N,\boldsymbol{w})\lambda_N + a]}{d\beta}\}. \tag{5.16}$$

Portanto, o parâmetro de forma ótimo β_{opt} é obtido a partir da solução do problema de otimização dado pela eq. (5.17), com β sendo a variável a ser otimizada:

$$\min \|\frac{2}{L}\{\sum_{\boldsymbol{w}\in T_3} y(\boldsymbol{w})\frac{d[\Psi_\beta(\boldsymbol{v}_1,\boldsymbol{w})\lambda_1 + ... + \Psi_\beta(\boldsymbol{v}_N,\boldsymbol{w})\lambda_N]}{d\beta}\}\|. \tag{5.17}$$

Uma solução da eq. (5.17) pode ser aproximada numericamente com algoritmos clássicos de busca de raizes baseados em gradientes (como o método de Newton (BEN-ISRAEL, 1966) e o método de Broyden, vistos na seção A.1), ou algoritmos mais avançados não dependentes de gradiente (como pesquisa Tabu (GLOVER, 1986), enxame de partículas (KENNEDY, 2010), entre outros). Então, para cada candidato a solução β', os pesos ótimos podem ser computados utilizando-se as equações (5.8) e (5.9). Caso $\Psi_{\boldsymbol{\beta}}(\boldsymbol{u},\boldsymbol{v})$ possua um parâmetro de forma $\boldsymbol{\beta}$ com mais de uma dimensão, isto é $\boldsymbol{\beta} = (\beta_1, ..., \beta_P)$, com P sendo o número de parâmetros de forma β_k, pode-se obter de forma similar um sistema de equações, via otimização do gradiente, cujas soluções serão os parâmetros de forma ótimos. Por se tratar de um problema numericamente bem complexo, pode-se utilizar métodos de otimização que não se baseiam em gradiente para se computar esse parâmetro de forma. Um método muito eficiente para resolver esse tipo de problema é a pesquisa Tabu, que é apresentada na seção 5.3. O principal desafio de otimização nesse tipo de problema são os mínimos locais que podem reduzir a qualidade do resultado da otimização, mas para evitar isso existem estratégias que são explicadas na seção 5.3.

Aplicando uma pesquisa em largura com uma fila de prioridade [2] para resolver o problema (5.17), obtém-se o algoritmo 3.

No algoritmo 3, $\delta(p)$ é a função de resfriamento utilizada no *simulated annealing* (LAARHOVEN; AARTS, 1987) e, geralmente, é uma função decrescente. $\delta(p^*+1)U(-1,1)$ pode ser substituído por um valor Δ_{p^*}, computado por um processo de otimização clássica, como o método de Newton ou Broyden (apresentados na Seção A.1). Os parâmetros internos do estimador $z_\beta(\boldsymbol{u};T_1)$ são obtidos otimizando-se as eqs. (5.8) e (5.9) e são recomputados toda vez que um novo valor da função objetivo $\epsilon(\beta)$ é avaliado. Esse algoritmo é uma implementação simplificada da pesquisa Tabu (GLOVER, 1986). Caso $\boldsymbol{\beta}$ seja multi-dimensional, isto é $\boldsymbol{\beta} = (\beta_1, ..., \beta_M)$, um vetor de distorção $\Delta = (1 + \delta_1(p)U(-1,1), ..., 1 + \delta_M(p)U(-1,1))$ deve ser utilizado. A grande vantagem da busca tabu é que ela lida com

[2] Busca em largura é um algoritmo importante em teoria de grafos, onde visita-se primeiro as arestas diretamente conectadas. Fila de prioridade é uma estrutura de dados que organiza os itens em ordem decrescente com base em um parâmetro de ponderamento.

Entrada: O modelo $z_\beta(\boldsymbol{u}; T_1) = a + \sum_{\boldsymbol{v} \in T_1} \Psi_\beta(\boldsymbol{v}, \boldsymbol{u})\lambda_{\boldsymbol{v}}$, onde T_1 é o conjunto de dados utilizado pelo interpolador radial e a é o *bias* do estimador. A função objetivo a ser otimizada $\epsilon(\beta) = \epsilon(\beta; T_3) = \| \frac{2}{|T_3|} \{ \sum_{\boldsymbol{w} \in T_3} y(\boldsymbol{w}) \frac{d[\Psi_\beta(\boldsymbol{v}_1, \boldsymbol{w})\lambda_1 + ... + \Psi_\beta(\boldsymbol{v}_N, \boldsymbol{w})\lambda_N]}{d\beta} \} \|$. T_3 é o conjunto de dados utilizado para se medir o erro do parâmetro β. E, por fim, T_2 é usado para se otimizar os pesos internos $\lambda_1, \lambda_2, ..., \lambda_N$ do estimador $z_\beta(\boldsymbol{u}, T_1)$. *max_steps* é o maximo número de iterações do algoritmo. K é o número de amostras iniciais. β_{min} e β_{max} são os limites do intervalo em que β é pertencente. $\delta(p)$ é a pertubação máxima aplicada em um ponto, essa função geralmente é decrescente e p é um número inteiro descrevendo a profundidade de β na busca. *n_branches* número de ramificações de cada pivô β^* na busca.

saída *O parâmetro de forma otimal* β'.
início
  $\beta' \leftarrow U(\beta_{min}, \beta_{max})$;
  $\epsilon_{min} \leftarrow \epsilon(\beta')$;
  $Q \leftarrow (\epsilon(\beta'), \beta', 0), (\epsilon(\beta_1, T_3), \beta_1, 0), ..., (\epsilon(\beta_K, T_3), \beta_K, 0)$, onde Q é uma fila de prioridade que ordena em ordem decrescente seus itens usando como peso principal o valor $\epsilon(\beta, T_3)$ e β_k, com $k = 1, ..., K$, é obtido a partir de uma amostragem aleatória de uma distribuição uniforme $U(\beta_{min}, \beta_{max})$ e, por fim, o terceiro parâmetro é a profundidade do valor;
  $n\_step \leftarrow 0$;
  para Q *não vazia e* $n\_step < max\_steps$ **faça**
    $(\epsilon^*, \beta^*, p^*) \leftarrow pop(Q)$, tire do topo da fila de prioridade o valor atual com menor erro ϵ^*;
    se $\epsilon^* > \epsilon_{min}$ **então**
      $(\epsilon_{min}, \beta') \leftarrow (\epsilon^*, \beta^*)$;
    fim
    para $i = 1...n\_branches$ **faça**
      $\beta^+ \leftarrow \beta^*(1 + \delta(p^* + 1)U(-1, 1))$;
      se $\beta^+ \in [\beta_{min}, \beta_{max}]$ *e* $\epsilon^+ < \epsilon^*$ **então**
        $Q \leftarrow Q \cup (\epsilon^+, \beta^+, p^* + 1)$;
      fim
    fim
    $n\_step \leftarrow n\_step + 1$;
  fim
  β' é o melhor parâmetro de forma encontrado na busca;
fim

Algoritmo 3: Algoritmo de otimização do parâmetro de forma β para o modelo linear definido pela eq. 5.2.

facilidade com problemas multidimensionais e apresenta convergência rápida, ou seja, com poucos passos é possível obter uma solução mínima local. Além disso, essa técnica pode ser vista como um meio de aprimorar a convergência de metodologias clássicas, já que se trata de um meta-algoritmo. O maior desafio, como em qualquer algoritmo de otimização, é criar estratégias para se evitar mínimos locais distantes do mínimo global. No algoritmo 3, a estratégia utilizada foi iniciar a fila de prioridade Q com amostras uniformemente retiradas do espaço de parâmetros. Outra estratégia é utilizar as sequências de Van der Corput (FREULON, 1994) apresentadas no capítulo 4 para se gerar sequências de valores que amostrem bem o espaço de parâmetros.

Nessa seção, foi apresentada a teoria básica para se construir estimadores a partir de validação cruzada. Na subseção 5.1.1, são apresentados os diferentes tipos de validação cruzada.

5.1.1 Tipos de validação cruzada

Existem diferentes modos de se particionar o conjunto de amostras para se calibrar um modelo (KOHAVI et al., 1995). Nessa subseção, é feito um sumário desses tipos de particionamento. Nos itens a seguir, é considerado que o conjunto de amostras será dividido em quatro partes: T_1 um conjunto usado para construir as vizinhanças de busca ao redor do ponto a ser estimado, T_2 conjunto utilizado para otimizar os pesos do estimador, T_3 conjunto utilizado para se computar o parâmetro de forma β e T para se medir o erro empírico de validação cruzada.

1. Validação cruzada exaustiva: nesse tipo de validação cruzada, o conjunto de amostras T é particionado de todos modos possíveis. Essa técnica só é viável caso o volume de dados for pequeno, já que o número de formas de particionar um conjunto com N amostras é 2^N. No caso de uso deste texto, o número de possibilidades é 4^N.

2. Validação cruzada "deixe p de fora" (*Leave-p-out cross-validation*): Ao invés de se explorar todas partições possíveis do conjunto, nesse método são construídas todas combinações de partição do conjunto de amostra em que p valores são deixados de fora e usados como conjunto T_1. É menos intenso computacionalmente que o caso da validação cruzada exaustiva, mas ainda assim são gerados C_p^N combinações de partições. O conjunto de calibração ainda precisa ser particionado recursivamente usando essa técnica, aumentando exponencialmente o número de possibilidades.

3. Validação cruzada "deixe 1 de fora" (*Leave-1-out cross-validation*) : Variação do "deixe p de fora" em que $p = 1$. Essa variação é menos intensa computacionalmente .

4. Validação cruzada *k-fold cross-validation*: Nessa variação da validação cruzada, o conjunto de amostras T é particionado k vezes em conjuntos de igual tamanho e

um valor é retido para validação e os outros $k-1$ são usados para calibração de parâmetros. O processo é repetido k vezes. Isso assegura que cada amostra seja usada para validação do modelo.

5. Validação cruzada *holdout*: O conjunto de amostras é particionado aleatoriamente em conjuntos de calibração e teste.

6. Validação com múltiplas sub-amostragens aleatórias: Aqui são criados k partições aleatórias do conjunto de amostras.

7. Validação aninhada: Esse é o tipo de validação descrito na seção anterior. Aqui, o conjunto de amostras é particionado utilizando algum dos métodos descritos anteriormente e para cada partição, um conjunto diferente é utilizado para se calibrar um conjunto de parâmetros.

Neste trabalho, o algoritmo de partição utilizado é o *holdout*, tendo em vista o tamanho geralmente reduzido do conjunto de amostras. É importante ressaltar que, como os algoritmos apresentados funcionam com diferentes tipos de particionamento, o principal impacto está na estabilidade e qualidade dos resultados gerados.

No Algoritmo 4 apresentado a seguir, os conjuntos T_k podem ser usados para construir técnicas de seleção de modelos. Pode-se notar que o conceito de validação cruzada possui uma natureza extremamente recursiva, já que todo conjunto de dados não utilizado para a calibração pode ser usado no processo de seleção de algum parâmetro de nível mais alto, como, por exemplo, o tipo de kernel $\Psi_\beta(\boldsymbol{u}, \boldsymbol{v})$ utilizado como função de base no estimador. Alternativamente, em meta-estimadores, ou seja, estimadores que usam outros estimadores como base, esse conjunto pode ser usado para selecionar os algoritmos de estimativa básicos que apresentem melhor performance no conjunto T. Esse tipo de otimização de alto nível, em que se busca o melhor tipo de algoritmo base, ou melhor anisotropia da vizinhança de busca, ou o tipo de filtragem a ser aplicada nos dados, ou ainda otimização de alguma função de transferência, é chamado de *Otimização hiper-paramétrica* (BERGSTRA; BENGIO, 2012).

É importante ressaltar que, independente da técnica de validação utilizada, sempre é necessário um tratamento nos dados usados para teste para assegurar que não ocorra viés de confirmação humano (SHI et al., 2010). Isto é, o modelo aparentar estar bem validado, mas isso só ocorrer porque o conjunto de testes possui correlação alta com o conjunto de dados usado na calibração. Por isso, uma técnica simples, mas muito importante, é reservar um subconjunto de dados representativo que nunca será usado no processo de calibração/validação, apenas para teste final de qualidade. Já que nesse trabalho, o conceito de validação cruzada é aplicado recursivamente para otimização hiper-paramétrica (isto é,

Entrada: Um conjunto de amostras T que será usado no treinamento do modelo. K número de partições desejada.

saída *Um conjunto de K partições $\{\{T_{1,1}, T_{2,1}, T_{3,1}, T_{4,1}\}, ..., \{T_{1,K}, T_{2,K}, T_{3,K}, T_{4,K}\}\}$. Onde $T_{1,k}$ são os conjuntos usados para construção das vizinhanças utilizadas pelo estimador, $T_{2,k}$ é o conjunto utilizado para otimização dos pesos do estimador, $T_{3,k}$, com $k = 1...K$ é utilizado para otimizar o parâmetro de forma β da função radial $\Psi_\beta(\boldsymbol{u}, \boldsymbol{v})$ utilizada pelo estimador e $T_{4,k}$ é o conjunto de teste utilizado para se computar o erro do estimador.*

início

  $P \leftarrow \emptyset$, P é o conjunto de partições gerado ;

  para $i = 1, ..., K$ **faça**

    $c \leftarrow \lfloor \frac{|T|}{3} \rfloor$, c é o número de itens de cada partição utilizada na otimização do modelo e $|T|$ é o tamanho do conjunto T. ;

    $t \leftarrow |T| - 3c$, t é o tamanho do conjunto de testes. ;

    $T_{4,i} \leftarrow \pi(|T|, t)$, π é uma função que sorteia t elementos aleatórios do conjunto T. ;

    $T_{1,i} \leftarrow \pi(T - T_{4,i}, c)$, π é uma função que sorteia c elementos aleatórios do conjunto $T - T_{4,i}$ (isto é, valores não presentes no conjunto T_4). ;

    $T_{2,i} \leftarrow \pi(T - T_{4,i} - T_{1,i}, c)$, π é uma função que sorteia c elementos aleatórios do conjunto $T - T_{4,i} - T_{1,i}$ (isto é, valores não presentes nos conjuntos T_4 e T_1). ;

    $T_{3,i} \leftarrow \pi(T - T_{4,i} - T_{1,i}, c)$, π é uma função que sorteia c elementos aleatórios do conjunto $T - T_{4,i} - T_{1,i}$ (isto é, valores não presentes nos conjuntos $T_{4,i}$ e $T_{1,i}$). ;

    $P \leftarrow P \cup \{T_{1,i}, T_{2,i}, T_{3,i}, T_{4,i}\}$;

  fim

  Retorne P ;

fim

Algoritmo 4: Algoritmo de particionamento do conjunto de amostras em K partições $\{T_{1,k}, T_{2,k}, T_{3,k}, T_{4,k}\}$.

nas otimizações de parâmetros de diferentes níveis - parâmetro de forma β e pesos λ_i de cada função $\Psi_\beta(\boldsymbol{u}, \boldsymbol{v})$), o risco de viés de confirmação humano é alto.

Tendo apresentado os tipos de validação cruzada e um procedimento para calibração construído a partir dessa metodologia, é preciso apresentar uma metodologia para se estimar erros baseada nessa técnica. Na próxima seção, é apresentado uma técnica baseada em *Jackknife* para se estimar o erro de um modelo. Além disso, essa técnica pode ser usada para se estimar a variância local de uma estimativa a partir de diferentes estimativas geradas pela otimização via validação cruzada. Esse procedimento é chave no algoritmo central desse capítulo: a simulação geoestatística usando *ensemble simulation*.

5.2 Computando estimativas de erro usando *Jackknife*

O estimador *Jackknife* (ASMUSSEN; GLYNN, 2011) é definido pelas eqs. (5.18) e (5.19).

$$\widehat{J}_{(r)} = Rz(\boldsymbol{u}_r) - (R-1)z(\frac{1}{R-1}\sum_{r' \neq r} \boldsymbol{u}_{r'}), \tag{5.18}$$

onde $z(\boldsymbol{u}_r)$ representa o valor estimado para o ponto central $\boldsymbol{u}_r$, deixando a amostra r fora do conjunto de calibração, R é o número de replicações do teste que serão realizados. Usando o procedimento de construção de conjuntos de calibração e teste descrito no algoritmo 4, $\boldsymbol{u}_r$ pode ser valores tomados no conjunto T_4.

$$\widehat{M}_{T_4} = \frac{1}{R}\sum_{r=1}^{R}\widehat{J}_{(r)}, \tag{5.19}$$

aqui $\widehat{M}_{T_4}$ é a média esperada para o modelo computada a partir de R amostras extraídas do conjunto de teste T_4. (ASMUSSEN; GLYNN, 2011) demonstram que o estimador definido em (5.19) é um estimador corretor de viés. Mais do que isso, é demonstrado que a variância computada utilizando a equação (5.20) converge para a variância local real dos dados σ^2.

$$\sigma_z^2 = \frac{1}{R-1}\sum_{r=1}^{R}(\widehat{J}_{(r)} - \frac{1}{R}\sum_{r'=1}^{R}\widehat{J}_{(r')})^2 \tag{5.20}$$

A utilização do *Jackknife* é importante para redução do impacto do viés local na qualidade das predições geradas. O custo computacional das eqs. (5.19) e (5.20), pode ser elevado, já que para cada teste é preciso recomputar os parâmetros do modelo excluindo a amostra selecionada para o teste. Então, por isso, (ASMUSSEN; GLYNN, 2011) sugerem que a variação apresentada na equação (5.21).

$$\widehat{J}_{(r)} = Nz(\boldsymbol{u}_r) - (N-1)z(\frac{1}{(N-1)K}\sum_{r'\notin\{nK+1,\ldots,(n+1)K\}}\boldsymbol{u}_{r'}), \tag{5.21}$$

onde K representa o tamanho do subconjunto de T_4 selecionado e $\{nK+1,\ldots,(n+1)K\}$ representa as amostras selecionadas para computação da média e variância. Além de melhorar a performance computacional, o estimador definido em (5.21) apresenta melhor estabilidade e maior capacidade de redução de viés da estimativa (ASMUSSEN; GLYNN, 2011).

Estimadores *Jackknife* fornecem um meio barato, estável e eficiente para fortalecer a capacidade de estimativa de estimadores "fracos". Neste trabalho, estimador "fraco" é um estimador que não incorpora um modelo de covariância, isto é, foca-se apenas na otimização do erro local sem preocupar-se com a minimização da variância do erro local. E, a combinação entre o estimador *Jackknife* usando estimadores "fracos" calibrados utilizando a metodologia apresentada na Seção 5.1 permite a criação de um estimador robusto e com quase nenhuma necessidade de interferência humana na calibração dos parâmetros. Já que internamente, os estimadores construídos na Seção 5.1 possuem uma estratégia para otimização dos parâmetros de forma internos.

Na Seção 5.3, o estimador *Jackknife* e a estratégia de calibração definida na Seção 5.1 são utilizados para melhorar a qualidade das estimativas obtidas pela krigagem.

5.3 Computando modelo de covariância utilizando pesquisa Tabu

A krigagem clássica pode ser vista como um modelo $z(\boldsymbol{u};\beta;T_1)$ em que a calibração de pesos é realizada em um único passo, utilizando um conjunto de amostras T_1, e $\Psi_\beta(\boldsymbol{u},\boldsymbol{v}) = C(\boldsymbol{u}-\boldsymbol{v};\boldsymbol{\beta})$ é uma função de covariância definida pelo usuário [3] com $\boldsymbol{\beta}$ sendo um vetor multidimensional representando as anisotropias aninhadas. Ou seja, $\boldsymbol{\beta} = \boldsymbol{\Gamma}_1, ..., \boldsymbol{\Gamma}_M$, onde $\boldsymbol{\Gamma}_k$ é uma par $(\boldsymbol{\Theta}_k, c_k(\boldsymbol{h}))$ formado por uma matriz de transformação anisotrópica $\boldsymbol{\Theta}_k$, com dimensão 3×3, e uma covariância unitária $c_k(h)$ (modelos esféricos, gaussianos e outros que podem ser encontrados em (PYRCZ; DEUTSCH, 2006)) e M é o número de transformações. A Tabela 4 apresenta uma lista de modelos de covariância mais usados.

Modelo	$c(r = \|\|\boldsymbol{h}\|\|)$, com $r \leq 1$
Esférico	$1 - 1.5r + 0.5r^3$
Gaussiano	e^{-3r^2}
Exponencial	e^{-3r}
Efeito pepita	1, para $r = 0$ e 0, para $r \neq 0$.

Tabela 4 – Lista de modelos de covariância mais usados.

A matriz de transformação anisotrópica $\boldsymbol{\Theta}_k$ pode ser obtida a partir da operação definida na eq. (5.22). A calibração dos pesos do estimador de krigagem é feita de forma trivial, resolvendo-se o sistema de equações dado em eq. (5.23) (GOOVAERTS, 1997), com um lagrangiano opcional L.

$$\begin{cases} \boldsymbol{\Theta}_k = S(r_{1,k}, r_{2,k}, r_{3,k})R(a_{1,k}, a_{2,k}, a_{3,k}) \\ S(r_{1,k}, r_{2,k}, r_{3,k}) = \begin{bmatrix} \frac{1}{r_{1,k}} & 0 & 0 \\ 0 & \frac{1}{r_{2,k}} & 0 \\ 0 & 0 & \frac{1}{r_{3,k}} \end{bmatrix}, R(a_{1,k}, a_{2,k}, a_{3,k}) = R_x(a_{1,k})R_y(a_{2,k})R_z(a_{3,k}) \\ R_z(a_{1,k}) = \begin{bmatrix} cos(a_{1,k}) & sen(a_{1,k}) & 0 \\ -sen(a_{1,k}) & cos(a_{1,k}) & 0 \\ 0 & 0 & 1 \end{bmatrix}, R_y(a_{2,k}) = \begin{bmatrix} cos(a_{2,k}) & 0 & -sen(a_{2,k}) \\ 0 & 1 & 0 \\ sen(a_{2,k}) & 0 & cos(a_{2,k}) \end{bmatrix}, \\ R_x(a_{3,k}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(a_{3,k}) & sen(a_{3,k}) \\ 0 & -sen(a_{3,k}) & cos(a_{3,k}), \end{bmatrix} \end{cases} \tag{5.22}$$

onde $a_{1,k}, a_{2,k}$ e $a_{3,k}$ são os ângulos de rotação e $r_{1,k}, r_{2,k}$ e $r_{3,k}$ são os fatores de escalas da transformação anisotrópica Γ_k. $S(r_{1,k}, r_{2,k}, r_{3,k})$ representa uma operação de escala e

[3] Lembrando que funções de covariância são uma classe customizada de funções radiais

$R(a_{1,k}, a_{2,k}, a_{3,k})$ é a rotação nos eixos z, y e x, respectivamente.

$$\begin{cases} C(\boldsymbol{v}_1, \boldsymbol{v}_1)\lambda_1 + ... + C(\boldsymbol{v}_1, \boldsymbol{v}_N)\lambda_N + L\lambda_{N+1} = C(\boldsymbol{u}, \boldsymbol{v}_1) \\ \vdots \\ C(\boldsymbol{v}_N, \boldsymbol{v}_1)\lambda_1 + ... + C(\boldsymbol{v}_N, \boldsymbol{v}_N)\lambda_N + L\lambda_{N+1} = C(\boldsymbol{u}, \boldsymbol{v}_N) \\ \lambda_1 + ... + \lambda_N = 1, \end{cases} \tag{5.23}$$

caso seja krigagem simples, a última equação e o termo à esquerda da igualdade são desnecessários e $L = 0$. Caso seja krigagem ordinária, todas equações são utilizadas.

Na prática, o maior desafio da krigagem é definir o melhor vetor de pares $(\boldsymbol{\Theta}_k, c_k(h))$ que minimize o erro EQM para um conjunto T_2. Esse problema é exatamente o problema resolvido na Seção 5.1. Mas, é preciso realizar uma adaptação no Algoritmo 3 para se realizar a busca tanto no espaço de covariância unitária $c_k(h)$ quanto no espaço de anisotropias $\boldsymbol{\Theta}_k$. Antes de apresentar a adaptação do algoritmo, é importante observar uma propriedade muito importante das sequências de Van Der Carput. Essas sequências podem ser usadas para se construir esferas aleatórias que podem ser utilizadas no processo de otimização de rotações e elipsoides. Mais do que isso, essas esferas podem reduzir significativamente o número de indivíduos iniciais necessários para se obter boa performance na otimização (essa mesma propriedade é explorada pelo *Turning Bands*, como pode ser visto no Capítulo 4). Uma população inicial representativa de ângulos e fatores de escala (raios do elipsoide de anisotropia) pode ser geradas a partir do conjunto de equações (5.24) e (5.25).

$$\begin{cases} u_n = \frac{a_0}{2} + \frac{a_1}{4} + ... + \frac{a_p}{2^{p+1}}; v_n = \frac{b_0}{3} + \frac{b_1}{9} + ... + \frac{b_p}{3^{q+1}}; w_n = \frac{c_0}{5} + \frac{c_1}{25} + ... + \frac{c_r}{5^{r+1}}; \\ n = a_p...a_2a_1a_0 = b_q...b_2b_1b_0 = c_r...c_2c_1c_0; \\ n = a_0 + 2a_1 + ... + 2^p a_p = b_0 + 3b_1 + ... + 3^q b_q = c_0 + 5c_1 + ... + 5^r c_r; \\ a_i = 0, 1, b_j = 0, 1, 2, c_k = 0, 1, 2, 3, 4; \\ a_{1,k}^{(n)} = 2\pi u_n, a_{2,k}^{(n)} = 2\pi v_n, a_{3,k}^{(n)} = 2\pi w_n, \end{cases} \tag{5.24}$$

a_p, b_q e c_r são os dígitos de n nas bases 2, 3 e 5, respectivamente. $a_{1,k}^{(n)}$, $a_{2,k}^{(n)}$ e $a_{3,k}^{(n)}$ são a n-ésima amostragem de ângulos de rotação usando números de Van Der Carput como gerador. A mesma metodologia pode ser usada para se gerar os raios do elipsoide de anisotropia:

$$\begin{cases} u_n^{(B)} = \frac{d_0}{B} + \frac{d_1}{B^2} + ... + \frac{d_p}{B^{p+1}}; \\ n = d_p^{(B)}...d_2^{(B)}d_1^{(B)}d_0^{(B)}; \\ n = d_0^{(B)} + Bd_1^{(B)} + ... + B^p d_p^{(B)}; \\ r_{1,k}^{(n)} = r_{max,1}u_n^{(7)}, r_{2,k}^{(n)} = r_{max,2}u_n^{(11)}, r_{3,k}^{(n)} = r_{max,3}u_n^{(13)}, \end{cases} \tag{5.25}$$

$u_n^{(B)}$ é o número de Van der Carput na base B. $r_{1,k}^{(n)}$, $r_{2,k}^{(n)}$ e $r_{3,k}^{(n)}$ são a n-ésima amostragem de raios do elipsoide anisotropia usando números de Van Der Carput como gerador. Por fim, $r_{max,1}$, $r_{max,2}$ e $r_{max,3}$ são os máximo valores para o raio de elipsoide. A transformação anisotrópica $\boldsymbol{\Theta}_k^{(n)}$ é definida como sendo $\boldsymbol{\Theta}_k^{(n)} = S(r_{1,k}^{(n)}, r_{2,k}^{(n)}, r_{3,k}^{(n)})R(a_{1,k}^{(n)}, a_{2,k}^{(n)}, a_{3,k}^{(n)}) = S^{(n)}R^{(n)}$, com S e R sendo as transformações de escala e rotação definidas em (5.22). O caso com múltiplas estruturas pode ser gerado a partir de amostras de combinações de valores $S^{(n)}$ e $R^{(n)}$. Isto é, para K estruturas, constrói-se uma amostra escolhendo-se K valores aleatórios p_i, com $i = 1, 2, ..., K$ e $p_i \geq p_{i-1}$ (para $i > 1$). Então, combina-se os valores aleatórios p_i para se construir o vetor de anisotropias $\boldsymbol{\beta}_k = \{\boldsymbol{\Theta}_k^{(p_1)}, \boldsymbol{\Theta}_k^{(p_2)}, ..., \boldsymbol{\Theta}_k^{(p_K)}\}$.

Tendo-se o algoritmo para se gerar fatores de forma $\boldsymbol{\beta}_k$ e definido as covariâncias unitárias $c_k(\boldsymbol{h})$, pode-se utilizar o algoritmo de otimização de busca Tabu. A sequencia de Van Der Carput, por gerar em cada amostra um valor mais distante possível dos valores amostados anteriormente, é um excelente modo de se construir populações iniciais para algoritmos de otimização estocástica. Tendo-se a população inicial, agora basta adaptar o Algoritmo 3. A adaptação é trivial nesse caso, já que basta aplicar uma distorção controlada para cada raio $r_{i,k}$ e ângulo $a_{i,k}$, com $i = 1, 2, 3$ que definem a k-ésima transformação anisotrópica do vetor de anisotropia $\boldsymbol{\beta}_k$:

$$\begin{cases} r_{i,k}^* = \min(r_{min,i}, \max(r_{max,i}, r_{i,k} + U(-\Delta_{r,i,k}, \Delta_{r,i,k}))) \\ a_{i,k}^* = \{2\pi + a_{i,k} + U(-\Delta_{a,i,k}, \Delta_{a,i,k})\} \mod 2\pi, \end{cases} \tag{5.26}$$

onde $U(a, b)$ é um distribuição uniforme no intervalo $[a, b)$, $\Delta_{r,i,k}$ é a distorção máxima do raio $r_{i,k}$ e $\Delta_{a,i,k}$ é a distorção máxima para o ângulo $a_{i,k}$. Os valores $r_{i,k}^*$ e $a_{i,k}^*$ são usados para se construir uma novo vetor de anisotropias $\boldsymbol{\beta}_k^*$. O erro $\epsilon(\boldsymbol{\beta}_k^*)$ é computado usando-se um conjunto de teste T. O restante do algoritmo é exatamente igual ao definido no Algoritmo 3. É importante notar que essas distorções aplicadas a $r_{i,k}$ e $\alpha_{i,k}$ podem, também, ser computadas utilizando-se os métodos iterativos apresentados na Seção A.1. Aliás, a combinação entre uma distorção aleatória e uma distorção computada por otimização numérica clássica fornece uma solução robusta em relação a mínimos locais e, inclusive, permite uma aceleração de convergência da técnica.

A covariância $C(\boldsymbol{h}; \boldsymbol{\beta}_k)$ é escrita como uma combinação linear $C(\boldsymbol{h}; \boldsymbol{\beta}_k) = \eta + \sum_{k=1}^{K} b_k C_k(\boldsymbol{h}; \boldsymbol{\Theta}_k)$, onde b_k é a contribuição da covariância $C_k(\boldsymbol{h}; \boldsymbol{\Theta}_k) = c_k(\boldsymbol{\Theta}_k(\boldsymbol{h}))$, η é o efeito pepita e K é o número de estruturas. A partir de amostras do variograma experimental γ_i, com $i = 1...M$ e M sendo o número de amostras, pode-se computar os

valores de η e b_k usando-se o sistema de equações (5.27):

$$\begin{cases} \delta_T^2 - b_1 C_1(\boldsymbol{h}_1;\boldsymbol{\Theta}_1) - b_2 C_2(\boldsymbol{h}_1;\boldsymbol{\Theta}_2) - ... - b_K C_K(\boldsymbol{h}_1;\boldsymbol{\Theta}_K) = \gamma_1 \\ \vdots \\ \delta_T^2 - b_1 C_1(\boldsymbol{h}_M;\boldsymbol{\Theta}_1) - b_2 C_2(\boldsymbol{h}_M;\boldsymbol{\Theta}_2) - ... - b_K C_K(\boldsymbol{h}_M;\boldsymbol{\Theta}_K) = \gamma_M \\ \eta + b_1 + ... + b_K - \delta_T^2 = 0 \end{cases} \tag{5.27}$$

δ_T^2 é o *total sill* do modelo, η é o efeito pepita do modelo. $\boldsymbol{h}_i$ é a posição da amostra de variograma γ_i. O sistema de equações (5.27) pode ser resolvido com mínimos quadrados. Esse procedimento é importante, porque reduz a complexidade da otimização, já que a busca se restringirá às estruturas de anisotropia.

O processo de computação das contribuições b_i pode ser aprimorado utilizando o procedimento iterativo definido no Algoritmo 5.

Entrada: Amostras de variograma γ_i, $i = 1, 2, ..., M$, M é o número de amostras de variograma γ_i, P é o número de iterações para se computar as melhores contribuições e α é a tolerância. Uma métrica de erro ϵ.

1 **saída** *O melhor conjunto de pesos* b_k, $k = 1, 2, ..., K$, K *é o número de estruturas, efeito pepita* η *e total sill* δ_T^2.
2 **início**
3 $\boldsymbol{\gamma}^{(1)} \leftarrow \{\gamma_1, \gamma_2, ..., \gamma_M\}$;
4 **para** $p = 1, ..., P-1$ **faça**
5 Compute as contribuições $b_k^{(p)}$, $k = 1, 2, ..., K$, $\eta^{(p)}$ usando o sistema de equações (5.27) e o conjunto $\boldsymbol{\gamma}^{(p)}$;
6 Compute o erro médio do modelo $C^{(p)}(\boldsymbol{h};\boldsymbol{\Theta}_k)$ usando a métrica ϵ, obtendo-se o valor de erro $\epsilon^{(p)}$;
7 $\boldsymbol{\gamma}^{(p+1)} \leftarrow \{\gamma_i : \epsilon(C^{(p)}(\boldsymbol{h}_i;\boldsymbol{\Theta}_k), \gamma_i) < (1+\alpha)\epsilon^{(p)}, i = 1, 2, ..., M\}$;
8 **fim**
9 Retorne $b_1^{(P)}, b_2^{(P)}, ..., b_K^{(P)}$, $\eta^{(p)}$ e δ_T^2 ;
10 **fim**

Algoritmo 5: Algoritmo para computação de contribuições b_k, efeito pepita η e *total sill* δ_T^2.

O Algoritmo 5 realiza internamente uma filtragem das amostras removendo aquelas que possuem erro muito elevado (forte indicativo de que trata-se de um *outlier*). Empiricamente, observa-se que para se obter bons resultados um valor P entre 3 e 5 é suficiente. Caso o banco de amostras apresente baixo ruído, um valor de $P < 3$ pode ser suficiente. O parâmetro P está extremamente atrelado à qualidade das amostras.

As técnicas apresentas são usadas nas seções seguintes como ferramentas auxiliares na construções de estimadores base da *ensemble simulation*. Mais precisamente, a computação de modelos ótimos a partir de dados é uma técnica importante para krigagem sem definição explícita de modelos de covariância. Além disso, essa técnica permite o desenvolvimento de modelos iterativos que agregam novas estimações como dados para

recalibração dos parâmetros dos estimadores. Esse processo iterativo é descrito na Seção 5.5.

5.4 *Ensemble simulation*: usando validação cruzada para aprimorar a simulação sequencial

As seções anteriores apresentam técnicas para se estimar o erro empírico e de particionamento dos dados amostrais para a obtenção de estimativas melhores. Além disso, o algoritmo de busca Tabu é apresentado e aplicado na computação de parâmetro de forma ótimo (no caso geoestatístico, anisotropias do modelo de covariância). Combinando-se otimização de parâmetros com particionamento das amostras para computação mais realista de erros empíricos, pode-se obter estimadores mais poderosos que os obtidos tradicionalmente a partir de ajuste manual de parâmetros de forma (anisotropias) e utilizando-se o conjunto de dados inteiro em um único passo.

Nessa Seção, é mostrada a construção de um simulador equivalente à simulação sequencial gaussiana em que a variância de krigagem e a estimativa são obtidas através da combinação de um conjunto de estimativas iniciais, feitas por um conjunto de estimadores base, nas partições dos conjuntos de amostras. A combinação é feita por meio de uma soma ponderada que utiliza o inverso do erro de validação cruzada do modelo como peso. Esse simulador é o *ensemble simulation*. As estimativas computadas nas partições das amostras utiliza um estimador por krigagem baseado no modelo de covariância ótimo $C(\boldsymbol{h};\boldsymbol{\beta})$ definido na Seção 5.3.

Por uma questão de simplicidade, nesta Seção, assume-se que após definido o estimador $z(\boldsymbol{u})$, este será imutável. Mas, nada impede que o novo valor estimado seja reinserido no conjunto de amostras e seja usado para se melhorar o estimador. Na Seção 5.5, é apresentado esse procedimento de aprimoramento do estimador a partir da recalibração de seu parâmetro de forma (modelo de covariância).

Para se construir o algoritmo de *ensemble simulation*, primeiramente, constrói-se um caminho aleatório π, usando-se os nós $\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_N$ da grade alvo g. A seguir, particiona-se k vezes um conjunto de amostras $T^{(i)}$ em duas partes $T_1^{(k,i)}$ e $T_2^{(k,i)}$, no passo i do caminho aleatório, e as equações (5.28) e (5.29) são usadas para se computar a estimativa m_{π_i} e o erro empírico e_{π_i} no ponto $\boldsymbol{u}_{\pi_i}$. π_i representa o i-ésimo passo do caminho aleatório π.

$$m_{\pi_i}^{(k)} = \sum_{\boldsymbol{v}\in\Omega(\boldsymbol{u}_{\pi_i};T_1^{(k,i)})} \lambda(\boldsymbol{v})y(\boldsymbol{v}) \tag{5.28}$$

e

$$e_{\pi_i}^{(k)} = \frac{1}{2} \sum_{\boldsymbol{v} \in \Omega(\boldsymbol{u}_{\pi_i}; T_2^{(k,i)})} \lambda(\boldsymbol{v}) C(\boldsymbol{u}_{\boldsymbol{\pi_i}} - \boldsymbol{v}), \tag{5.29}$$

onde $\Omega(\boldsymbol{u}_{\pi_i}; T^{(i)})$, $T^{(i)} = T_1^{(k,i)}, T_2^{(k,i)}$ é a vizinhança de $\boldsymbol{u}_{\pi_i}$, $y(\boldsymbol{v})$ é o valor da amostra do entorno de $\boldsymbol{u}_{\pi_i}$, $z(\boldsymbol{v})$ é o valor estimado no ponto $\boldsymbol{v}$ usando os pesos $\lambda(\boldsymbol{v})$ e $C(\boldsymbol{h})$ é o modelo de covariância ótimo. $\lambda(\boldsymbol{v})$ são os pesos de krigagem computados a partir da vizinhança $\boldsymbol{u}_{\pi_i}$, ponto central u_{π_i} e modelo de covariância $C(\boldsymbol{h})$ (isto é, a variância de krigagem é utilizada como estimativa do erro na partição).

Os estimadores (5.28) e (5.29) são uma adaptação do estimador *Jackknife* apresentado na Seção 5.2 e, portanto, esse par de estimadores apresenta as mesmas propriedades. Em um dado passo i, esse procedimento de computação de estimativa m_{π_i} e e_{π_i} é repetido K vezes para diferentes partições $T_1^{(k,i)}$ e $T_2^{(k,i)}$ das amostras $T^{(i)}$. No fim, é computada a média $m_{\pi_i}^{(K)} = \frac{\sum_{k=1}^{K} m_{\pi_i}^{(k)}}{K}$ e $e_{\pi_i}^{(K)} = \frac{\sum_{k=1}^{K} e_{\pi_i}^{(k)}}{K}$, onde $m_{\pi_i}^{(k)}$ e $e_{\pi_i}^{(k)}$ são a média e o erro computados usando-se as k-ésimas partições $T_1^{(k,i)}$ e $T_2^{(k,i)}$, respectivamente, e as eqs. (5.28) e (5.29). É importante lembrar que $T_1^{(k,i)}$ e $T_2^{(k,i)}$ são conjuntos disjuntos, isto é, $T_1^{(k,i)} \cap T_2^{(k,i)} = \emptyset$ e $T_1^{(k,i)} \cup T_2^{(k,i)} = T^{(i)}$.

Conforme visto na Seção 5.2, $m_{\pi_i}^{(K)}$ convergirá para uma estimativa da média não-enviesada e $e_{\pi_i}^{(K)}$ convergirá para a variância do erro quadrático local. Se T for um conjunto de dados gaussianos, pode-se simular um valor $z(\boldsymbol{u}_{\pi_i})$ utilizando-se a distribução gaussiana $G(m^{(k)}(\boldsymbol{u}_{\pi_i}), e^{(k)}(\boldsymbol{u}_{\pi_i}))$. O valor $\boldsymbol{u}_{\pi_i}$ é usado para se criar o conjunto de amostras $T^{(i+1)} = T^{(i)} \cup \{\boldsymbol{u}_{\pi_i}\}$ e $y(\boldsymbol{u}_{\pi_i}) = z(\boldsymbol{u}_{\pi_i})$. Isso fecha o processo de simulação sequencial. Esse processo de simulação é realizado para todos nós π_i do caminho aleatório π.

Por sua vez, uma partição $T_3^{(k,i)}$ do conjunto $T^{(i)}$ pode ser usada para otimização dos parâmetros do estimador $z(\boldsymbol{u})$. Esse processo melhora o algoritmo visando permitir uma melhor reprodução tanto do histograma quanto do variograma. O Algoritmo 6 apresenta a descrição completa do procedimento descrito nesta seção.

É importante ressaltar que a etapa de combinação das médias e erros realizada no Algoritmo 6, para construção da distribuição, pode ser feita utilizando-se média ponderada usando o erro como peso. Além disso, nesse estágio pode ser realizada a eliminação de *outliers* via filtragem. A agregação é um estágio chave, onde diversas estratégias de seleção de estimativas podem ser realizadas para se selecionar valores que melhor adiram à função objetivo estudada (que pode ser uma equação física, exemplo).

O fato dos erros serem computados via validação cruzada permite que processos de otimização e seleção local de diversos parâmetros seja realizada, reduzindo significativamente o número de simulações necessárias para se obter soluções que satisfaçam funções objetivo definidas pelo usuário. A etapa de estimativa pode ser realizada no espaço original

Entrada: Conjunto de amostras T no espaço gaussiano, grade de simulação g e um caminho aleatório π definido em g. $|g|$ é o tamanho da grade g. K é o número de partições por passo.

saída *Simulação* $z(\boldsymbol{u})$.
início
  $T^{(0)} \leftarrow T$;
  para $i = 1, ..., |g|$ **faça**
    para $k = 1, ..., K$ **faça**
      Crie partições $T_1^{(k,i)}$, $T_2^{(k,i)}$ a partir de $T^{(i)}$. ;
      Opcionalmente, pode-se definir uma partição $T_3^{(k,i)}$ para otimização dos parâmetros do estimador ;
      Compute $m_{\pi_i}^{(k)}$ usando a eq. (5.28). ;
      Compute $e_{\pi_i}^{(k)}$ usando a eq. (5.29). ;
    fim
    $m(\boldsymbol{u}_{\pi_i}) \leftarrow \dfrac{\sum_{k=1}^{K} m_{\pi_i}^{(k)}}{K}$. ;
    $e(\boldsymbol{u}_{\pi_i}) \leftarrow \dfrac{\sum_{k=1}^{K} e_{\pi_i}^{(k)}}{K}$. ;
    Amostre $z(\boldsymbol{u}_{\pi_i})$ usando a distribuição gaussiana $G(m(\boldsymbol{u}_{\pi_i}), e(\boldsymbol{u}_{\pi_i}))$. ;
    $T^{(i+1)} \leftarrow T^{(i)} \cup \{\boldsymbol{u}_{\pi_i}\}$.;
  fim
  Retorne $z(\boldsymbol{u}_{\pi_i})$. ;
fim

Algoritmo 6: Algoritmo de simulação *ensemble*.

dos dados e a distribuição gaussiana ser construída somente para a amostragem de $z(\boldsymbol{u})$. Isso define a simulação direta *ensemble*.

Na próxima seção, é mostrada uma variação do algoritmo de simulação *ensemble* em que o modelo de covariância é atualizado a cada etapa do processo de simulação e no fim do processo obtém-se, além da simulação, um modelo de covariância ótimo baseado nos dados.

5.5 Simulação sequencial evolutiva: combinando matrizes de covariância computadas implicitamente e *ensemble simulation*

As seções anteriores apresentam um conjunto de metodologias para se gerar estimativas mais robustas baseadas em dados. Além disso, é mostrado um algoritmo baseado em sequências de Van Der Carput para se otimizar as anisotropias associadas ao modelo. A combinação dessas estratégias permite a construção de uma nova geração de estimadores com parâmetros otimizados pela validação cruzada que não necessitam de definição explícita de parâmetros complexos como modelos de covariância.

Nesta seção, é apresentado um algoritmo que constrói implicitamente os modelos de covariância e permite a evolução de modelos iniciais quer seja obtidos a partir de conhecimento *a priori* do modelador, quer sejam obtidos a partir de algoritmos de otimização como o apresentado na Seção 5.3. A ideia-chave é utilizar a autocorreção possível graças à validação cruzada para ajustar a cada iteração o modelo de covariância atual de forma que o erro seja minimizado. Esse processo pode ser feito após o passo de amostragem da distribuição $G(m(\boldsymbol{u}_{\pi_i}), e(\boldsymbol{u}_{\pi_i}))$ construída no Algoritmo 6, definido na Seção 5.4.

A simulação sequencial e *ensemble* emula o processo de obtenção de novas amostras da realidade. A cada estágio da simulação, o conjunto de dados converge para uma representação que possui distribuição e variância espacial mais próxima da distribuição e variância espacial alvos. O processo de validação cruzada permite, em cada estágio da simulação, uma melhor aproximação do erro real, que também pode ser visto como variância local (caso o erro seja o erro quadrático médio). Então, naturalmente, esse novo erro pode ser usado como referência para ajustar o modelo de covariância atual. Baseado nisso, uma forma natural de aprimorar a qualidade da estimativa e da simulação é, após realizar algumas simulações de pontos, realizar o processo de otimização de modelo de covariância descrito na Seção 5.3, utilizando um conjunto T_c amostrado aleatoriamente do conjunto formado pelos nós previamente simulados e pelas amostras. O conjunto T_c é usado para se computar uma nova versão do modelo de covariância. T_c pode ser exatamente a união dos conjuntos de nós previamente simulados e de amostras. Embora usar todos nós simulados e amostras gere um modelo melhor, isso pode aumentar proibitivamente o custo computacional do processo de simulação.

No Algoritmo 7, como é fornecido um modelo inicial de covariância, ao invés de um ponderador mais simples, pode-se usar diretamente a krigagem para se computar as estimativas. Nesta variação do algoritmo de simulação *ensemble*, é preciso fornecer três parâmetros extras: o tamanho máximo do conjunto T_c usado para otimização do modelo de covariância, de quantos em quantos passos a otimização do modelo covariância será realizada e, por fim, o modelo de covariância inicial C_0. A otimização do modelo de covariância pode ser realizada usando qualquer algoritmo de otimização de parâmetros. Neste trabalho, o algoritmo usado é o descrito na Seção 5.3. Uma variação mais rápida computacionalmente, mas que pode apresentar uma qualidade inferior, seria utilizar somente o passo de otimização do modelo corrente, o que é feito por meio da aplicação de distorções aleatórias nesse modelo (ou distorções computadas utilizando-se algoritmos de otimização clássicos como os vistos na Seção A.1) e selecionando-se a versão que minimiza o erro em T_c. O estágio de otimização local rápida é descrito no Algoritmo 8.

Entrada: Conjunto de amostras T no espaço gaussiano, grade de simulação g e um caminho aleatório π definido em g. $|g|$ é o tamanho da grade g. K é o número de partições por passo. M_c é o tamanho máximo do conjunto usado para calibrar o modelo de covariância. P_c indica de quantos em quantos passos a otimização do modelo de covariância será realizada. C_0 é o modelo de covariância inicial.

saída *Simulação* $z(\boldsymbol{u})$.
início
  $T^{(0)} \leftarrow T$;
  para $i = 1, ..., |g|$ **faça**
    para $k = 1, ..., K$ **faça**
      Crie partições $T_1^{(k,i)}$, $T_2^{(k,i)}$ a partir de $T^{(i)}$. ;
      Opcionalmente, pode-se definir uma partição $T_3^{(k,i)}$ para otimização dos parâmetros do estimador ;
      Compute $m_{\pi_i}^{(k)}$ usando krigagem e o modelo de covariância C_{i-1}. ;
      Compute $e_{\pi_i}^{(k)}$ usando a eq. (5.29) com $z(\boldsymbol{v})$ sendo um krigador que usa o modelo de covariância C_{i-1}. ;
    fim
    $m(\boldsymbol{u}_{\pi_i}) \leftarrow \dfrac{\sum_{k=1}^{K} m_{\pi_i}^{(k)}}{K}$. ;
    $e(\boldsymbol{u}_{\pi_i}) \leftarrow \dfrac{\sum_{k=1}^{K} e_{\pi_i}^{(k)}}{K}$. ;
    Amostre $z(\boldsymbol{u}_{\pi_i})$ usando a distribuição gaussiana $G(m(\boldsymbol{u}_{\pi_i}), e(\boldsymbol{u}_{\pi_i}))$. ;
    $T^{(i+1)} \leftarrow T^{(i)} \cup \{\boldsymbol{u}_{\pi_i}\}$.;
    se $|T^{(i+1)}| < M_c$ **então**
      $T_c \leftarrow T^{(i+1)}$;
    fim
    senão
      T_c é construído amostrando-se aleatoriamente M_c elementos de $T^{(i+1)}$;
    fim
    se $i \mod P_c = 0$ **então**
      Compute o modelo de covariância C_i usando algum algoritmo de otimização e o conjunto de amostras T_c. ;
    fim
    senão
      $C_i \leftarrow C_{i-1}$;
    fim
  fim
  Retorne $z(\boldsymbol{u}_{\pi_i})$. ;
fim

Algoritmo 7: Algoritmo de simulação *ensemble* com otimização contínua do modelo de covariância.

Entrada: Conjunto de amostras T_c usado para computação do erro de validação. C_0 é o modelo de covariância a ser otimizado. M número de distorções aplicadas. $e(T_c, C)$ é a métrica de erro usando o conjunto de amostras T_c e o modelo de covariância C. Essa métrica pode ser o erro de estimativa, o erro de ajuste às amostras de covariância ou uma combinação de ambas, por exemplo.

saída *Modelo de covariância otimizado C.*
início
  $\epsilon_{min} \leftarrow e(T_c, C_0)$;
  $C_{opt} \leftarrow C_0$;
  para $i = 1, ..., M$ **faça**
    para $k = 1, ..., K$ **faça**
      Compute uma distorção $\Theta_{k,i}$ da anisotropia Θ_k do modelo C somando-se um ruído com distribuição uniforme a cada parâmetro de *range* e ângulo. Ou compute a distorção de ângulo e *range* usando-se um dos método iterativos descrito na Seção A.1 ;
    fim
    Crie C_i usando as anisotropias distorcidas $\Theta_{k,i}$;
    $\epsilon \leftarrow e(T_c, C_i)$;
    se $\epsilon < \epsilon_{min}$ **então**
      $C_{opt} \leftarrow C_i$;
      $\epsilon_{min} \leftarrow \epsilon$;
    fim
  fim
  Retorne C_{opt}. ;
fim

Algoritmo 8: Otimização de um modelo covariância.

5.6 Usando *ensemble simulation* para se construir Modelos Base de Covariância (MBC)

Nesta seção, é apresentado um estudo de caso em que o algoritmo de simulação *ensemble* é utilizado para se computar modelos base de covariância. Modelos base de covariância (MBC) (KLOECKNER et al., 2019) são modelos que representam a variabilidade espacial do fenômeno construídos a partir de diferentes técnicas de estimativa. Em (KLOECKNER et al., 2019), o algoritmo de simulação *ensemble* desenvolvido na Seção 5.4 é aplicado na construção desses modelos e a qualidade dos MBC gerados é comparada com os dados exaustivos, demonstrando a boa capacidade de extração de informação de variabilidade espacial do simulador *ensemble*. A seguir, os resultados desses experimentos são apresentados.

Os MBC computados são usados para se extrair as tabelas de covariância descritas no Capítulo 3. Por sua vez, essa tabelas de covariância podem ser usadas tanto como de modelo de covariância diretamente em algoritmos de simulação geoestatística clássica, quanto como modelo de covariância inicial que é otimizado pelo Algoritmo 7, descrito na Seção 5.5. Além disso, pode-se utilizar o algoritmo de otimização descrito na Seção 5.3 para se extrair o modelo associado à tabela de covariância.

Estrutura	Variância	Azimuth	r_1	r_2	r_3
Efeito pepita	0.20	-	-	-	-
Esférico	0.80	135	80	60	40

Tabela 5 – Modelo de covariância utilizado na geração do conjunto de dados sintético.

Esse processo de geração de MBCs, e a utilização deles na criação de modelos de covariância otimizados, permite a construção de um processo de exploração do espaço de incerteza em que algoritmos mais lentos, como a simulação *ensemble*, define um MBC-semente e este modelo pode ser usado para a geração de um modelo de covariância que é usado por algoritmos de simulação clássica. As realizações do algoritmo de simulação *ensemble* podem ser vistas como polos no espaço de incerteza e, a partir desses polos, novas simulações podem ser geradas de forma muito mais eficiente usando geoestatística clássica. Indiretamente, isso fornece uma metodologia para se explorar o espaço de incerteza de modelos de covariância. Essa combinação entre a abordagem *ensemble* e clássica é vital para a viabilização computacional da realização de um volume maior de simulações com melhor qualidade.

Para ilustrar a capacidade da simulação *ensemble* de extrair as informações de variabilidade espacial de um conjunto de amostras, foi construído um conjunto de dados sintético exaustivo e, desse conjunto, foram extraídas diferentes amostragens. O conjunto sintético foi gerado utilizando-se uma realização não condicional obtida com a Simulação Sequencial Gaussiana. A simulação honra os parâmetros definidos na Tabela 5. As dimensões do modelo são 100 blocos na direção Norte-Sul, 100 blocos na direção Leste-Oeste e 100 blocos na vertical, e cada bloco mede 1x1x1.

Três conjuntos de dados foram amostrados simulando processos de amostragem em malhas de 18x18, 22x22 e 25x25 metros. O sumário estatístico dos conjuntos de dados são representativos do modelo exaustivo. Na Fig. 44, é apresentado o modelo exaustivo em vista em perspectiva (a) e os furos amostrais para 22x22 (b). A Fig. 45 mostra uma seção no plano de vista horizontal do modelo exaustivo (a), e amostras para 18x18 (b), 22x22 (c) e 25x25 (d) em z = 0. Por fim, a Fig. 46 mostra o histograma e o sumário estatístico para o modelo exaustivo (a), e para amostras 18x18 (b), 22x22 (c) e 25x25 (d).

O MBC foi gerado por meio da simulador *ensemble* usando K partições dos conjuntos de calibração. A Tabela 6 mostra a porcentagem de bins espectrais definidos como zero para cada caso. A Fig. 47 mostra a tabela de covariância do modelo exaustivo na vista horizontal em z = 0 (a), comparando com as tabelas de covariância obtidas de MBCs computados a partir das amostras 18x18 (b), 22x22 (c) e 25x25 (d) em z = 0. Fig. 48 mostra a tabela de covariância do modelo exaustivo na seção vertical em x = 0 (a) comparando com MBCs de amostras 18x18 (b), 22x22 (c) e 25x25 (d) em x = 0. As linhas azuis na tabela de covariância do modelo exaustivo e as linhas verde, vermelha e roxa nas

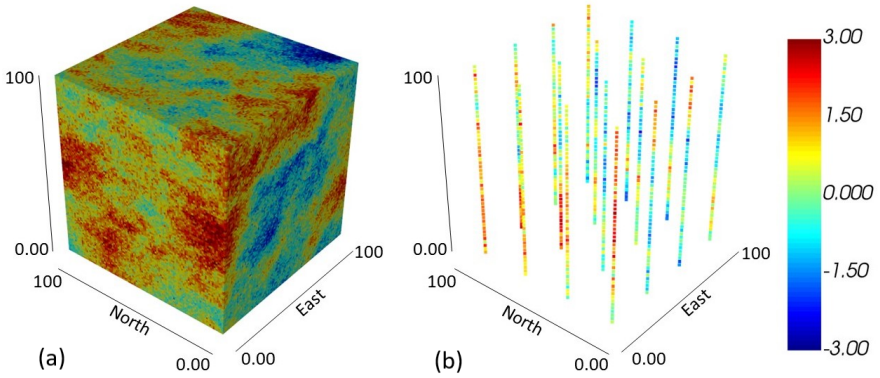


Figura 44 – Modelo exaustivo (a) e amostras para 22x22 (b). (KLOECKNER et al., 2019)

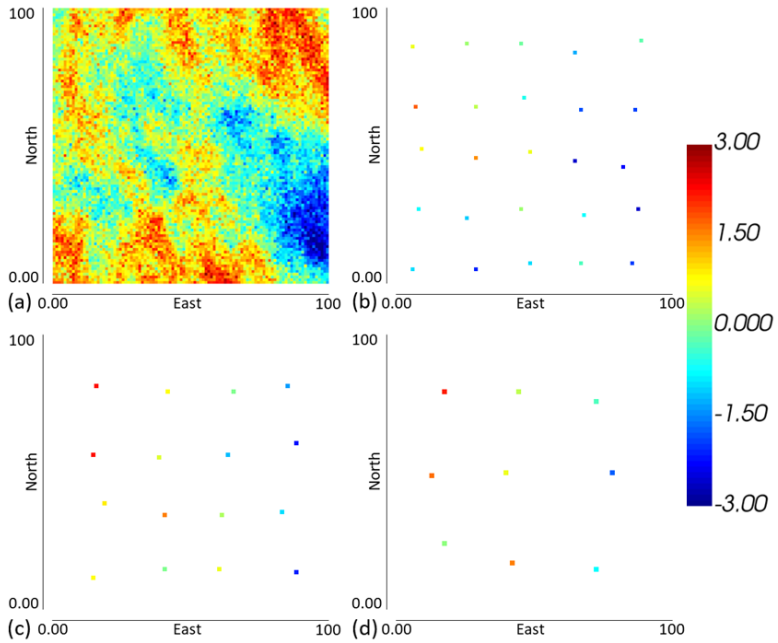


Figura 45 – Uma seção no plano de vista horizontal do modelo exaustivo (a), amostras 18x18 (b), 22x22 (c) e 25x25 (d).(KLOECKNER et al., 2019)

	CT 18x18	CT 22x22	CT 25x25
Bin espectral [4] definido como zero	2.0%	1.9%	1.9%

Tabela 6 – Porcentagem de bins espectrais definidos como zero para cada caso.

tabelas de covariância dos MBCs são visualizadas nos variogramas mostrados na Fig. 49.

Três conjuntos de simulações sequenciais gaussianas (SSG) foram construídos usando tabelas de covariância dos MBCs e comparados com o modelo exaustivo de referência. Para cada caso, 20 realizações foram geradas: SSG usando tabela de covariância do MBC de amostras 18x18, 22x22 e 25x25. Fig. 50 mostra três realizações, uma para cada caso em comparação com o modelo exaustivo em z = 0. Figs. 51, 52 e 53 mostra as funções de distribuição cumulativas (CDF) para os três casos em comparação com o caso exaustivo. Figs. 54, 55 e 56 apresentam as flutuações ergódicas orientadas nos eixos de

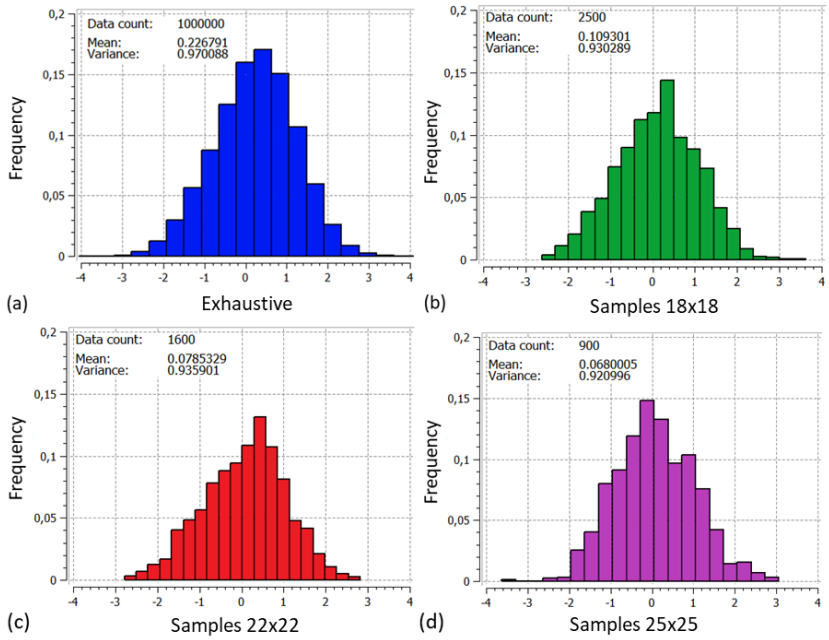


Figura 46 – Histograma do modelo exaustivo (a), das amostras 18x18 (b), 22x22 (c) e 25x25 (d).(KLOECKNER et al., 2019)

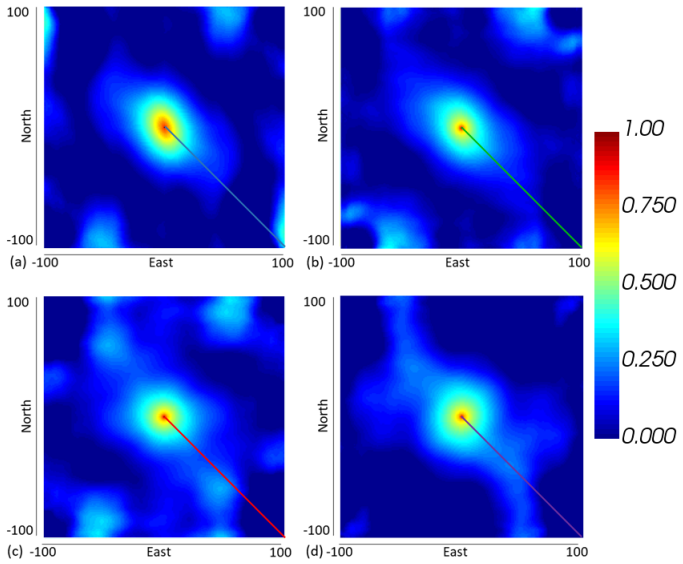


Figura 47 – Tabela de covariância do modelo exaustivo na vista horizontal em z = 0 (a), tabelas de covariância dos MBCs para 18x18 (b), 22x22 (c) e 25x25 (d) em z = 0.(KLOECKNER et al., 2019)

maior e menor continuidade.

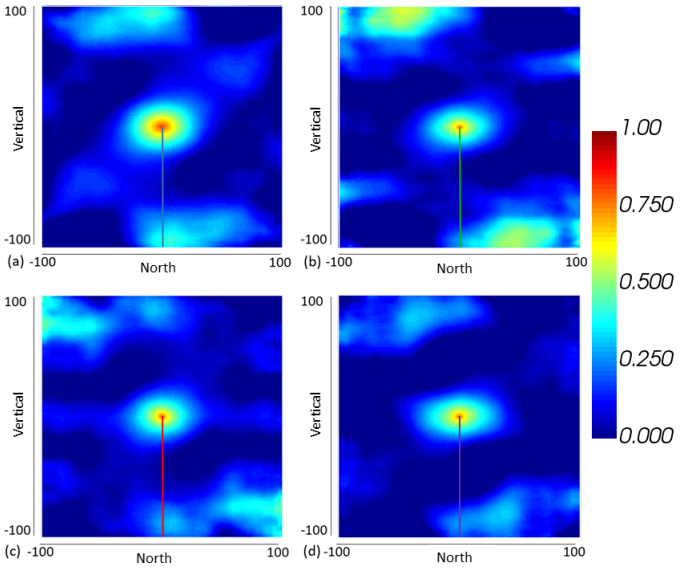


Figura 48 – Tabela de covariância do modelo exaustivo na vista vertical em x = 0 (a), tabelas de covariância dos MBCs para 18x18 (b), 22x22 (c) e 25x25 (d) em x = 0.(KLOECKNER et al., 2019)

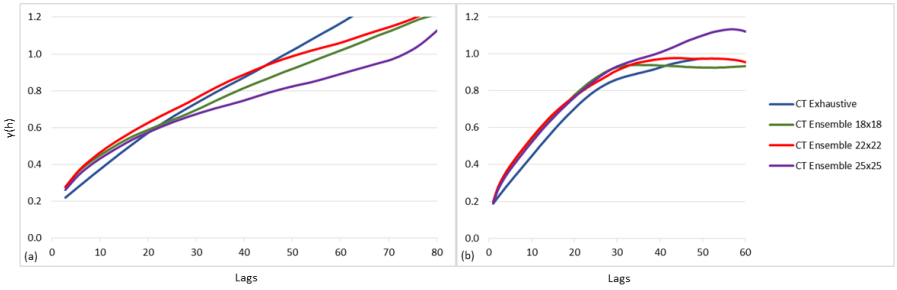


Figura 49 – Os variogramas para a direção principal (a) variograma vertical (b) das tabelas de covariância usando modelo exaustivo e MBCs.(KLOECKNER et al., 2019)

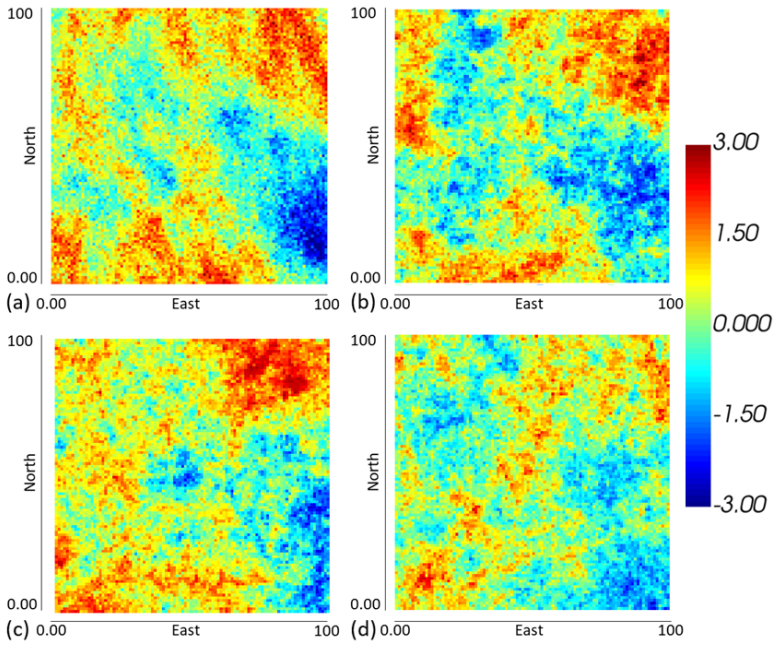


Figura 50 – Uma seção no plano de vista horizontal em z = 0 do modelo exaustivo (a), uma simulação SSG para amostras 18x18 (b), 22x22 (c) e 25x25 (d) usando tabelas de covariância de seus respectivos MBCs.(KLOECKNER et al., 2019)

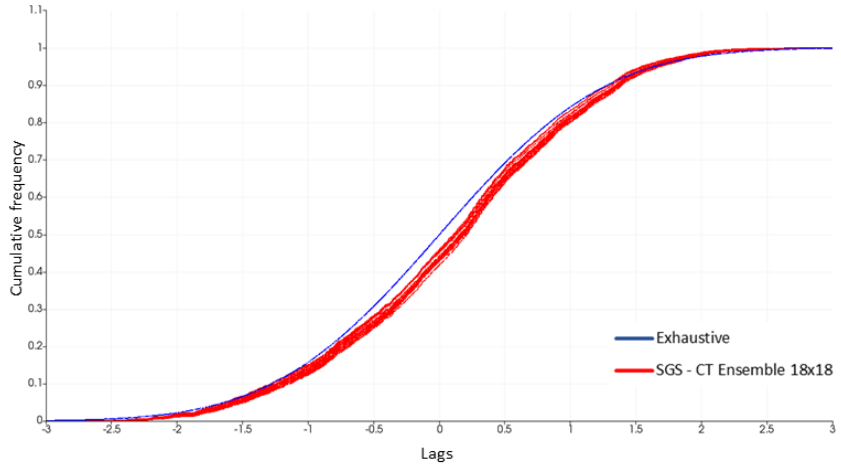


Figura 51 – Função de distribuição cumulativa para SSG usando MBCs de tabela de covariância de amostras 18x18.(KLOECKNER et al., 2019)

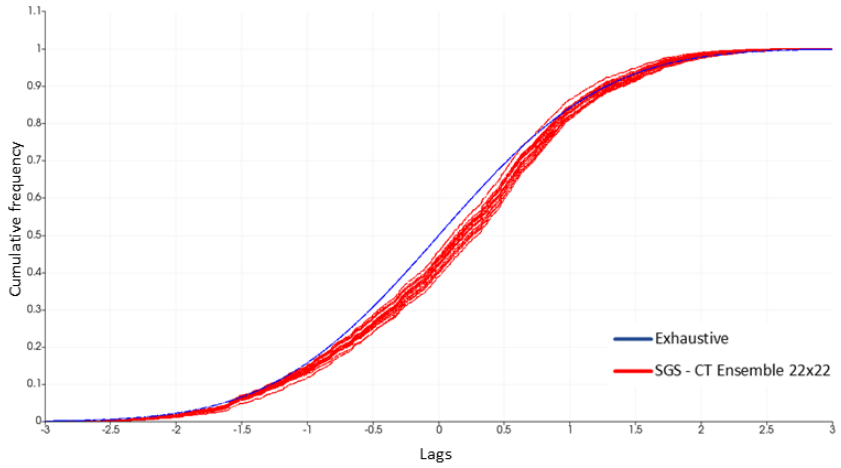


Figura 52 – Função de distribuição cumulativa para SSG usando MBCs de tabela de covariância de amostras 22x22.(KLOECKNER et al., 2019)

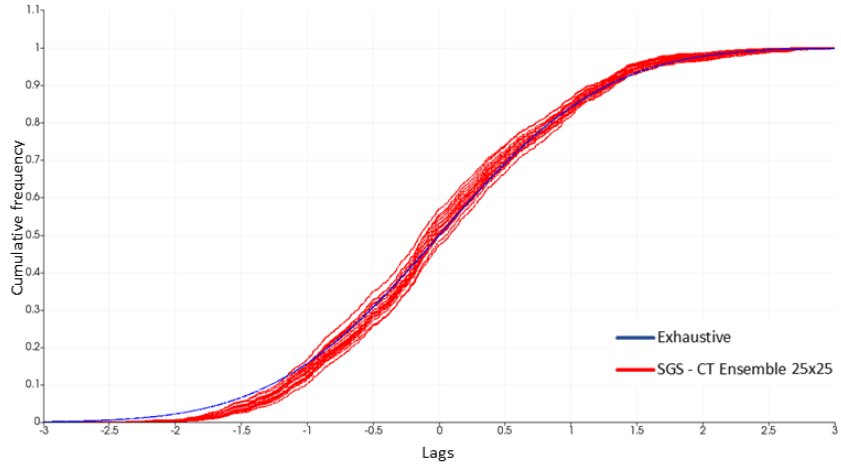


Figura 53 – Função de distribuição cumulativa para SSG usando MBCs de tabela de covariância de amostras 25x25.(KLOECKNER et al., 2019)

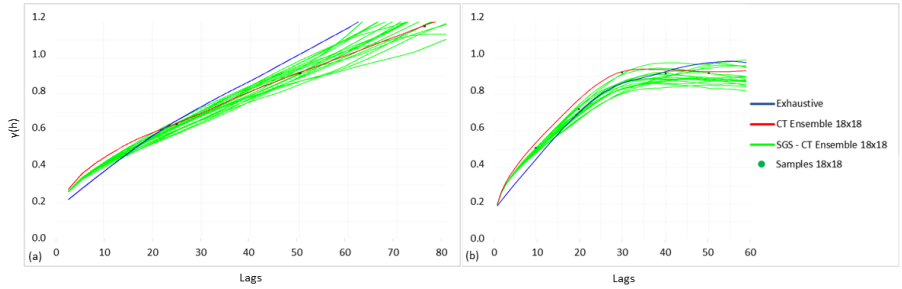


Figura 54 – Flutuações ergódicas nos eixos de maior (a) e menor (b) continuidade para SSG de amostras 18x18.(KLOECKNER et al., 2019)

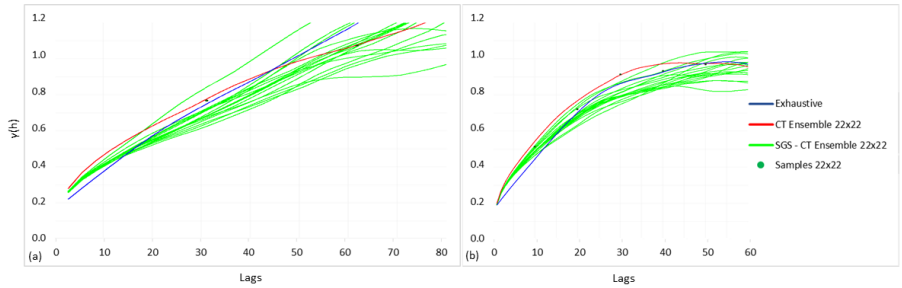


Figura 55 – Flutuações ergódicas nos eixos de maior (a) e menor (b) continuidade para SSG de amostras 22x22.(KLOECKNER et al., 2019)

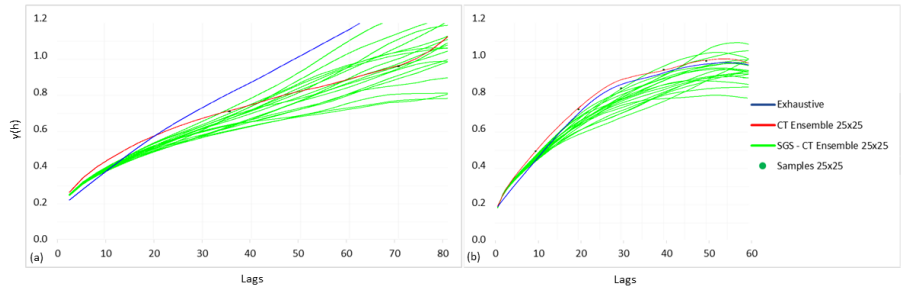


Figura 56 – Flutuações ergódicas nos eixos de maior (a) e menor (b) continuidade para SSG de amostras 25x25.(KLOECKNER et al., 2019)

Os resultados mostram que a metodologia apresentada produz bons modelos de covariância para os três conjuntos de dados amostrais deste caso estudado. Além disso, a perda de qualidade no modelo de covariância devido a redução do número de amostras é bem reduzida, mostrando a robustez do método. MBCs é uma metodologia nova e é aberta para diferentes modos para se gerar os modelos-base.

5.7 Resultados experimentais: comparando *ensemble simulation* com *TBSIM*

O desempenho do algoritmo *ensemble simulation* é avaliado nessa seção. Para isso, um banco de dados sintético foi construído usando-se o algoritmo *TBSIM* com os

parâmetros a seguir:

- 1000 linhas.
- Variograma: $\gamma(h) = 0.07 + sph(sill = 0.93, ranges = (r1 = 360, r2 = 227, r3 = 62), angles = (azimuth = 207.3, dip = -10.1, rake = 0.))(h)$.
- Distribuição: $\Gamma(\kappa = 1.4, \theta = 2.7)$, média $\mu = 3.5062$ e variância $\delta^2 = 9.85$. Uma distribuição gama Γ com parâmetro de forma $\kappa = 1.4$ e escala $\theta = 2.7$.

Os dados foram gerados em uma malha cartesiana com dimensões: $sx, sy, sz = (2, 2, 2)$, $Lx, Ly, Lz = (300, 300, 130)$ e número total de células igual a 1.462.500. A Fig. 57 apresenta o sumário estatístico dos dados exaustivos.

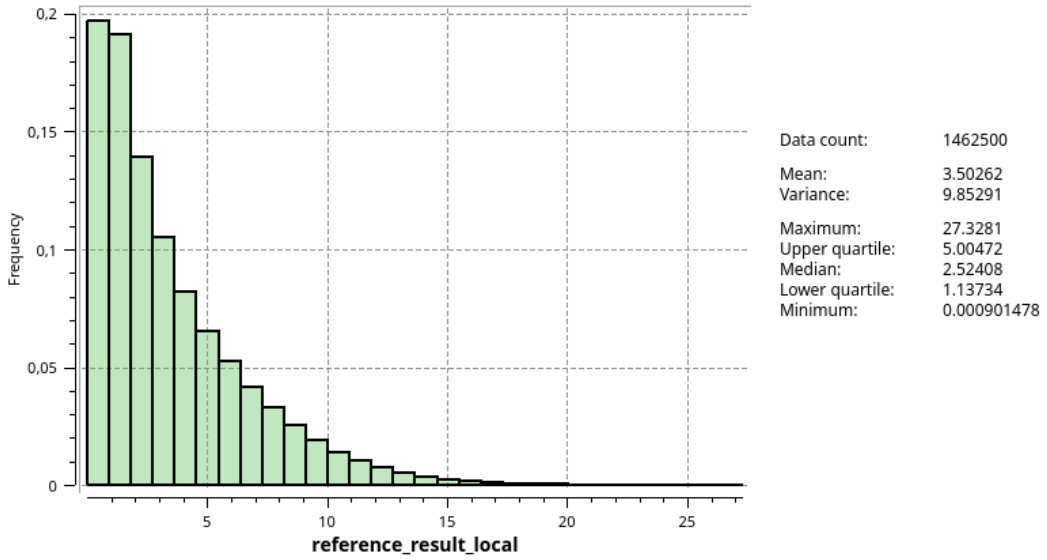


Figura 57 – Distribuição dos dados sintéticos exaustivos usado no teste comparativo.

Dos dados exaustivos, foram extraídas 100 amostras que foram utilizadas nos experimentos. As amostras são ilustradas na Fig. 58 e, na Fig. 59, é mostrado o dado exaustivo de referência.

Foram realizadas 20 simulações *ensemble* seguindo os passos a seguir:

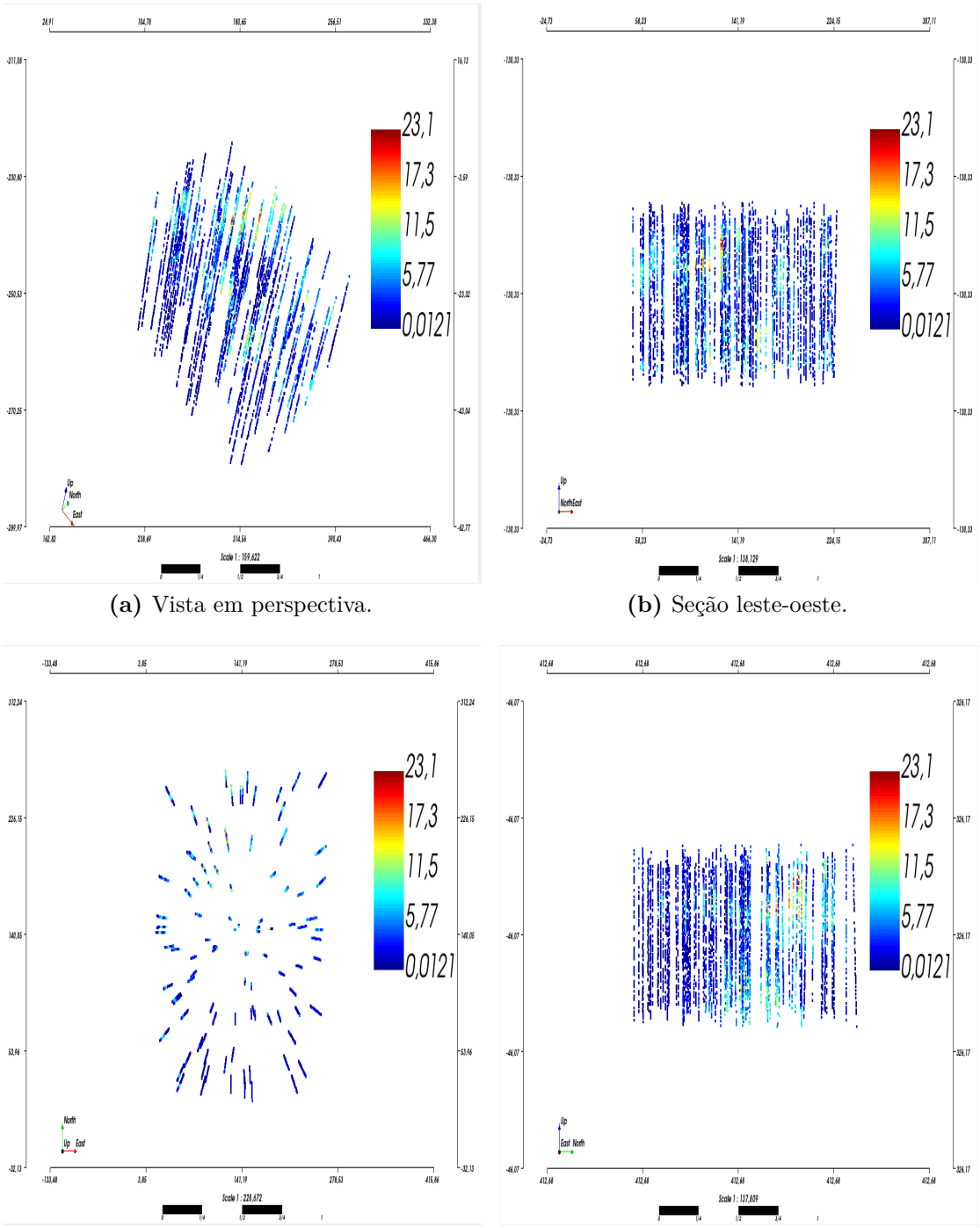
- Um estimador-base foi otimizado utilizando-se busca tabu e LBFGS. Como estimador inicial foi usado krigagem com modelo isotrópico com estrutura única com raio 150 e modelo esférico. Esse estimador inicial foi otimizado utilizando o procedimento descrito no Algoritmo 3. LBFGS foi usado para se construir os vizinhos necessários na busca tabu. Esse procedimento, permite encontrar um modelo de covariância inicial que minimiza globalmente o erro de validação cruzada.

- O estimador otimizado é usado no processo de simulação *ensemble* para se computar os erros e o valor esperado, conforme descrito no Algoritmo 6.

- Foram usadas oito partições. Em cada partição, 85% dos dados foram usados para condicionamento e 15% foram utilizados na estimativa do erro.

- A vizinhança utilizada na validação cruzada contém 32 dados amostrais e 16 previamente simulados.

As simulações *ensemble* são comparadas com simulações *TBSIM* geradas com os mesmos parâmetros (distribuição, número de linhas e modelo de covariância) utilizados para se construir os dados de referência e, além disso, foram utilizados 32 dados condicionantes por ponto simulado. Isto é, o *TBSIM* foi executado com parâmetros praticamente ótimos. Em todos testes, a vizinhança de busca é isotrópica com raio 200.



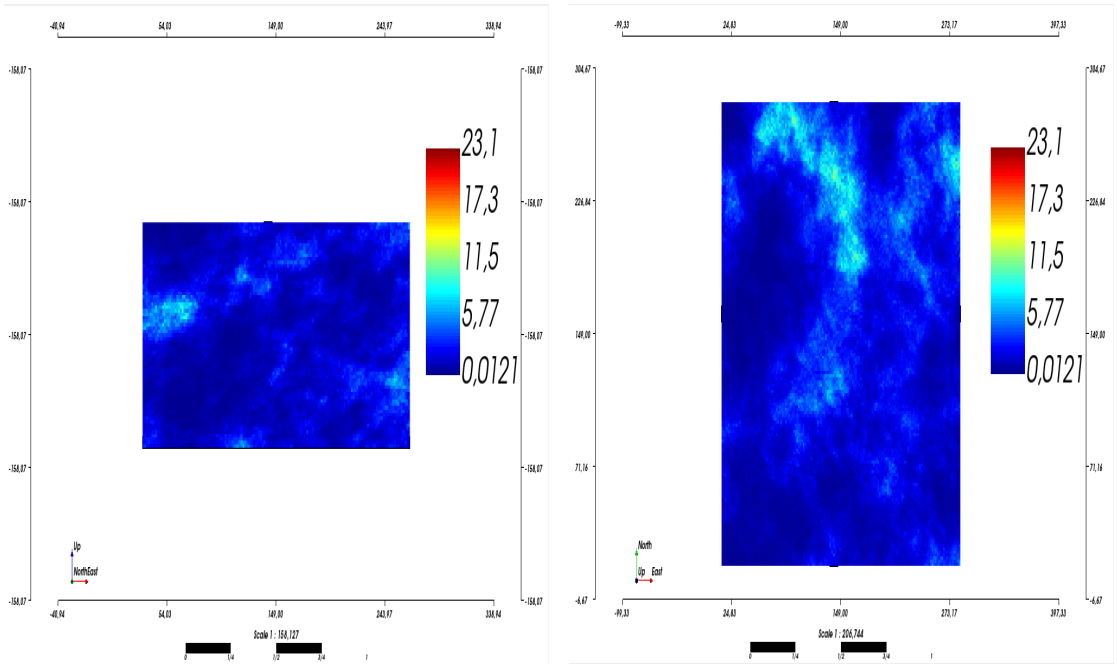
(a) Vista em perspectiva.

(b) Seção leste-oeste.

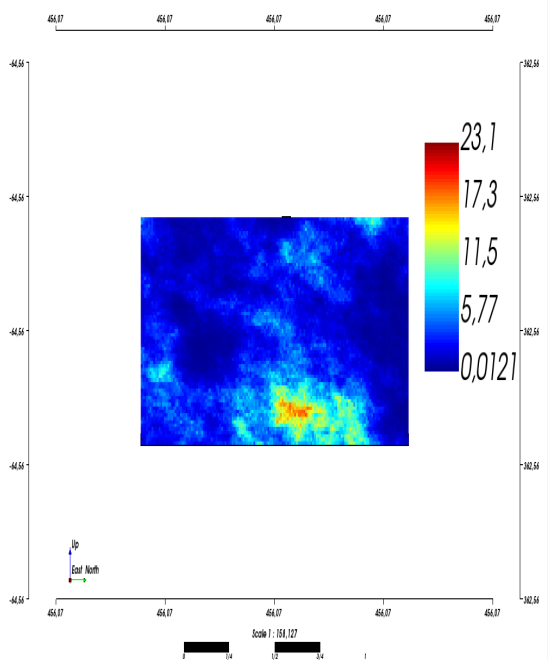
(c) Vista superior.

(d) Seção norte-sul.

Figura 58 – 100 amostras usadas nos experimentos.



(a) Seção leste-oeste. **(b)** Vista superior.

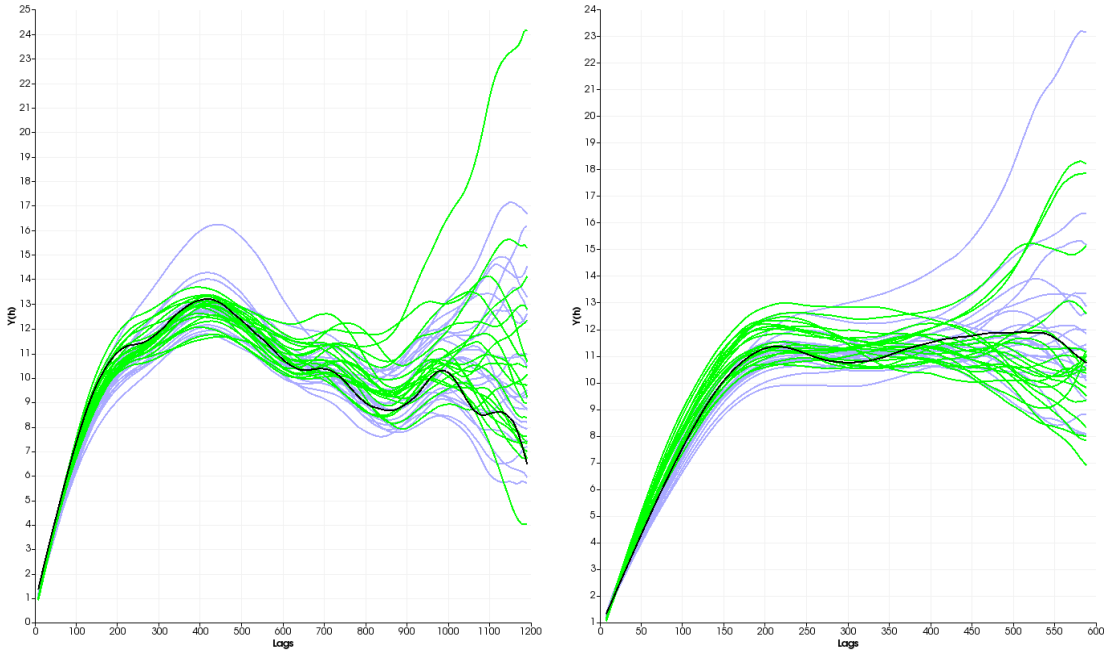


(c) Seção norte-sul.

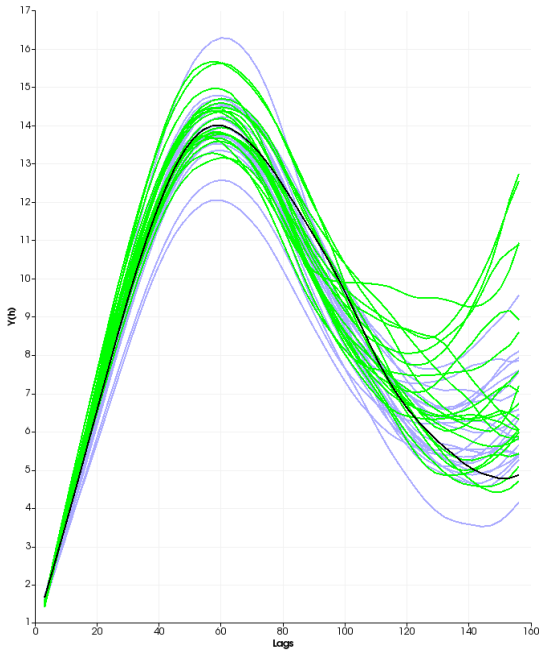
Figura 59 – Dados exaustivos de referência.

Conforme visto na Fig. 60, as simulações *ensemble* produzem simulações com variogramas com flutuações ergódicas similares às obtidas pela simulação *TBSIM*. Nas Fig. 61 e 62, são vistas a distribuição do erro absoluto com relação aos dados exaustivos do *E-Type* das simulações *ensemble* e *TBSIM*, respectivamente. Pode-se notar, que a média, a mediana e a variância do erro absoluto da simulação *ensemble* são muito similares

às obtidas pela simulação *TBSIM*. Além disso, nas Figs. 63 e 64 são vistos os gráficos de dispersão comparando o *E-Type* das simulações *ensemble* e *TBSIM* contra os dados exaustivos e, de forma similar, nota-se que a correlação de ambas técnicas é similar. Por fim, na Fig. 65 são mostrados os histogramas das simulações realizadas comparadas com os dados exaustivos.



(a) Direção maior continuidade. **(b)** Direção continuidade média.



(c) Direção menor continuidade.

Figura 60 – Comparação dos variogramas das simulações *ensemble* (em roxo) e *TBSIM* (em verde) nas direções de continuidade maior, média e menor, respectivamente. Em preto, têm-se o variograma experimental dos dados exaustivos.

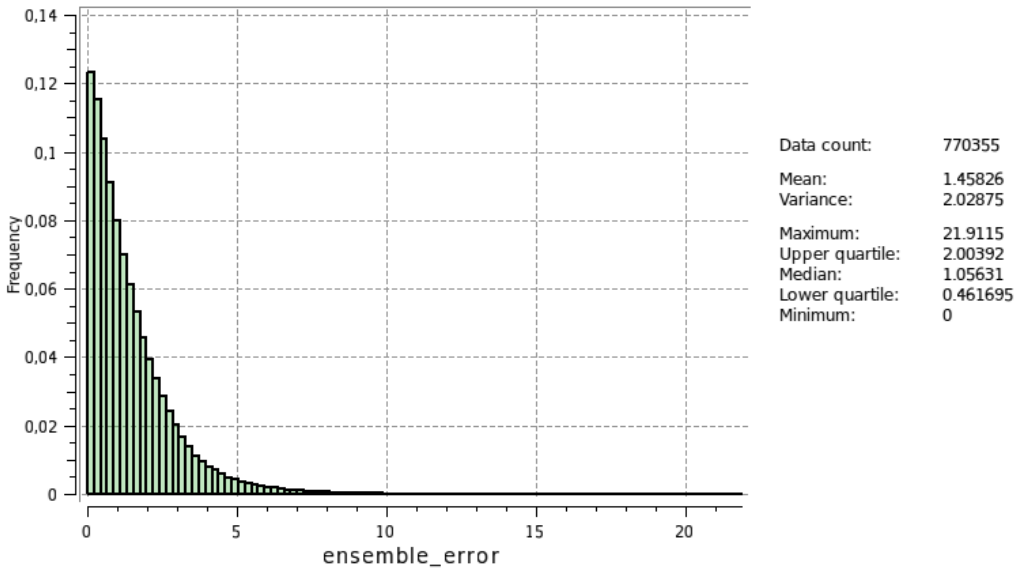


Figura 61 – Distribuição do erro absoluto da simulação *ensemble* computado relativo aos dados exaustivos.

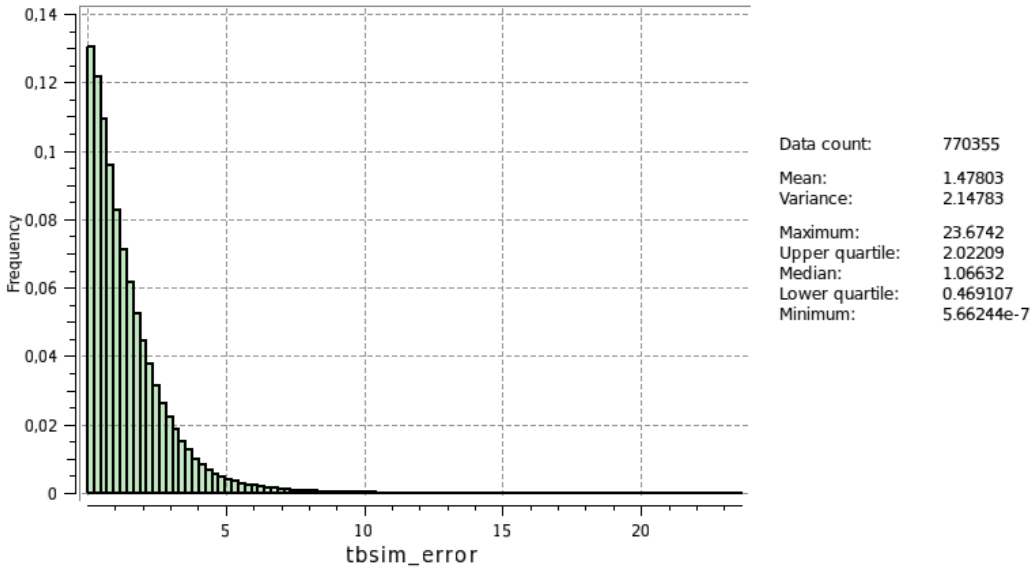


Figura 62 – Distribuição do erro absoluto da simulação *TBSIM* computado relativo aos dados exaustivos.

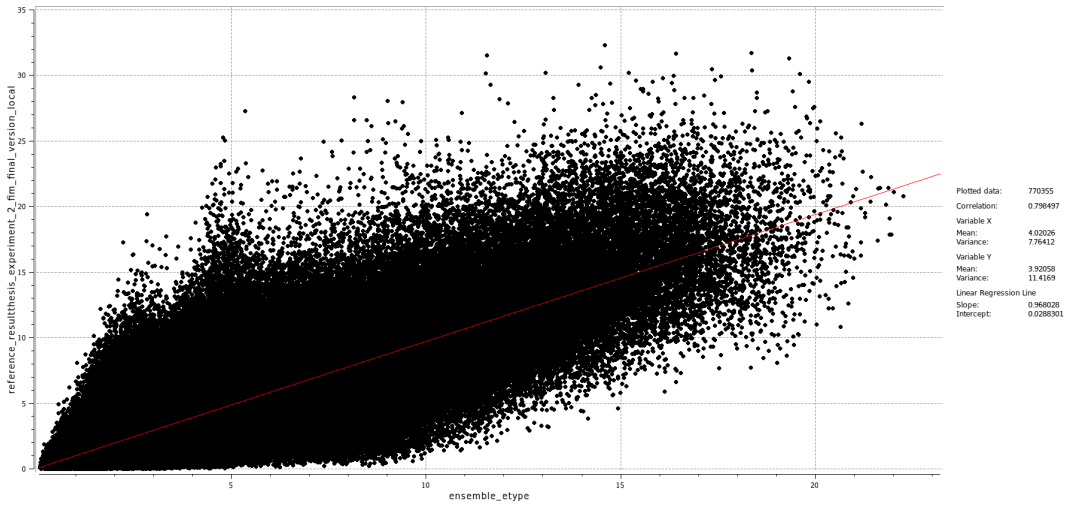


Figura 63 – Gráfico de dispersão comparando o *E-Type* de simulações *ensemble* com os dados exaustivos de referência.

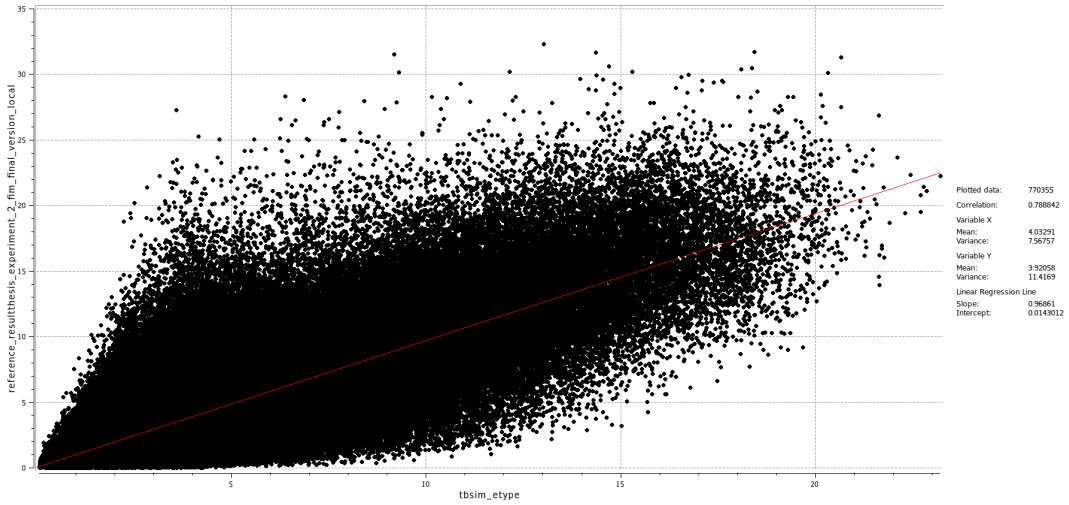


Figura 64 – Gráfico de dispersão comparando o *E-Type* de simulações *TBSIM* com os dados exaustivos de referência.

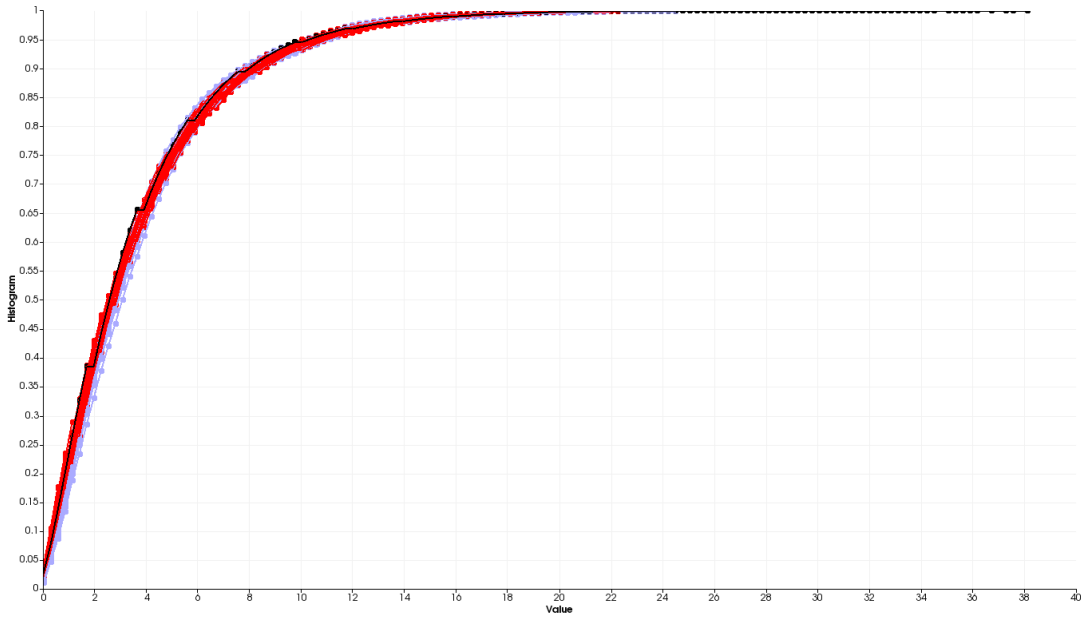


Figura 65 – Histogramas das simulações *TBSIM* (em vermelho), das simulações *ensemble* (em roxo) e dos dados exaustivos (em preto).

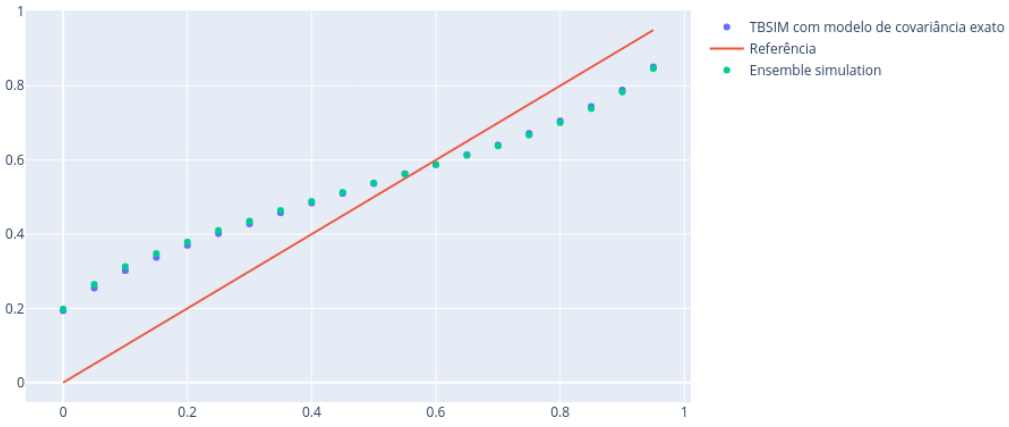


Figura 66 – *Accuracy plot* das simulações TBSIM e das simulações *ensemble*.

A Tabela 7 apresenta o resumo dos resultados e a fig. 66 apresenta a comparação dos *accuracy plots* de cada algoritmo. Analisando-se a tabela e os *accuracy plot*, pode-se concluir que, no cenário estudado, o simulador *ensemble* produziu resultados com qualidade similar à obtida com o simulador *TBSIM* usando parâmetros ideais. Embora, a reprodução das distribuições locais das simulações *ensemble* seja ligeiramente pior. Mas, esse resultado é esperado, já que o algoritmo utiliza dados amostrados para obter implicitamente o modelo de covariância dos dados.

Algoritmo	Erro médio	Erro med.	Var. do Erro	Corr.	Goodness
Ens. E-Type	1,458	1,056	2,028	0,798	0,866
TBSIM E-Type	1,478	1,066	2,147	0,788	0,872

Tabela 7 – Sumário comparativo dos experimentos realizados.

Os experimentos realizados nessa seção mostram a capacidade dos algoritmos de otimização combinados com a técnica de simulação *ensemble* de reproduzirem características espaciais contidas nos dados. A partir de um estimador inicial, o processo de otimização combinado com a validação cruzada obtém um estimador que minimiza o erro quadrático. Esse estimador no processo de simulação *ensemble* é utilizado para se estimar as variâncias locais. Pelos resultados obtidos, pode-se concluir que a simulação *ensemble* combinada com estimadores otimizados pode obter resultados similares aos obtidos por meio de algoritmos clássicos de simulação, como o *TBSIM*. A principal vantagem da simulação *ensemble* é sua capacidade de produzir bons resultados a partir de modelos iniciais grosseiros. Isso permite tanto a melhoria de resultados obtidos a partir de estimadores por krigagem clássicos, quanto a construção de modelos base de covariância.

Por fim, é importante lembrar que cada simulação *ensemble* obtida pode ser usada como base para se construir modelos de covariância que poderão ser utilizadas por simulações SGSIM ou FIM. Isso produz uma interessante medologia para se explorar com eficiência o espaço de incerteza, sem necessariamente se definir um modelo explícito de covariância.

5.8 Conclusão

O objetivo desse capítulo é apresentar uma metodologia para se transformar algoritmos clássicos de geoestatística em algoritmos com capacidade de aprendizado e uma metodologia para otimizar parâmetros de estimadores baseado em computação de erros de validação. O aprendizado de máquina é basicamente a aplicação de um algoritmo de otimização para se calibrar os parâmetros do estimador baseado nos erros de validação fornecidos pelos dados. A validação cruzada é o principal elemento para se construir um algoritmo moderno de estimativa, e, usando-se essa metodologia, pode-se aprimorar a qualidade de estimativa de qualquer estimador.

Com a validação cruzada, é mostrado que mesmo estimadores baseados em ponderadores simples podem ser usados para se construir simuladores capazes de reproduzir os modelos de covariância e a distribuição inerente aos dados amostrais. Obviamente, essa técnica não é universal e depende bastante da qualidade e quantidade das amostras, mas ela fornece uma boa alternativa prática para se obter melhores simulações.

Em situações com menos amostras e onde já existe um candidato inicial para o

modelo de covariância, a técnica de simulação *ensemble* pode ser usada para se aprimorar o modelo de covariância a cada passo. Os candidatos iniciais podem também ser aprendidos a partir dos dados usando-se busca tabu ou outro algoritmo de otimização. Além disso, também é mostrado que o processo de otimização de modelo de covariância é recursivo e pode ser usado para se obter no final do processo uma melhor representação da covariância conhecido como Modelo Base de Covariância (MBC).

Aprendizado de máquina permite uma evolução significativa da qualidade de simulações e estimativas. Graças à evolução do poder computacional, se torna cada vez mais viável a realização de mais etapas de calibração durante o processo de estimativa ou simulação. Com mais processamento, a qualidade dos modelos computados evolui naturalmente.

O foco desse capitulo foi o caso univariado, mas essas técnicas podem ser estendidas para o caso multivariado.

Conclusão e trabalhos futuros

Ao longo desse trabalho, foram apresentados algoritmos que se beneficiam da evolução tanto das técnicas de programação, quanto da capacidade de processamento dos computadores atuais. Algoritmos espectrais, como *Turning Bands* e *Fourier Integral Method - FIM*, se mostraram interessantes alternativas que se beneficiam de técnicas computação distribuída. Por não possuírem caminho aleatório em sua estrutura, esses algoritmos não precisam de estratégias de sincronização como as utilizadas na paralelização de algoritmos de simulação sequencial. Isto é, esses algoritmos permitem abordagens *shared nothing*, que são mais simples de implementar, mais robustas e menos propensas a erros de implementação.

O FIM, além de ser um algoritmo eficiente, permite a utilização de uma representação implícita dos modelos de covariância, a *Tabela de Covariância*: uma estrutura que tanto pode ser utilizada tanto como modelo computacional de covariância em algoritmos de estimativa e simulação (após algum pós-processamento), quanto como pode ser usado para obtenção de modelos analíticos de variograma via métodos de otimização (conforme visto no Capítulo 5). A obtenção semi-automática de modelos de covariância exerce um papel fundamental na redução de custo humano para se realizar aplicar técnicas de geoestatística, pois esse é o estágio mais custoso do processo.

O objetivo da tese foi alcançado e mostrou-se que otimização matemática, aprendizado de máquina e computação de alta performance podem ajudar a construir uma nova geração de poderosos algoritmos que ou são mais eficientes ou demandam menos interferência humana. Demonstrou-se que simulação espectral é uma técnica ideal para a utilização de computação distribuída. Além disso, a validação cruzada fornece um modo de incorporar informação local via otimização de um estimador mais fraco (isto é, um estimador que minimiza só o erro esperado da média, por exemplo).

Em linhas gerais, neste trabalho, procurou-se entender como técnicas de computação moderna podem contribuir na aceleração das etapas de modelagem geoestatística. Para isso, tanto computação distribuída aplicada a algoritmos mais adequados a esse tipo técnica, quanto técnicas de otimização numérica foram estudadas. O aprendizado de máquina é um ramo recente da computação que combina técnicas estatísticas com técnicas de otimização e análise matemática para se obter uma nova geração de metodologias para modelagem de dados. Desse ramo de estudo, foram exploradas técnicas de validação cruzada, otimização estocástica (busca tabu) e métodos iterativos para se construir um *workflow* de modelagem evolutiva. Isto é, foi desenvolvido um processo chamado de *ensemble simulation* que permite ao longo de diferentes passos evoluir a qualidade de um modelo, via minimização do erro

de validação cruzada. O meta-algoritmo proposto permite que técnicas de otimização do modelo de covariância sejam utilizadas periodicamente de modo a reprodução de histograma e variograma sejam mais precisas, além de garantir uma redução de erros de validação.

Foi mostrado que utilizando-se computação moderna é possível obter uma nova geração de algoritmos que, embora mais custosos, reduzem a necessidade de intervenção humana e produzem modelos que incorporam mais precisamente informações contidas nos dados amostrais. Os experimentos realizados, conforme visto no Capítulo 5, demonstraram que os modelos obtidos implicitamente pela metodologia *ensemble simulation* foram capazes de capturar o modelo de covariância contido nos dados e foram capazes de reproduzir as estatísticas de primeira e segunda ordem (histograma e variograma).

As técnicas desse trabalho foram aplicadas no caso univariado, mas nada impede que sejam generalizadas para o caso multivariado.

Como trabalhos futuros, sugere-se os seguintes tópicos:

i Adaptação da metodologia para incorporação de informação multivariada (minimização de erro de validação cruzada para múltiplas variáveis);

ii Adicionar restrições ao processo de otimização realizado pelo *ensemble simulation*. Isto é, aplicar aos processo iterativo técnicas de programação não-linear com restrições para que relações não-lineares sejam satisfeitas (por exemplo, as proporções de cada variável sigam uma desigualdade ou igualdade arbitrária $g(v_1, v_2, \ldots, v_N) \leq 0$, onde v_i são variáveis sendo estimadas).

iii Generalizar a técnica para o caso não estacionário. Isto é construir uma técnica de *ensemble simulation* que possa ser combinada com metodologias de LVA.

iv Construção de funções de erro empírico que combine parâmetros econômicos e operacionais.

v Investigar interpoladores baseados em séries de funções ortogonais como estimadores "fracos" utilizados pelo *ensemble simulation*.

A principal conclusão desse trabalho é que computação moderna fornece promissoras ferramentas para agilizar e aprimorar metodologias já estabelecidas, mas isso jamais substitui a capacidade analítica de um bom modelador. As técnicas estudadas não devem ser usadas cegamente, sempre sendo necessária alguma avaliação do modelador para garantir bons resultados. O que se nota na prática é que metodologias lineares tradicionais fornecem excelentes soluções iniciais, elemento fundamental para garantir a convergência dos métodos iterativos discutidos ao longo do Capítulo 5. Por isso, pode-se concluir que

a computação moderna fornece algoritmos que permitem aprimoramento de soluções lineares clássicas, mas dificilmente esses algoritmos substituirão completamente técnicas já estabelecidas. Haja vista que sempre será necessário um modelo inicial para se iniciar o processo.

Como os problemas estudados em geoestatística apresentam naturalmente milhares ou mesmo milhões de parâmetros locais, dificilmente poder-se-á substituir complementamente os algoritmos já estabelecidos. Com isso em mente, pode-se ver os algoritmos desse trabalho como um processo evolutivo que auxiliam o modelador a incorporar iterativamente novas características ao seu modelo de modo que cada vez mais restrições sejam satisfeitas. Além disso, a computação distribuída permite que esses algoritmos possam ser implementados de forma eficiente.

Referências

ALBRECHT, P. *Análise numérica: um curso moderno*. [S.l.]: LTC, 1973. Citado 4 vezes nas páginas 167, 168, 172 e 175.

ALBRECHT, T. Pitfalls of object oriented programming. *Proceedings of Game Connect: Asia Pacific (GCAP)*, 2009. Citado na página 31.

ASMUSSEN, S.; GLYNN, P. W. *Stochastic simulation: algorithms and analysis*. [S.l.]: Springer, 2011. Citado 6 vezes nas páginas 38, 113, 114, 116, 123 e 124.

BEN-ISRAEL, A. A newton-raphson method for the solution of systems of equations. *Journal of Mathematical analysis and applications*, Academic Press, v. 15, n. 2, p. 243–252, 1966. Citado na página 119.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, v. 13, n. Feb, p. 281–305, 2012. Citado na página 122.

BERTSEKAS, D. P. *Nonlinear programming*. [S.l.]: Athena scientific Belmont, 1999. Citado na página 63.

BOASHASH, B. *Time-Frequency Signal Analysis and Processing: A Comprehensive Reference*. [S.l.]: Oxford: Elsevier Science, 2003. Citado 2 vezes nas páginas 49 e 51.

BOCHNER, S. Additive set functions on groups. *Annals of Mathematics*, JSTOR, p. 769–799, 1939. Citado na página 64.

BOYD, S.; BOYD, S. P.; VANDENBERGHE, L. *Convex optimization*. [S.l.]: Cambridge university press, 2004. Citado na página 171.

BROYDEN, C. G. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, Oxford University Press, v. 6, n. 1, p. 76–90, 1970. Citado na página 38.

CHEN, Y.; CUI, X.; MEI, H. Large-scale fft on gpu clusters. In: ACM. *Proceedings of the 24th ACM International Conference on Supercomputing*. [S.l.], 2010. p. 315–324. Citado na página 64.

COOLEY, J. W.; TUKEY, J. W. An algorithm for the machine calculation of complex fourier series. *Math. Comp.*, v. 19, p. 297–301, 1965. Citado 6 vezes nas páginas 40, 49, 51, 54, 55 e 56.

DAGUM, L.; MENON, R. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, IEEE, v. 5, n. 1, p. 46–55, 1998. Citado na página 37.

DEUTSCH, C. V.; JOURNEL, A. Geostatistical software library and user's guide. *Oxford University Press, New York*, 1998. Citado na página 63.

DIETRICH, C. R.; NEWSAM, G. N. Efficient generation of conditional simulations by chebyshev matrix polynomial approximations to the symmetric square root of the covariance matrix. *Mathematical geology*, Springer, v. 27, n. 2, p. 207–228, 1995. Citado na página 84.

DMITRUK, P.; WANG, L.-P.; MATTHAEUS, W.; ZHANG, R.; SECKEL, D. Scalable parallel fft for spectral simulations on a beowulf cluster. *Parallel Computing*, Elsevier, v. 27, n. 14, p. 1921–1936, 2001. Citado na página 64.

DREYBAND, J.; NILVA, L.; SHAPIRO, M. *Descriptive data construct mapping method and apparatus*. [S.l.]: Google Patents, 2001. US Patent App. 09/844,993. Citado na página 33.

EMERY, X.; LANTUÉJOUL, C. Tbsim: A computer program for conditional simulation of three-dimensional gaussian random fields via the turning bands method. *Computers & Geosciences*, Elsevier, v. 32, n. 10, p. 1615–1628, 2006. Citado 5 vezes nas páginas 34, 37, 70, 76 e 80.

FIGUEIREDO, D. G. de. *Equações diferenciais aplicadas*. [S.l.]: Impa, 1979. Citado na página 168.

FIGUEIREDO, D. G. de. *Análise de Fourier e equações diferenciais parciais*. [S.l.]: Instituto de Matemática Pura e Aplicada, 2000. Citado 2 vezes nas páginas 43 e 44.

FOURIER, J.-B. J. Théorie analytique de la chaleur. *Paris : F. Didot*, 1822. Citado na página 44.

FREULON, X. Conditional simulation of a gaussian random vector with non linear and/or noisy observations. In: *Geostatistical Simulations*. [S.l.]: Springer, 1994. p. 57–71. Citado 2 vezes nas páginas 74 e 121.

FRÖBERG, C.-E. *Introduction to numerical analysis*. [S.l.]: Addison-Wesley Publishing Company Reading Massachusetts, 1969. Citado na página 168.

GAUSS, C. F. Nachlass: Theoria interpolationis methodo nova tractata. *Carl Friedrich Gauss Werke*, v. 3, p. 265–327, 1866. Citado na página 54.

GEORGE, C.; JEAN, D.; TIM, K. Distributed systems: Concepts and design. *Addison-Wesley*, v. 2, p. 410, 2005. Citado na página 31.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, Elsevier, v. 13, n. 5, p. 533–549, 1986. Citado 2 vezes nas páginas 38 e 119.

GOOVAERTS, P. *Geostatistics for natural resources evaluation*. [S.l.]: Oxford University Press on Demand, 1997. Citado 3 vezes nas páginas 34, 116 e 125.

GOOVAERTS, P. Geostatistical modelling of uncertainty in soil science. *Geoderma*, Elsevier, v. 103, n. 1-2, p. 3–26, 2001. Citado na página 95.

GU, M.; EISENSTAT, S. C. A divide-and-conquer algorithm for the bidiagonal svd. *SIAM Journal on Matrix Analysis and Applications*, SIAM, v. 16, n. 1, p. 79–92, 1995. Citado na página 175.

GUENNEBAUD, G.; JACOB, B. et al. Eigen: a c++ linear algebra library. *URL http://eigen. tuxfamily. org, Accessed*, v. 22, 2014. Citado na página 175.

HEIDEMAN, M. T.; JOHNSON, D. H.; BURRUS, C. S. Gauss and the history of the fast fourier transform. *Archive for history of exact sciences*, Springer, v. 34, n. 3, p. 265–277, 1985. Citado na página 54.

HIGHAM, N. J. Computing the nearest correlation matrix—a problem from finance. *IMA journal of Numerical Analysis*, Oxford University Press, v. 22, n. 3, p. 329–343, 2002. Citado na página 63.

HOFFBECK, J. P.; LANDGREBE, D. A. Covariance matrix estimation and classification with limited training data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 18, n. 7, p. 763–767, 1996. Citado na página 64.

HUNGER, L.; COSENZA, B.; KIMESWENGER, S.; FAHRINGER, T. Spectral turning bands for efficient gaussian random fields generation on gpus and accelerators. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 27, n. 16, p. 4122–4136, 2015. Citado na página 34.

ISAAKS, E. H. Indicator simulation: application to the simulation of a high grade uranium mineralization. In: *Geostatistics for natural resources characterization. Part 1*. [S.l.: s.n.], 1984. Citado na página 34.

ISAAKS, E. H. *The application of Monte Carlo methods to the analysis of spatially correlated data*. Tese (Doutorado), 1991. Citado na página 34.

JEONG, H.; LEE, W.; PAK, J.; CHOI, K.-j.; PARK, S.-H.; YOO, J.-s.; KIM, J. H.; LEE, J.; LEE, Y. W. Performance of kepler gtx titan gpus and xeon phi system. *arXiv preprint arXiv:1311.0590*, 2013. Citado na página 32.

JOHNSON, S. G.; FRIGO, M. Implementing ffts in practice. *Fast Fourier Transforms*, Rice University, Houston TX: Connexions, 2008. Citado 5 vezes nas páginas 54, 55, 56, 58 e 59.

JOURNEL, A.; ALABERT, F. Non-gaussian data expansion in the earth sciences. *Terra Nova*, Wiley Online Library, v. 1, n. 2, p. 123–134, 1989. Citado na página 34.

JOURNEL, A. G.; HUIJBREGTS, C. J. *Mining geostatistics*. [S.l.]: Academic press, 1978. Citado na página 68.

KENNEDY, J. Particle swarm optimization. *Encyclopedia of machine learning*, Springer, p. 760–766, 2010. Citado na página 119.

KLOECKNER, J.; MACHADO, P. L.; RODRIGUES, Á. L.; COSTA, J. F. C. L. Covariance table: A fast automatic spatial continuity mapping. *Computers & Geosciences*, Elsevier, 2019. Citado 10 vezes nas páginas 19, 20, 38, 115, 134, 136, 137, 138, 139 e 140.

KOHAVI, R. et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: MONTREAL, CANADA. *Ijcai*. [S.l.], 1995. v. 14, n. 2, p. 1137–1145. Citado 2 vezes nas páginas 114 e 121.

LAARHOVEN, P. J. V.; AARTS, E. H. Simulated annealing. In: *Simulated annealing: Theory and applications*. [S.l.]: Springer, 1987. p. 7–15. Citado na página 119.

LANTUÉJOUL, C. *Geostatistical simulation: models and algorithms*. [S.l.]: Springer Science & Business Media, 2013. Citado 3 vezes nas páginas 69, 70 e 71.

LARRONDO, P. F.; NEUFELD, C. T.; DEUTSCH, C. V. Varfit: A program for semi-automatic variogram modeling. *Fifth Annual Report of the Centre for Computational Geostatistics, University of Alberta, Edmonton*, 2003. Citado na página 63.

LEMONS, D. S.; LANGEVIN, P. *An introduction to stochastic processes in physics*. [S.l.]: JHU Press, 2002. Citado na página 69.

LIMA, E. L. Espaços métricos, projeto euclides. *São Paulo, SP, Brasil*, v. 954, 1977. Citado 2 vezes nas páginas 165 e 166.

LOOMIS, L. An introduction to abstract harmonic analysis. *New York*, 1954. Citado na página 61.

MANTOGLOU, A.; WILSON, J. L. The turning bands method for simulation of random fields using line generation by a spectral method. *Water Resources Research*, Wiley Online Library, v. 18, n. 5, p. 1379–1394, 1982. Citado na página 85.

MARCOTTE, D. Fast variogram computation with fft. *Computers & Geosciences*, Elsevier, v. 22, n. 10, p. 1175–1186, 1996. Citado 4 vezes nas páginas 34, 40, 52 e 62.

MARIETHOZ, G. A general parallelization strategy for random path based geostatistical simulation methods. *Computers & Geosciences*, Elsevier, v. 36, n. 7, p. 953–958, 2010. Citado 3 vezes nas páginas 34, 35 e 36.

MARIETHOZ, G.; RENARD, P.; STRAUBHAAR, J. The direct sampling method to perform multiple-point geostatistical simulations. *Water Resources Research*, Wiley Online Library, v. 46, n. 11, 2010. Citado na página 35.

MATHERON, G. Principles of geostatistics. *Economic geology*, Society of Economic Geologists, v. 58, n. 8, p. 1246–1266, 1963. Citado na página 35.

MATHERON, G. The intrinsic random functions and their applications. *Advances in applied probability*, JSTOR, p. 439–468, 1973. Citado 2 vezes nas páginas 69 e 71.

MISHRA, S. K. Optimal solution of the nearest correlation matrix problem by minimization of the maximum norm. *Available at SSRN 573241*, 2004. Citado na página 63.

MISHRA, S. K. The nearest correlation matrix problem: Solution by differential evolution method of global optimization. *Available at SSRN 980403*, 2007. Citado na página 64.

MORELAND, K.; ANGEL, E. The fft on a gpu. In: EUROGRAPHICS ASSOCIATION. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. [S.l.], 2003. p. 112–119. Citado na página 64.

MORITZ, P.; NISHIHARA, R.; JORDAN, M. A linearly-convergent stochastic l-bfgs algorithm. In: *Artificial Intelligence and Statistics*. [S.l.: s.n.], 2016. p. 249–258. Citado na página 173.

NUNES, R.; ALMEIDA, J. A. Parallelization of sequential gaussian, indicator and direct simulation algorithms. *Computers & Geosciences*, Elsevier, v. 36, n. 8, p. 1042–1052, 2010. Citado na página 35.

PARDO-IGUZQUIZA, E.; CHICA-OLMO, M. The fourier integral method: an efficient spectral method for simulation of random fields. *Mathematical Geology*, Springer, v. 25, n. 2, p. 177–217, 1993. Citado 2 vezes nas páginas 34 e 37.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V. et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, v. 12, n. Oct, p. 2825–2830, 2011. Citado na página 113.

PEREDO, O.; ORTIZ, J. M. Parallel implementation of simulated annealing to reproduce multiple-point statistics. *Computers & geosciences*, Elsevier, v. 37, n. 8, p. 1110–1121, 2011. Citado na página 34.

PYLE, D. *Data preparation for data mining*. [S.l.]: morgan kaufmann, 1999. Citado na página 113.

PYRCZ, M. J.; DEUTSCH, C. V. Semivariogram models based on geometric offsets. *Mathematical geology*, Springer, v. 38, n. 4, p. 475–488, 2006. Citado 2 vezes nas páginas 63 e 125.

QI, H.; SUN, D. A quadratically convergent newton method for computing the nearest correlation matrix. *SIAM journal on matrix analysis and applications*, SIAM, v. 28, n. 2, p. 360–385, 2006. Citado na página 63.

RADER, C. M. Discrete fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, IEEE, v. 56, n. 6, p. 1107–1108, 1968. Citado 2 vezes nas páginas 49 e 57.

RASERA, L. G.; MACHADO, P. L.; COSTA, J. F. C. A conflict-free, path-level parallelization approach for sequential simulation algorithms. *Computers & Geosciences*, Elsevier, v. 80, p. 49–61, 2015. Citado 2 vezes nas páginas 36 e 69.

RAVÉ, E. G. D.; JIMÉNEZ-HORNERO, F. J.; ARIZA-VILLAVERDE, A. B.; GÓMEZ-LÓPEZ, J. Using general-purpose computing on graphics processing units (gpgpu) to accelerate the ordinary kriging algorithm. *Computers & Geosciences*, Elsevier, v. 64, p. 1–6, 2014. Citado na página 36.

ROQUE, C.; FERREIRA, A. J. Numerical experiments on optimal shape parameters for radial basis functions. *Numerical Methods for Partial Differential Equations: An International Journal*, Wiley Online Library, v. 26, n. 3, p. 675–689, 2010. Citado na página 118.

RUGGIERO, M. A. G.; LOPES, V. L. d. R. *Cálculo numérico: aspectos teóricos e computacionais*. [S.l.]: Makron Books do Brasil, 1997. Citado 2 vezes nas páginas 170 e 172.

SHI, L.; CAMPBELL, G.; JONES, W. D.; CAMPAGNE, F.; WEN, Z.; WALKER, S. J.; SU, Z.; CHU, T.-M.; GOODSAID, F. M.; PUSZTAI, L. et al. The microarray quality control (maqc)-ii study of common practices for the development and validation of microarray-based predictive models. *Nature biotechnology*, Nature Publishing Group, v. 28, n. 8, p. 827, 2010. Citado na página 122.

SOARES, A. Direct sequential simulation and cosimulation. *Mathematical Geology*, Springer, v. 33, n. 8, p. 911–926, 2001. Citado na página 34.

STONEBRAKER, M. The case for shared nothing. *IEEE Database Eng. Bull.*, Citeseer, v. 9, n. 1, p. 4–9, 1986. Citado na página 33.

STORN, R.; PRICE, K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, Springer, v. 11, n. 4, p. 341–359, 1997. Citado na página 64.

STRAUBHAAR, J.; RENARD, P.; MARIETHOZ, G.; FROIDEVAUX, R.; BESSON, O. An improved parallel multiple-point algorithm using a list approach. *Mathematical Geosciences*, Springer, v. 43, n. 3, p. 305–328, 2011. Citado na página 35.

STREBELLE, S. Conditional simulation of complex geological structures using multiple-point statistics. *Mathematical Geology*, Springer, v. 34, n. 1, p. 1–21, 2002. Citado na página 34.

TAHMASEBI, P.; SAHIMI, M.; MARIETHOZ, G.; HEZARKHANI, A. Accelerating geostatistical simulations using graphics processing units (gpu). *Computers & Geosciences*, Elsevier, v. 46, p. 51–59, 2012. Citado na página 34.

TAKAHASHI, D. A hybrid mpi/openmp implementation of a parallel 3-d fft on smp clusters. In: SPRINGER. *International Conference on Parallel Processing and Applied Mathematics*. [S.l.], 2005. p. 970–977. Citado na página 64.

TILKOV, S.; VINOSKI, S. Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, IEEE Computer Society, v. 14, n. 6, p. 80, 2010. Citado na página 33.

VARGAS, H.; CAETANO, H.; FILIPE, M. Parallelization of sequential simulation procedures. In: *EAGE Conference on Petroleum Geostatistics*. [S.l.: s.n.], 2007. Citado na página 35.

WILKINSON, B.; ALLEN, M. *Parallel programming*. [S.l.]: Prentice hall New Jersey, 1999. v. 999. Citado na página 37.

WINOGRAD, S. On computing the discrete fourier transform. *Mathematics of computation*, v. 32, n. 141, p. 175–199, 1978. Citado na página 57.

WIRTH, N. A plea for lean software. *Computer*, IEEE, v. 28, n. 2, p. 64–68, 1995. Citado na página 31.

XIONG, J.; ZOLOTOV, V.; HE, L. Robust extraction of spatial correlation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, IEEE, v. 26, n. 4, p. 619–631, 2007. Citado na página 63.

YAO, R.; YANG, J.; SHAO, H. Accuracy and uncertainty assessment on geostatistical simulation of soil salinity in a coastal farmland using auxiliary variable. *Environmental monitoring and assessment*, Springer, v. 185, n. 6, p. 5151–5164, 2013. Citado na página 95.

YAO, T. *Automatic covariance modeling and conditional spectral simulation with Fast Fourier Transform*. [S.l.: s.n.], 1998. Citado na página 64.

YAO, T. Conditional spectral simulation with phase identification. *Mathematical Geology*, v. 30, n. 3, p. 285–308, 1998. Citado 3 vezes nas páginas 69, 83 e 84.

YAO, T.; JOURNEL, A. G. Automatic modeling of (cross) covariance tables using fast fourier transform. *Mathematical Geology*, v. 30, n. 6, p. 589–615, 1998. Citado 6 vezes nas páginas 34, 63, 64, 65, 68 e 69.

YIN, J.-F.; ZHANG, Y. Alternative gradient algorithms for computing the nearest correlation matrix. *Applied Mathematics and Computation*, Elsevier, v. 219, n. 14, p. 7591–7599, 2013. Citado na página 63.

ZAGAYEVSKIY, Y.; DEUTSCH, C. V. Multivariate geostatistical grid-free simulation of natural phenomena. *Mathematical Geosciences*, Springer, v. 48, n. 8, p. 891–920, 2016. Citado 2 vezes nas páginas 73 e 77.

Apêndices

APÊNDICE A – Alguns conceitos fundamentais de análise matemática

Todo algoritmo de estimativa ou otimização necessita de uma *métrica* (LIMA, 1977) d, isto é, uma função matemática que meça a distância entre dois candidatos à solução. Esses candidatos (que podem ser objetos matemáticos como vetores, matrizes, funções, etc...) são representados, sem perda de generalidade, por pontos $v \in D$, onde D é o domínio do problema. O conjunto D, combinado com uma métrica d, é um *espaço métrico* (D, d). Uma métrica $d : D \to \mathbb{R}$ possui as seguintes propriedades:

1. $d(u, w) + d(w, v) \geq d(u, v)$, com $u, v, w \in D$ (desigualdade triangular);
2. $d(u, v) = 0$ se, e somente se, $u = v$.

Dos itens (1) e (2), conclui-se que $d(u, v) \geq 0$. A seguir, temos alguns exemplos de métrica:

- $d(u, v) = ||u - v||$, com $||\cdot||$ sendo uma norma no conjunto D. Um exemplo de norma, em $\mathbb{R}^N$, é a distância euclidiana com relação a origem $||u|| = ||(u_1, u_2, \ldots, u_N)|| = \sqrt{u_1^2 + u_2^2 + \cdots + u_N^2}$.
- $d(u, v) = \dfrac{||u - v||}{1 + ||u - v||}$.
- $d(u, v) = \max |u(x) - v(x)|$, com $u(x), v(x) \in D$ e D sendo um conjunto de funções reais limitadas, isto é, funções $f(x)$ para as quais existem um número real $L > 0$ tal que $|f(x)| < L$, para todo x no domínio de $f(x)$.
- $d(u, v) = \int_S |u(x) - v(x)|dx$, com $u(x)$ e $v(x)$ sendo funções integráveis no conjunto S.
- $d(u, v) = \sqrt[p]{\int_S |u(x) - v(x)|^p dx}$, esta métrica é conhecida como métrica L^p (a norma L^p é obtida de forma similar, $||u||_p = \sqrt[p]{\int_S |u(x)|^p dx}$). S é o domínio das funções $u(x)$ e $v(x)$.

Métricas são uma forma objetiva de avaliar a qualidade de uma solução obtida por algum algoritmo matemático, já que com uma métrica é possível medir o quão distante uma solução aproximada está da solução real. Sob certas condições, mesmo que não se

conheça a solução real, por meios indiretos e sendo satisfeitas certas condições, é possível identificar quando uma solução está convergindo para a solução real. Quando uma medida de erro é definida, implicitamente, um espaço métrico está sendo construído, onde teoremas específicos são válidos. O erro quadrático médio, por exemplo, pode ser visto como uma versão discreta da métrica L^2.

Em espaços vetoriais, é comum a definição do conceito de uma *norma* $||.||$, que é uma função com as seguintes características:

1. $||u|| = 0$, com $u \in D$ se, e somente se, $u = 0$. Onde 0 é o elemento nulo do espaço vetorial D.
2. $||u|| + ||v|| \geq ||u + v||$ (desigualdade triangular);
3. $||cu|| = |c|||u||$, com $c \in \mathbb{R}$ e $u \in D$ (homogeneidade).

Um espaço vetorial com uma norma é chamado de *espaço normado*.

Além da *norma*, também é comum se definir o conceito de *produto interno* $< u, v >$. Um espaço vetorial com *produto interno* é chamado de *espaço unitário*. A função $< u, v >$: $D \times D \to \mathbb{C}$, com $u, v \in D$, associa um par de elementos do espaço vetorial D a um escalar $< u, v >$. Essa função possui as propriedades:

1. $< cu, v >= c < u, v >$, com $c \in \mathbb{R}$;
2. $< u + v, w >=< u, w > + < v, w >$;
3. $< u, v >= \overline{< v, u >}$, onde $\overline{< v, u >}$ é o conjugado complexo de $< u, v >$;
4. $< u, u >> 0$, se $u \neq 0$.

Uma *norma* pode ser construída a partir de um *produto interno* fazendo-se $||u|| = \sqrt{< u, u >}$. É fácil observar que todo espaço vetorial normado é um espaço métrico, bastando fazer $d(u, v) = ||u - v||$. Embora a recíproca não seja verdadeira, como pode ser visto em (LIMA, 1977).

Combinando os conceitos de norma e produto interno é possível definir um conceito mais geral de ângulo entre vetores de um espaço vetorial qualquer:

Definição A.0.1 *Seja $u, v \in D$, vetores não-nulos e D um espaço normado e unitário. O $cos(\theta)$, isto é, o cosseno do ângulo θ entre os vetores u e v é definido por:*

$$cos(\theta) = \frac{< u, v >}{||u||||v||}.$$

A norma adiciona o conceito de comprimento a um conjunto e junto com o *produto interno*, que adiciona o conceito de ângulo entre vetores, esses conceitos permitem a construção de um sistema de coordenadas para o espaço vetorial D. Esse é um resultado fundamental que é utilizado em otimização na construção de técnicas para se navegar no domínio do problema sendo resolvido, permitindo identificar em que direção um mínimo local pode estar localizado.

Outra definição fundamental, construída a partir do conceito de espaço métrico é a *sequência de Cauchy*:

Definição A.0.2 *Uma sequência de elementos u_n de um espaço métrico D é dita de Cauchy, se $\lim_{n,m\to\infty} d(u_n, u_m) = 0$. Isto é, se a partir de um determinado índice $K > 0$ os termos da sequência ficam tão próximos quanto se queira. (ALBRECHT, 1973).*

É importante notar que toda sequência v_n definida em um espaço métrico D que converge para um valor $y \in D$ é uma *sequência de Cauchy*, mas a recíproca nem sempre é verdadeira, Como nos exemplos a seguir, onde as sequências convergem para valores fora do conjunto D:

1. $D : (0, 1], d(u, v) = |u - v|, u_n = \dfrac{1}{n}, n = 1, 2, 3, \ldots;$
2. $D = \mathbb{Q}, d(u, v) = |u - v|, u_n = (1 + \dfrac{1}{n})^n.$

Um subespaço métrico $D_1 \subset D$ é dito *completo* se toda sequência de *Cauchy* em D_1 converge para elementos de D_1.

A partir desse conjunto de definições pode-se apresentar os conceitos de espaços de *Banach* e *Hilbert*.

1. Um espaço normado e completo é chamado de *espaço de Banach*;
2. Um espaço unitário e completo é chamado de *espaço de Hilbert*.

Esses dois espaços possuem enorme aplicação prática, já que existem teoremas muito úteis, baseados nesses espaços, que são verdadeiros para diferentes tipos de objetos matemáticos (como equações diferenciais, grafos, funções, matrizes, tensores e vetores). O usuário desses teoremas só precisa, a partir de um espaço vetorial qualquer, adicionar o conceito de norma e produto interno.

Métodos iterativos fundamentais são baseados em teoremas de convergência definidos em espaços de *Banach* e/ou *Hilbert*. Um exemplo clássico, é o teorema de Picard, sobre existência e unicidade de soluções de equações diferenciais ordinárias com condições iniciais,

que utiliza um teorema de ponto fixo definido em um espaço de *Banach* para demonstrar que um processo iterativo converge para a solução única da equação (FIGUEIREDO, 1979).

Ponto fixo é um conceito importante usado na construção de métodos iterativos. Um ponto fixo u associado a uma função $T : D \to D$ é um vetor u que satisfaz a equação $u = T(u)$. Além disso, um método iterativo pode ser definido como sendo a sequência $u_{n+1} = T(u_n)$. Se $d(T(u_1), T(u_2)) \leq Pd(u_1, u_2)$, com $u_1, u_2 \in D$ e $0 \leq P < 1$, $T(u)$ é chamado de *operador de contração* ou, simplesmente, uma *contração* em D. A maioria das técnicas de solução numérica de problemas de estimativa, equações, sistemas de equações, equações diferenciais, otimização, entre outros problemas comuns em matemática, pode ser traduzida como sendo métodos iterativos. Esses métodos convergem para alguma solução ou mínimo local sob certas condições definidas em teoremas de análise matemática. O trabalho do analista numérico é ser capaz de traduzir um algoritmo em um formato em que possa ser aplicado teoremas de convergência, como o teorema a seguir (cuja demonstração pode ser encontrada em (ALBRECHT, 1973)):

Teorema A.0.1 *Seja D um espaço métrico e T uma contração definida em D_1, um subespaço completo de D. Então, a equação $x = T(x)$ possui uma única solução $x = u$ em D_1 e a sequência $u_{n+1} = T(u_n)$ tomando elementos em D_1 converge pra u. Além disso, o erro pode ser estimado usando a equação:*

$$d(u_{n+1}, u) \leq \frac{P}{1 - P} d(u_{n+1}, u_n).$$

O Teorema A.0.1 fornece um resultado fundamental usado para estimativa de erro e para demonstrar convergência local em um vizinhança de uma solução. Ao mesmo tempo que esse teorema demonstra a capacidade de métodos iterativos convergirem para uma solução local de um problema, não se tem garantia de que essa solução será a melhor para o problema sendo estudado. Teoremas de convergência global exigem muitas assunções, que podem não ser satisfeitas na prática. Por isso, muito esforço de pesquisa tem sido investido em estratégias para escapar das armadilhas de mínimos locais, permitindo uma combinação de técnicas iterativas locais e técnicas para explorar melhor o domínio completo da solução.

Essa seção apenas apresenta alguns conceitos básicos de análise matemática. Este campo de estudo é bastante amplo e possui resultados fundamentais para a construção de soluções eficientes de problemas matemáticos complexos. Por isso, recomenda-se uma leitura futura mais detalhada desse tema, principalmente de textos como (FRÖBERG, 1969), que fornecem uma boa aplicação de conceitos de análise matemática a problemas numéricos.

A Seção A.1, apresenta alguns métodos iterativos clássicos amplamente utilizados em diversos algoritmos numéricos e técnicas de aprendizado de máquina. Muitos algoritmos

modernos são evolução dos algoritmos apresentados nesta seção, principalmente os métodos baseados em gradiente. Nas seções posteriores, é apresentado uma variação de métodos iterativos aplicados a estimativa geoestatística e otimização de modelos de covariância.

A.1 Métodos iterativos para otimização e solução de equações

Um problema fundamental em matemática é o de se encontrar raízes de funções $f : D \to I$, onde D é o domínio de f e I é a imagem de f. Isto é, valores $x \in D$, para os quais $f(x) = 0$, onde 0 é o elemento nulo do conjunto I. Nesta seção, assume-se que o conjunto I sempre contém o elemento nulo 0.

Se x for um vetor definido em $\mathbb{R}^N$ e $f(x)$, um vetor definido em $\mathbb{R}^M$, $f(x) = 0$ define um sistema com M equações escalares $f_i(x) = 0, i = 1, 2, \ldots, M$. A seguir, têm-se alguns exemplos de sistemas de equações escalares definidas em $\mathbb{R}^N$:

1. Exemplo de sistema de equações não-lineares:

$$\begin{cases} xy - 2 = 0, \\ x + y^3 - 3xy + 37 = 0, \\ xz + yz - 31 = 0; \end{cases}$$

2. Exemplo de sistema de equações lineares

$$\begin{cases} x + 3y - 2 = 0, \\ x + y = 0, \\ x - z - 4 = 0. \end{cases}$$

Caso D seja um espaço de funções, temos exemplos de sistemas de equações diferenciais como:

1. Sistema de equações diferenciais lineares:

$$\begin{cases} u' + v' - 3u = 0 \\ u' + 2v' - 2v = 0 \end{cases}$$

2. Sistema de equações diferenciais não-lineares:

$$\begin{cases} (u')^2 + vv' - 3vu = 0 \\ \log u' + 2\sqrt{v'} - 2v^3 = 0 \end{cases}$$

Infelizmente, somente um conjunto restrito de sistemas de equações possuem métodos exatos de resolução. Além disso, muitos problemas práticos estão associados a resolução de um sistema de equações não-lineares. Por isso, um ramo fundamental da matemática, que se baseia em resultados sólidos construídos pela análise matemática, é o ramo da análise numérica. Um de seus objetos de estudo mais importantes são os métodos iterativos.

Conforme visto na Seção A, métodos iterativos são técnicas que, a partir de um *valor inicial* u_0, constroem uma sequência $u_{n+1} = T(u_n), n = 1, 2, \ldots$, que sob condições específicas converge para uma possível solução do problema. O operador T representa a lógica do método iterativo. Nessa seção, serão apresentados dois métodos que são base para a construções de muitos algoritmos numéricos sofisticados: o *Método de Newton* e o *Método de Broyden* (RUGGIERO; LOPES, 1997). Os conceitos utilizados na construção desses métodos iterativos fornecem a inspiração dos algoritmos evolutivos apresentados nas próximas seções.

A.1.1 O Método de Newton

Nesta seção, é apresentado um método iterativo clássico utilizado na resolução dos sistemas de equações $f(u) = 0$ apresentados na seção anterior. Trata-se do método de Newton, que é utilizado como base para a maioria dos métodos baseados em gradiente. Esse método se fundamenta na resolução da sequência de problemas definida pela eq. (A.1):

$$\begin{cases} J(u_n)s_n = -f(u_n); \\ u_{n+1} = u_n + s_n. \end{cases} \tag{A.1}$$

Onde $J(u_n)$ é a *matriz Jacobiana* da função f avaliada no ponto u_n. A matriz Jacobiana é uma matriz $M \times N$ definida pela eq. (A.2).

$$J(u) = \begin{vmatrix} \frac{\partial f_1}{\partial u_1}(u) & \frac{\partial f_1}{\partial u_2}(u) & \ldots & \frac{\partial f_1}{\partial u_N}(u) \\ \vdots & \ldots & \ldots & \vdots \\ \frac{\partial f_M}{\partial u_1}(u) & \frac{\partial f_M}{\partial u_2}(u) & \ldots & \frac{\partial f_M}{\partial u_N}(u) \end{vmatrix}, \tag{A.2}$$

onde $u = [u_1, u_2, ..., u_N]$ e $f(u) = [f_1(u), f_2(u), ..., f_M(u)]$ é uma função diferenciável.

Quando $M = N$, a resolução iterativa da eq. (A.1) se reduz a uma sequência de resolução de sistemas lineares, caso contrário, precisa-se resolver um problema de regressão linear via mínimos quadrados. Na prática, o método de Newton termina sua execução quando $f(u_n) < \epsilon$, $|u_{n+1} - u_n| < \epsilon$ ou $n > K$, com $K \in \mathbb{N}$. Ou seja, esse método termina quando $f(u_n)$ converge para uma solução de $f(u) = 0$, isto é, $u_n \to u$, ou quando u_n

converge para um valor w, não necessariamente solução do problema, ou quando n se torna grande o suficiente. Nessas duas últimas situações, diz-se que o método divergiu (caso a condição de parada seja $n > K$) ou atingiu um mínimo local (caso $|u_{n+1} - u_n| < \epsilon$, com $f(u_n) \geq \epsilon$ e $J(u_n) \approx 0$).

A convergência do método de Newton depende da qualidade do valor inicial u_0, que precisa ser de preferência próximo da solução real do problema. Isto é, como muitos métodos iterativos, a convergência do método de Newton é extremamente sensível à qualidade do candidato inicial. Por isso, na prática, é comum a utilização de soluções de versões simplificadas do problema estudado como uma solução inicial para o método de Newton. Pode-se ver o método de Newton, nesse contexto, como uma técnica para incorporar complexidade e características adicionais a uma solução mais básica obtida anteriormente. Em geoestatística, por exemplo, pode-se utilizar soluções estacionárias obtidas por métodos lineares clássicos, como a krigagem, para se gerar essas soluções iniciais u_0. Justamente essa é a inspiração que o método de Newton fornece para outras metodologias iterativas de resolução de problemas matemáticos. Nas próximas seções desse capítulo, essa inspiração será aplicada no problema de utilização dos erros de validação cruzada para construção de modelos geoestatísticos e no problema de otimização de modelos de covariância (ambos problemas são inter-relacionados, e suas soluções podem ser relacionadas para se construir um processo iterativo de construção de soluções melhores).

É importante lembrar que problemas de minimização, como o definido na eq. (A.3) podem ser reduzidos a problemas de busca de raízes, fazendo-se $f(x) = \nabla g(x) = 0$. Além disso, restrições de igualdade $h(x) = 0$ e de desigualdade $b(x) < 0$, podem ser adicionadas ao problema via adição de multiplicadores de Lagrange e condições de Karush-Kuhn-Tucker (ou condições KKT) (BOYD; BOYD; VANDENBERGHE, 2004). Esses tipos de problemas constituem o campo de *programação não linear*, que possui uma série de algoritmos essenciais para resolução de complexos problemas de otimização que ocorrem na prática.

$$\min g(x) \tag{A.3}$$

Muitos problemas em geoestatística, como a incorporação de características não-estacionárias nos modelos estimados e a identificação da melhor estrutura de covariância obtida a partir de amostras de covariância experimental, podem ser transformados em problemas de programação não linear. Isso é amplamente explorado ao longo desse trabalho. Nesta tese, o foco é dado às técnicas não baseadas em gradientes, como a busca tabu e o *ensemble estimator*, embora técnicas como o metodo de Newton possam ser utilizadas indiretamente na geração de vizinhos (no caso da busca tabu) ou na otimização do parâmetro de forma ou do modelo de covariância local (no caso do *ensemble estimator*).

Uma importante observação prática é que nem sempre é possível definir a expressão

da derivada da função f, por exemplo, no caso em que f é definida implicitamente por amostragem (como nos problemas de minimização de erro). Nesses casos, é comum a utilização de uma aproximação numérica da derivada parcial $\frac{\partial f}{\partial u_i}$ por meio de uma equação de diferenças como as eq. (A.4) ou (A.5) (onde $\epsilon > 0$ é a discretização da derivada).

$$\frac{\partial f}{\partial u_i}(u) \approx Df_\epsilon(u) = \frac{f(u + \epsilon I_i) + f(u)}{\epsilon} \tag{A.4}$$

$$\frac{\partial f}{\partial u_i}(u) \approx Df_\epsilon(u) = \frac{f(u + \epsilon I_i) + f(u - \epsilon u_i)}{2\epsilon}, \tag{A.5}$$

onde $I_i = [0, ..., 1, ..., 0]$ é um vetor indicador, isto é vetor com valor 1 somente no índice i e zero nos outros valores e ϵ é um valor tão pequeno quanto se queira.

O método de Newton possui grandes virtudes, mas pode possuir uma convergência lenta e é muito vulnerável à arestas, isto é, regiões em que as derivadas parciais de f podem apresentar valores muito elevados ou em que a matriz jacobiana J associada ao problema seja uma matriz singular (impossibilitando a computação do valor s_n na eq. (A.1)). Nessa situação, recomenda-se a utilização de métodos menos dependentes de gradiente, como métodos baseado em secante (por exemplo, o método de Broyden, visto na subseção A.1.2), ou métodos *gradient-less* (como a busca tabu, visto na Seção 5.3).

Por fim, é importante comentar que essa metodologia, também, pode ser aplicada na resolução de sistema de equações diferenciais, conforme pode ser visto em (ALBRECHT, 1973).

A.1.2 Método de Broyden

Para finalizar essa sucinta apresentação de algumas técnicas importantes de análise numérica, nesta subseção, é apresentado o método de *Broyden* (RUGGIERO; LOPES, 1997) que não necessita da computação da matriz jacobiana J em cada iteração do algoritmo. Além disso, em situações práticas, esse algoritmo apresenta convergência mais rápida e é mais robusto a arestas e regiões onde a derivada não é definida. A intuição desse método se baseia no fato de que a secante de uma curva também fornece informação sobre a localização de um mínimo local e pode, portanto, substituir a derivada durante o processo

iterativo. A eq. (A.6) apresenta o método de Broyden.

$$\begin{cases} B_0 = J(u_0) \\ B_k s_k = -f(u_k) \\ x_{k+1} = x_k + s_k \\ y_k = f(x_{k+1}) - f(x_k) \\ u_k = \dfrac{y_k - B_k s_k}{||s_k||} \\ B_{k+1} = B_k + u_k (s_k)^T, k = 0, 1, 2, \ldots \end{cases} \tag{A.6}$$

O método de Broyden é uma generalização de técnicas baseadas no teorema de valor médio que aproxima a derivada de uma função por meio da reta secante. Na prática, esses métodos baseados em secante, além de serem mais robustos, apresentam melhor performance do que técnicas puramente baseadas em derivadas (gradientes). Pode-se dizer que o método de Broyden e sua variação mais moderna, o algoritmo BFGS (Broyden–Fletcher–Goldfarb–Shanno) (MORITZ; NISHIHARA; JORDAN, 2016), encontram-se no meio termo entre técnicas *gradient-less* e as mais tradicionais. A variação LBFGS (*Limited Memory* BFGS, BFGS com memória) tem se tornado uma solução popular em problemas com milhares de variáveis.

APÊNDICE B – Alguns comentários sobre detalhes de implementação de algoritmos de otimização numérica

A grosso modo, pode-se dizer que as técnicas numéricas de otimização transformam problemas contínuos em um processo iterativo que consiste na resolução de uma série de problemas lineares. Então, aqui se faz necessário a utilização de boas bibliotecas numéricas para se garantir bons resultados, já que tanto a estabilidade numérica, a qualidade da solução e a performance do algoritmo estão extretamente atreladas à qualidade dos resolvedores lineares utilizados. Neste trabalho, o algoritmo BDCSVD (GU; EISENSTAT, 1995), implementado no pacote *Eigen* (GUENNEBAUD; JACOB et al., 2014), foi utilizado na resolução dos subproblemas de mínimos quadrados associados.

Outra observação importante é que, numericamente, o conceito de derivada pode ser substituído por qualquer métrica que indique o quanto uma função varia dado a variação de um parâmetro específico. Aqui, por exemplo, pode-se utilizar alguma medida estatística como a variação média de uma função empírica de erro com relação a variação de algum parâmetro. Tanto que existe uma definição de derivada mais geral, chamada de *derivada de Fréchet* (ALBRECHT, 1973):

Definição B.0.1 *Sejam D e D_1 dois espaços de Banach e $T : D_1 \to D$ um operador com argumentos em $D_1 \subset D$. T é diferenciável segundo Fréchet em u, $u \in D$, se para um dado $||h||$ existem um operador linear $T_u^{'}$ e uma função real $\epsilon(\delta)$, tal que:*

$$||T(u+h) - T(u) - T_u^{'}|| \leq ||h||\epsilon(||h||), \lim_{\delta \to 0} \epsilon(\delta) = 0. \tag{B.1}$$

Intuitivamente, o conceito de derivada de Fréchet nos permite substituir uma derivada "de verdade" (isto é, analítica e exata), que pode ser impossível de ser computada devido a baixa qualidade dos dados ou baixa amostragem, por alguma função que se comporte de forma similar dado a discretização espacial do problema, dada por $||h||$. Na prática, isso significa que a substituição da derivada por funções secante, e outras medidas de variação relativa comuns em estatística, são aceitáveis. Esse fato nos fornece alguma solidez teórica para aplicar os algoritmos de análise numérica em problemas de otimização com dados ruidosos e amostrados (situações em que não temos como usar uma derivada bem definida).

Nesta seção, foi apresentada uma introdução bem sucinta de algumas técnicas de análise numérica fundamentais. A intuição adquirida com os métodos iterativos será amplamente usada para se construir algoritmos práticos para resolver problemas de geoestatística, conforme é visto nas seções seguintes. É importante notar que há uma relação de simbiose entre algoritmos baseados em gradiente (e/ou alguma noção de derivada) e algoritmos de busca estocástica como os algoritmos genéticos e baseados em busca tabu. Os métodos iterativos clássicos fornecem excelentes estratégias para se computar um mínimo local a partir de palpites iniciais fornecido pelos algoritmos de busca estocástica. Esse fato é explorado na implementação prática dos algoritmos apresentados neste trabalho.

Por fim, neste trabalho, o conceito de validação cruzada foi utilizado para se construir uma medida de variação que exerce um papel similar ao da derivada de Fréchet. Então, muitos dos conceitos apresentados nessa seção podem ser utilizados em conjunção com as métricas definidas nas seções seguintes.