MINISTÉRIO DA EDUCAÇÃO

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Escola de Engenharia

Programa de Pós-Graduação em Engenharia de Minas,

Metalúrgica e de Materiais (PPGE3M)

DECOMPOSIÇÃO AUTOMATIZADA DE IMAGENS EM FATORES GEOMORFOLÓGICOS

Paulo Roberto Moura de Carvalho

Tese para obtenção de título de

Doutor em Engenharia

Porto Alegre, RS

2020

MINISTÉRIO DA EDUCAÇÃO

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Escola de Engenharia

Programa de Pós-Graduação em Engenharia de Minas,

Metalúrgica e de Materiais (PPGE3M)

DECOMPOSIÇÃO AUTOMATIZADA DE IMAGENS EM FATORES GEOMORFOLÓGICOS

Paulo Roberto Moura de Carvalho

Tese realizada no Laboratório de Pesquisa Mineral e Planejamento Mineiro da Escola de Engenharia da UFRGS, dentro do programa de Pós-Graduação em Engenharia de Minas, Metalúrgica e de Minerais (PPGE3M), como parte dos requisitos para a obtenção de Título de Doutor em Engenharia.

Área de Concentração: Tecnologia Mineral, Ambiental e Metalurgia Extrativa

Porto Alegre, RS

2020

Esta tese foi julgada adequada para a obtenção do Título de Doutor em Engenharia e aprovada em sua forma final, pelo Orientador e pela Banca Examinadora do Curso de Pós-Graduação.

Orientador:

Prof. Dr. João Felipe Coimbra Leite Costa

Coorientador:

Prof. Dr. Amílcar Soares

Banca Examinadora:

Dr. Áttila Leães Rodrigues

Prof. Dr. Diego Machado Marques

Prof. Dr. Leonardo Azevedo Guerrar Raposo Pereira

Prof. Dr. Afonso Reguly

Coordenador do PPGE3M

Agradecimentos

Ao Prof. Dr. João Felipe, meu orientador e lente em quase todas as disciplinas da área de geoestatística nos programas de mestrado e doutorado no PPGE3M. Nesses oito anos tivemos oportunidades de participar de congressos, apresentar trabalhos e de publicar artigos juntos. Com um pé firme na teoria e outro na indústria, aprendi todo um mundo de conhecimento que até minha primeira disciplina cursada era totalmente desconhecido. Devo-lhe ainda uns dois ou três almoços em alguma boa churrascaria de Porto Alegre! O Prof. João Felipe é certamente um daqueles gigantes a que aludiu Newton em sua famosa frase, sobre cujos ombros eu pude ver mais longe.

Ao Dr. Péricles Machado, colega meu durante os cursos de mestrado e doutorado no PPGE3M, pelas ótimas contribuições tanto em teoria quanto implementação em C++ de algoritmos de análise espectral de imagens amplamente empregados no desenvolvimento desta tese. Contribuições sem as quais eu teria sérios obstáculos práticos. Quem já tentou implementar ou usar uma API de Transformada de Fourier, por exemplo, sabe que não é trivial e que muita vez os autores das documentações, artigos e livros omitem os famosos "pulos do gato".

Ao Dr. Alexandre Boucher, por ter gentilmente compartilhado a API de métodos espectrais do software Ar2GeMS da Ar2Tech para inclusão no software livre e de código aberto GammaRay, criado para conduzir os experimentos desta tese. Essa ajuda poupou uma enorme quantidade de tempo de escritura de código.

Aos diversos colegas do PPGE3M, como o Dr. Áttila e o Dr. Marcel, pelos conhecimentos das áreas de ciência da computação, geologia e geofísica presentes neste trabalho de natureza multidisciplinar, que me ajudaram na compreensão, com algumas figuras e indicação de leituras nessas áreas. Alguns dos quais hoje são professores da UFRGS com quem tive o prazer de cursar algumas disciplinas. Pelo visto, o Prof. João Felipe fez um excelente trabalho deixando um legado sob a forma de um novo corpo docente para dar continuidade a seu belo trabalho.

Ao pessoal administrativo do PPGE3M, pelo apoio, presteza e paciência sempre que eu precisava de algum trâmite ou documento estando eu fisicamente longe da UFRGS.

Aos meus pais, pelo estímulo à leitura desde a pré-escola e pelo apoio e interesse no acompanhamento desta jornada. Especialmente à minha mãe, a Sra. Leomar, que, enquanto

eu escrevia estas páginas, deixou este mundo em paz e fazendo o que amava, que era viajar e estar perto do mar. Na sua simplicidade, dizia ela que nada entendia do meu trabalho e do que eu escrevia. Nada que um pouco de tempo e a didática singular do Prof. João Felipe não resolvesse. Por outro lado, nunca entenderei o que é ser mãe, por melhores mestres eu tenha e por mais teses eu escreva.

Às seguintes pessoas que foram determinantes em decisões importantes ao longo da minha vida que me levaram até aqui: Sr. Neivaldo, especialista em Unix da antiga Irwin Industrial e Comercial LTDA pela minha primeira boa oportunidade de emprego; Sérgio Hirohak Kawai, o então estagiário de engenharia da computação na Irwin que me estimulou a fazer vestibular para Ciência da Computação; Prof. Dr. Paulo Eustáquio, da UERJ, cujo último ensinamento a mim dado na festa de formatura foi "trabalhe bem e o sucesso virá como consequência". Se a vida é como uma viagem de trem compartilhada com pessoas que embarcam e desembarcam em momentos diferentes, esses foram os viajores que me fizeram mudar de trem.

Ao Criador, pela vida.

Sumário

Lista de figuras

Lista de tabelas

Lista de equações

Lista de Abreviaturas e Definições

backend – Jargão usado em informática para se referir ao software que executa "nos bastidores", ou seja, a parte do sistema com a qual o usuário não interage. Normalmente o software chamado de *frontend* admite o intercâmbio de vários *backends* de forma totalmente transparente para o usuário.

head – no cálculo do variograma experimental, refere-se à variável usada como primeira no cômputo da semivariância. Ver *tail*. Se *head* = *tail*, diz-se que é um autovariograma, caso não, diz-se que é um variograma cruzado.

lag – termo usado para se referir a separação entre dois pontos no cálculo do variograma experimental.

spline – termo originalmente usado para denominar réguas de material flexível especial usadas em desenho técnico manual as quais, quando fixadas em determinados pontos, produziam uma curva como uma parábola, catenária ou cúbica, dependendo da fixação dos pontos. No contexto da geometria computacional, o termo se refere à curva obtida pelo processo numérico de interpolação que deve passar por determinados pontos.

tail – no cálculo do variograma experimental, refere-se à variável usada como segunda no cômputo da semivariância. Ver *head*. Se *head* = *tail*, diz-se que é um autovariograma, caso não, diz-se que é um variograma cruzado.

Resumo

Nos estudos das geociências, dados de amostragem regular (ex.: imagens) são observações de estruturas geológicas. Assim, espera-se que elas retratem a grande riqueza de feições e complexidade estrutural que a natureza apresenta. Remover ruído puramente aleatório desses dados (sem uma estrutura espacial) é uma tarefa trivial. Separar feições com estruturas espaciais diferentes, por exemplo, variações de relevo da escala de 10km das variações na de 1km, é de crescente interesse na atividade de geomodelagem. No entanto, esse objetivo requer um algoritmo sofisticado. Os métodos existentes para essa tarefa requerem uma parametrização cujo ajuste é tarefa laboriosa, por vezes tediosa, e objeto de interpretação. Após visitar os principais métodos para decomposição estrutural manual e com aprendizado de máquina, esta tese propõe um algoritmo que automatiza a decomposição estrutural baseando-se no variograma. Ou seja, um dado de entrada cujo variograma apresente múltiplas estruturas com diferentes escalas e anisotropias é decomposto automaticamente em imagens cujos variogramas individuais correspondem a cada estrutura. Software de código aberto foi escrito para testar o algoritmo proposto com um dado sintético cujo variograma contém quatro estruturas imbricadas. Os resultados mostraram eficácia semelhante ao da krigagem fatorial, porém de forma automatizada e com bom desempenho computacional.

Abstract

In the geoscience studies, regularly sampled data (e.g. images) are observations of geologic structures. Hence, it is expected that they bear the great variety of shapes and structural complexity that nature presents. Removing purely random noise from these data (without spatial structure) is a trivial task. Separating shapes with different spatial structures, for example, variations in relief with 10km scale from those with 1km scale, is of growing interest in the geomodeling practice. However, such objective requires a sophisticated algorithm. The existing methods for this task demand parameters of laborious, sometimes tedious, adjustment that is object of interpretation. After visiting the main methods of manual structural decomposition and with machine learning, this thesis proposes an algorithm that automatizes structural decomposition based on the variogram. That is, a given input data, whose variogram has multiple nested structures with different scales and anisotropies, is decomposed automatically into images with variograms corresponding to each nested structure. Open-source software was written to test the proposed algorithm on synthetic data with variogram bearing four nested structures. Results showed an effectiveness similar to factorial kriging, although by automatized way and with good computational performance.

Prólogo

Ao longo dos anos, muitas pessoas têm comentado que eu exponho as ideias de forma muito didática. Não sei se é verdade, mas essa é minha intenção. Nos anos 1980, eu assisti, como um garoto fascinado, à famosa série de televisão Cosmos, apresentada pelo maior divulgador da Ciência que eu conheço e com quem o leitor há de concordar: Carl Sagan. Foi Sagan que me ensinou a ver em 4 dimensões, algo que dava nó na minha mente. Não sei que mágica ele usa, mas Sagan consegue ensinar coisas aparentemente impossíveis aos não iniciados. Mais tarde, li alguns de seus livros e assisti às duas continuações de Cosmos apresentadas pelo seu pupilo, hoje astrofísico e divulgador de envergadura semelhante, Neil DeGrasse Tyson. Por forte influência deles, eu também acredito verdadeiramente em que a Ciência deva ser acessível a todos, bastando a vontade e a liberdade de buscá-la. No entanto, diferentemente de um livro ou de um programa de televisão, as fórmulas matemáticas herméticas devem estar presentes em um texto científico. Porém, por que elas não podem estar acompanhadas por muitas ilustrações coloridas e explicações simples? Como desdobramento desse desejo de não criar obstáculos desnecessários e para estimular o leitor a testar por si mesmo o que está exposto neste trabalho, todo o software empregado nesta tese foi construído inteiramente com tecnologia livre (*free as in beer* e *free as in speech*) e de código aberto. Por fim, meu desejo é que a leitura deste texto não seja um remédio contra insônia, pois um texto cuja leitura seja minimamente palatável ajuda na assimilação do conteúdo. Isso é particularmente verdadeiro e necessário em um trabalho de natureza multidisciplinar como este.

Capítulo 1

Introdução

Nos estudos das geociências, são comuns os dados em forma de malhas regulares tais como imagens de satélite, volumes sísmicos, mapas e fotografias de afloramentos. Esses dados de amostragem regular podem ser comumente referidos simplesmente por imagens. Normalmente, esses dados se apresentam no domínio espacial, mesmo que a escala vertical esteja em tempo no caso de dados sísmicos migrados (Rosa, 2010, pp. 392-490 e 591-597) em tempo [de reflexão das ondas sísmicas]. Sendo essas imagens de observações feitas de estruturas geológicas, espera-se que elas retratem grande riqueza de feições e complexidade estrutural.

Filtrar ruído puramente aleatório (sem correlação espacial) é uma tarefa relativamente trivial, normalmente bastando qualquer filtro passa-baixa. Porém, separar feições com estruturas espaciais diferentes requer um método mais sofisticado. Assim, decompor uma imagem em seus elementos estruturais é de crescente interesse na atividade de geomodelagem. Essa decomposição pode encontrar aplicação, por exemplo, na filtração de ruídos não puramente aleatórios que apresentam estrutura espacial (ex. marcas de aquisição sísmica – Chopra e Marfurt, 2007, pp. 158-166) e separação das feições geológicas de diferentes escalas, formas e anisotropias tal como na decomposição de uma imagem de um fundo marinho em nove fatores feita por Carvalho *et al.* (2018). A Figura 1-1 ilustra um caso representativo desse problema, em que o método de aquisição normalmente registra não somente a informação como também introduz artefatos (estruturados ou não) que podem acontecer desde o estímulo do sensor até o processamento computacional que resulta em um arquivo digital utilizável.

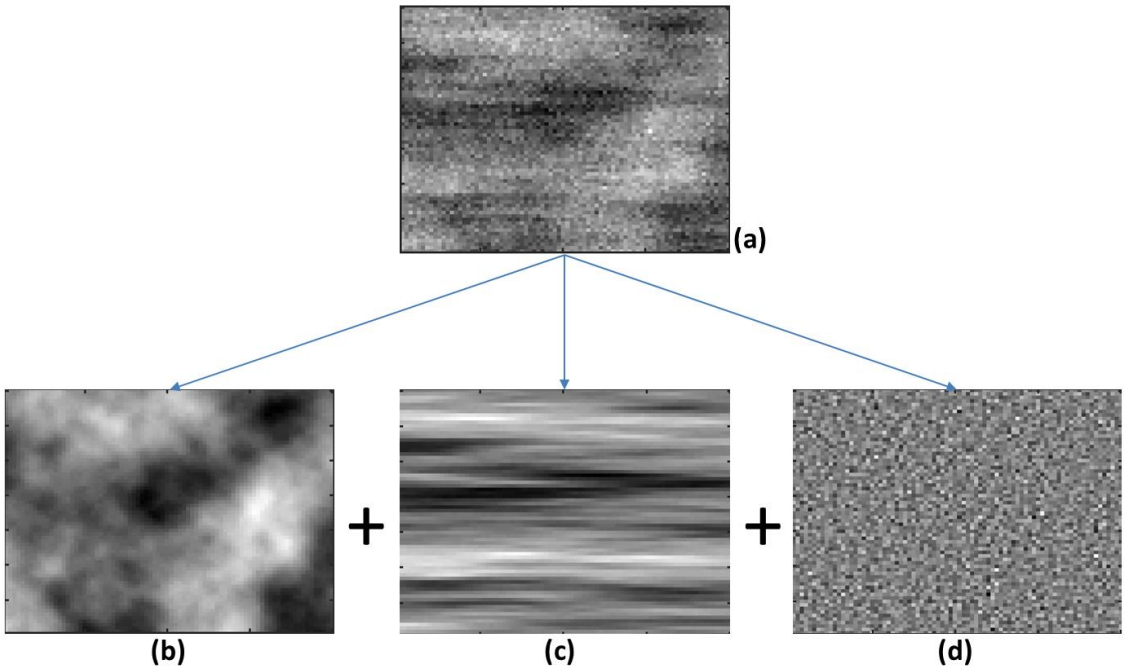


Figura 1-1: Uma imagem (a) decomposta em três fatores: informação real (b); artefato estruturado (c) e artefato puramente aleatório (d). Modificado de Shamsipour *et al.* (2017, fig. 5).

Assim, uma imagem M pode ser expressa como um somatório cujos termos são imagens suportadas pela mesma malha regular. Para os objetivos desta tese, esses m termos devem conter, cada qual, feições de mesma forma geológica e, idealmente, formas diferentes entre si. Doravante, esses termos serão referidos por fatores geomorfológicos g_k com $k \in [1, m] \subset \mathbb{N}$ e o número m é normalmente informado pelo modelador a depender do objetivo do estudo. Portanto, uma imagem pode ser expressa como na Equação 1-1. Esta ideia está ilustrada pela Figura 1-2. Nesta tese, o operador de adição entre imagens se comporta como o operador de adição entre matrizes.

$$M = g_1 + g_2 + \cdots + g_m$$

Equação 1-1: Uma imagem M expressa como uma soma de m fatores geomorfológicos g.

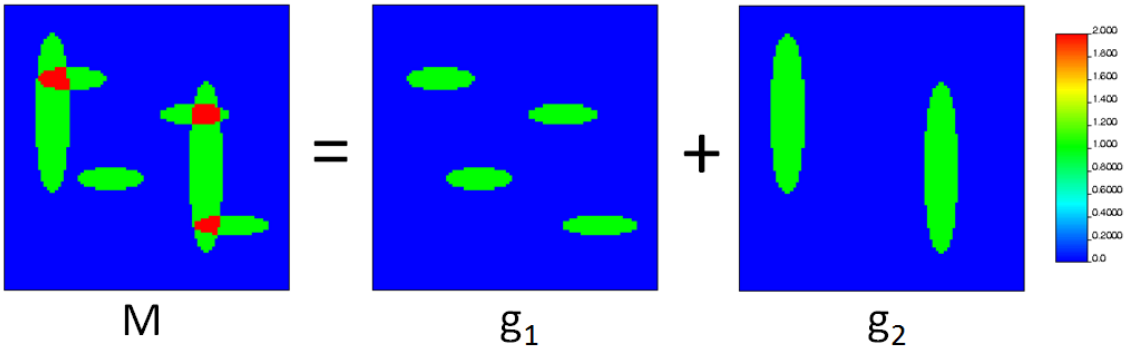


Figura 1-2: Uma imagem M contendo geologia complexa decomposta como a soma de duas imagens contendo corpos geológicos de mesma geometria (g_1 e g_2).

O número de fatores geomorfológicos, *m*, é esperado que seja pequeno (da ordem de 10). Por exemplo, decompor a imagem em: fator com feições de pequena escala; fator com feições de grande escala e um fator contendo artefatos ou ruído.

1.1 Estado da arte em análise estrutural de imagens

Análise: "ana" significa "para cima" e "lise" significa "separar". Refere-se à técnica ancestral de atirar o trigo para cima a fim de que o vento separe a palha do grão. Trata-se de um vocábulo de origem grega que significa decompor algo em suas partes fundamentais para exame.

Análise estrutural significa, portanto, neste contexto, descobrir uma composição de formas simples que resulta nas formas intrincadas que a natureza apresenta. No campo do processamento de imagens, é conhecida por análise, extração ou discriminação de texturas. Existem muitos métodos para obter uma decomposição estrutural, sendo os principais apresentados nas subseções que se seguem.

1.1.1 Análise com krigagem fatorial

Embora originalmente concebida como um método de interpolação (estimativa) de amostras espalhadas, a krigagem fatorial (FK – Seção 2.2.1) pode ser aplicada a imagens prontas para obter, para cada estrutura aninhada de um modelo variográfico, uma imagem contendo feições de determinadas escala e orientação de acordo com a respectiva estrutura variográfica aninhada. Para obter uma decomposição, o modelador infere ou ajusta um variograma teórico ao variograma experimental da imagem. O variograma (Matheron, 1963) é uma função que expressa a semivariância entre dois dados em função de sua separação espacial (*h*). Assim, a imagem original é decomposta em fatores de mesma correlação espacial. A Figura 1-3 ilustra um exemplo em que um variograma, contendo uma estrutura de curto

alcance e abrupta e outra de longo alcance e suave, é utilizado com FK para separar as tendências (*trends*) de grande escala das variações locais de curta escala.

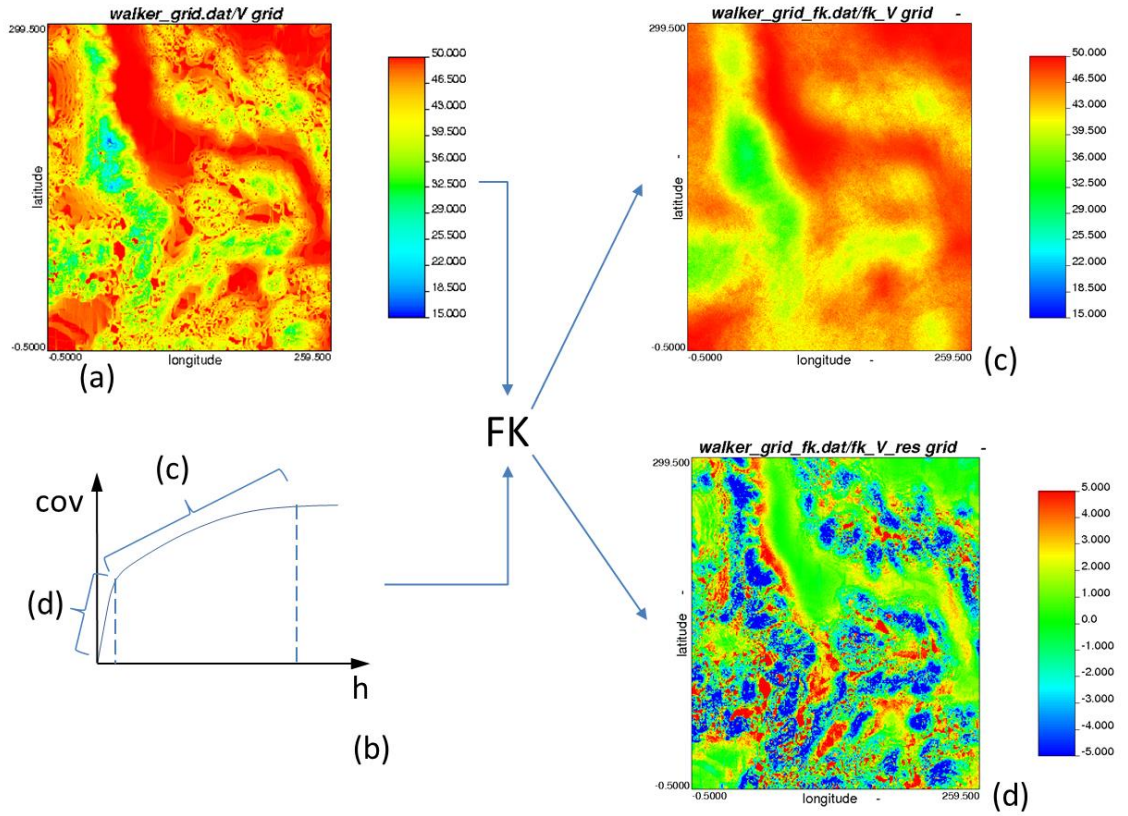


Figura 1-3: Emprego de FK na separação das feições suaves de larga escala (c) das abruptas de pequena escala (d). As escalas e os perfis das feições a serem separadas são modeladas como estruturas no variograma (b). (a): mapa de Walker Lake (Isaaks e Srivastava, 1989).

1.1.2 Análise de Fourier

A Transformada de Fourier (Seção 2.2.2) é bem conhecida e empregada no campo da análise de sinais (séries temporais). A sua aplicação para análise estrutural é relativamente recente devido à tecnologia computacional atual para realizá-la em grandes imagens 2D e 3D e às suas implementações eficientes em software (algoritmos de Transformada Rápida de Fourier e sua inversa – FFT e RFFT – dentre os quais o Cooley-Tukey FFT (Cooley e Tukey, 1965) é o mais popular).

As feições estruturais em uma imagem podem ser pensadas como tendo uma frequência espacial, ou seja, uma feição de pequena escala tem uma frequência espacial alta. Assim, editando o espectro de amplitudes nas frequências espaciais de uma imagem, é possível separar as formas fundamentais que a compõem, tal como mostrado na Figura 1-4.

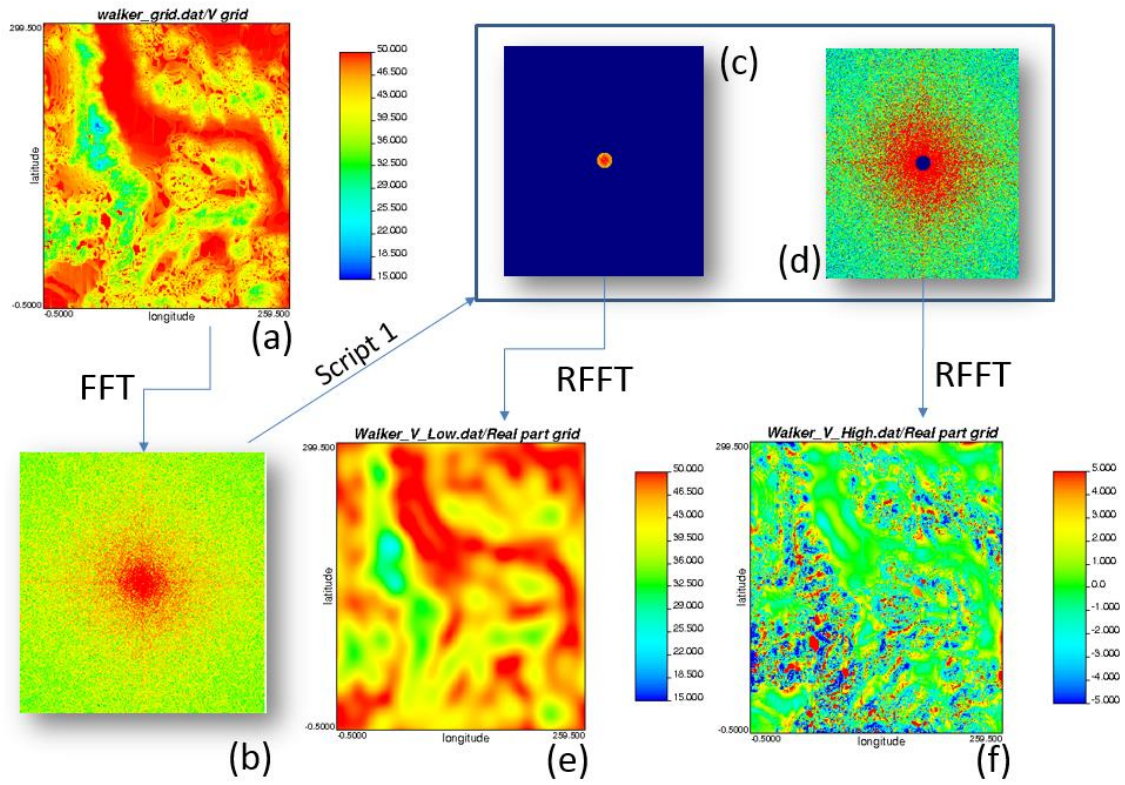


Figura 1-4: Obtenção de tendências e de resíduos com análise de Fourier. (a) é o mapa original do famoso dado de Walker Lake (Isaaks e Srivastava, 1989); (b) é seu mapa de amplitudes (baixas frequências no centro); (c) e (d) são os espectros de amplitudes separados com o Script 1 (Anexo 1); (e) e (f) são os mapas obtidos ao retrotransformar (c) e (d) respectivamente. O mapa de fases (que não é alterado) também é resultante da FFT, é a outra entrada da RFFT e está omitido por simplicidade.

1.1.3 Análise com valores singulares

Imagens de duas dimensões (por exemplo, mapas) podem ser vistas como matrizes, portanto podem ser submetidas a operações sobre matrizes, particularmente SVD (Decomposição em Valores Singulares – Singular Value Decomposition – Seção 2.2.3). Uma das muitas aplicações de SVD é a decomposição de uma matriz (imagem 2D) em uma soma de matrizes.

Embora SVD não seja uma análise estrutural e sim uma análise espectral (matrizes), existe na natureza uma tendência de a maior quantidade de energia (informação) estar concentrada nas maiores escalas. Assim, como demonstrado pelo exemplo da Figura 1-5, existe uma tendência de encontrarmos estruturas de grande escala distribuídas pelos primeiros fatores e as feições de menor escala estarem presentes nos seguintes. Isto porque, a SVD tem a característica interessante de ordenar os valores singulares em ordem decrescente ($\sigma_1 > \sigma_2 > \ldots > \sigma_n$). Consequentemente, os fatores ficam em ordem decrescente por quantidade de informação.

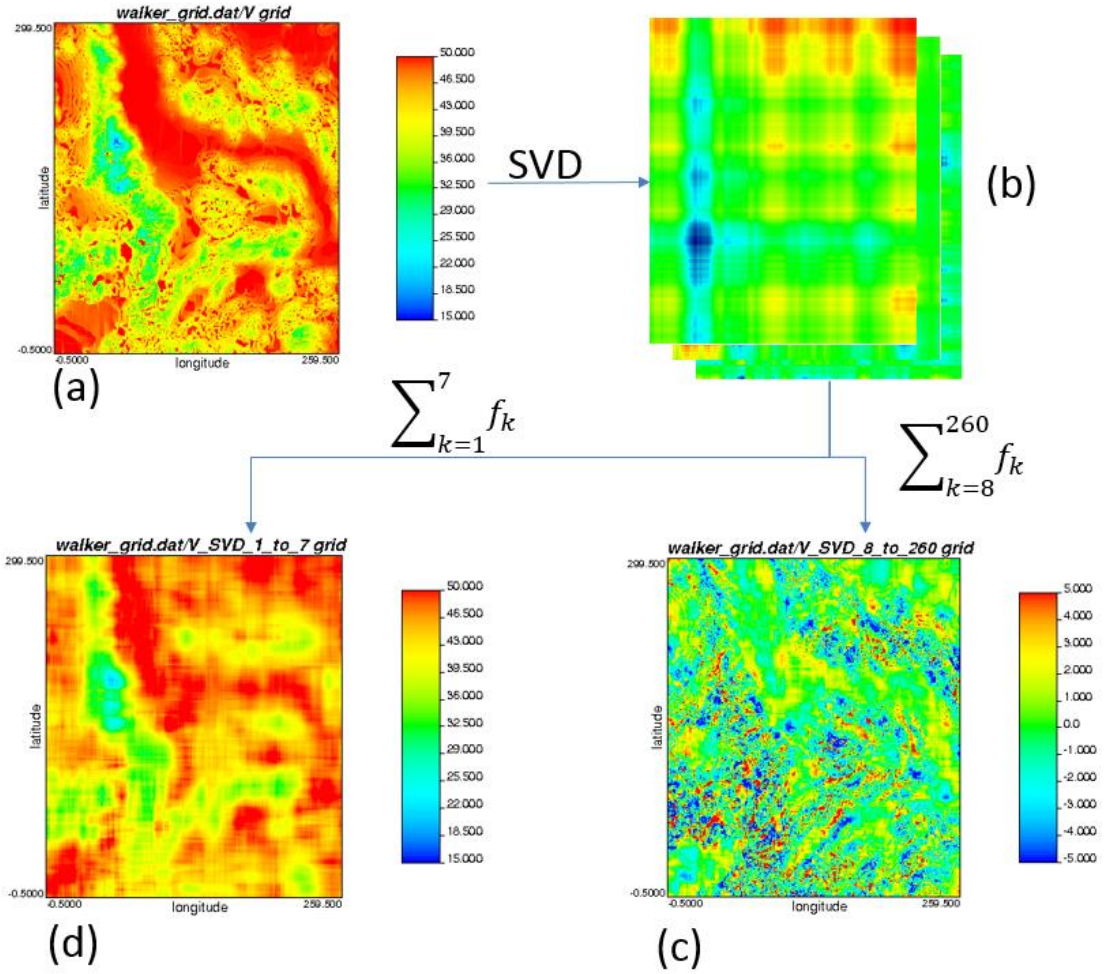


Figura 1-5: Obtenção de tendências e de resíduos com análise de valores singulares. (a) é o mapa original do famoso dado de Walker Lake (Isaaks e Srivastava, 1989); (b) representam as 260 imagens fundamentais f_k obtidas (o mapa tem 260 colunas); (c) foi obtido ao somar as imagens fundamentais da 8ª à 260ª; (d) foi obtido ao somar as imagens fundamentais da 1ª à 7ª.

1.1.4 Análise com Empirical Mode Decomposition

Este método de análise estrutural (EMD – Seção 2.2.4) resulta em uma série de imagens chamadas de *intrinsic mode functions* (IMFs). As IMFs são imagens com cada vez mais componentes de frequência espacial (da maior para a menor) removidas. É um método empírico, isto é, não requer um modelo e também não envolve transformações no dado, portanto é muito fácil de utilizar. A Figura 1-6 ilustra uma aplicação, onde as imagens (e), (f), (g) e (h) são IMFs com feições removidas em número crescente de bandas de frequência espacial filtradas.

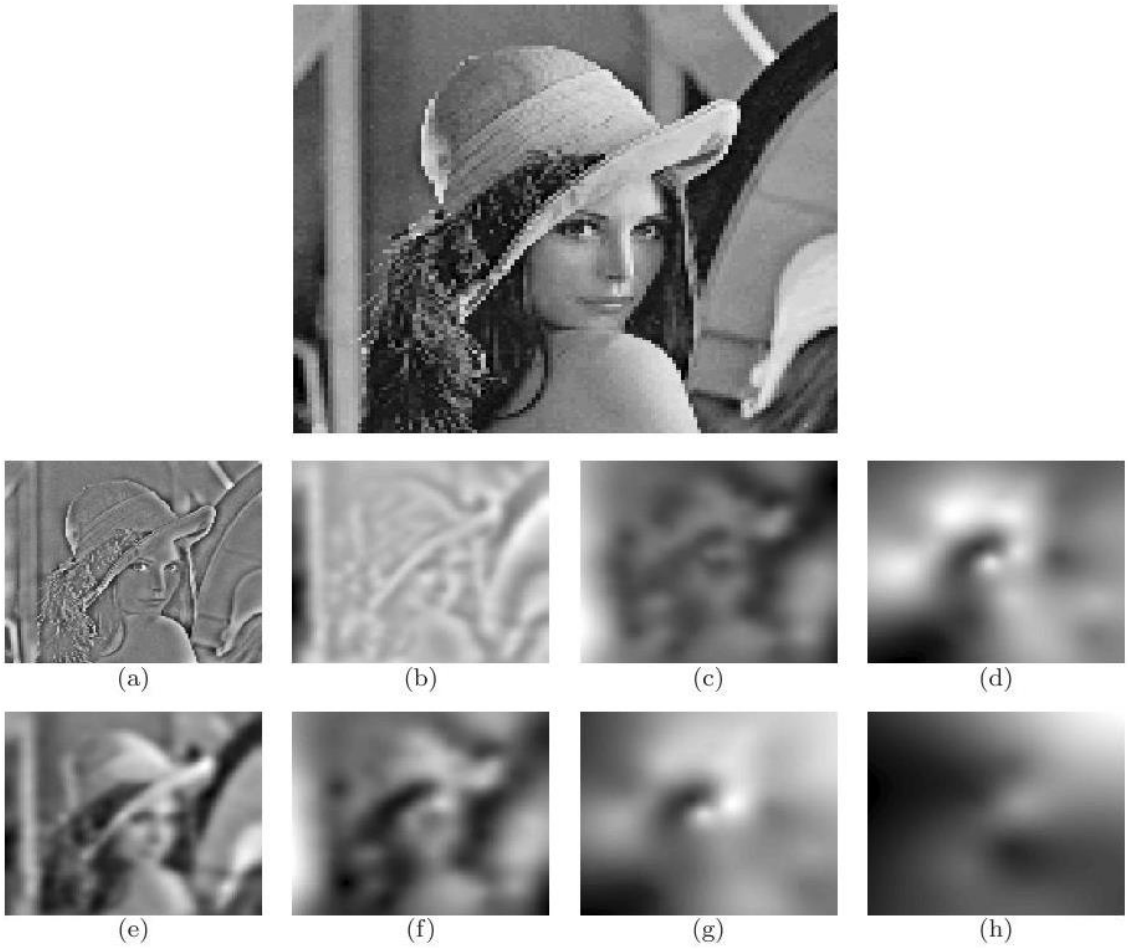


Figura 1-6: Exemplo de aplicação da EMD (retirado de Linderhed, 2009, figs. 1 e 2). (a)-(d): médias dos envelopes de máximos e mínimos locais (ver Seção 2.2.4, item iii); (e)-(h): IMFs obtidas com o método. A imagem original (alto) é usada como IMF de entrada para a primeira iteração.

1.1.5 Análise de Gabor

A análise com filtro de Gabor é a convolução de uma imagem contra um núcleo (*kernel*) com propriedades especiais, o chamado núcleo de Gabor, destinado a separar estruturas orientadas com um azimute e de uma frequência específicos. A Figura 1-7 ilustra um exemplo do emprego do filtro em uma imagem 2D para obtenção de a resposta de um núcleo com azimute e frequência fixos, que resulta no descarte das feições fisionômicas para o reconhecimento da expressão facial. A resposta é obtida ao convolver o núcleo contra a malha regular do mapa, produzindo a resposta do filtro.

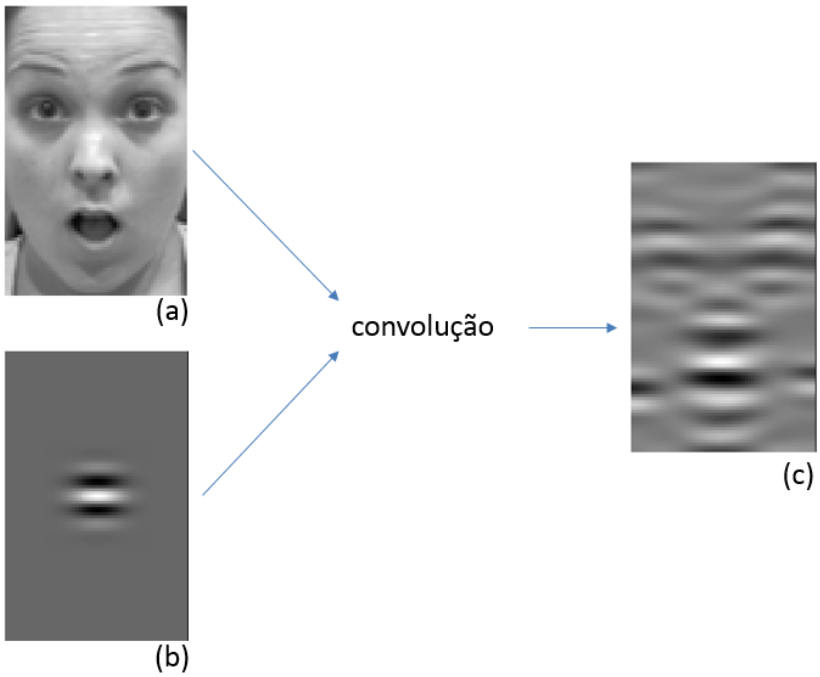


Figura 1-7: Exemplo de uma resposta do filtro de Gabor a uma imagem. (a): imagem original; (b): núcleo de convolução Gabor; (c): resposta do filtro (a uma frequência e a um azimute). (a), (b) e (c) retirados de [1].

Fixando o azimute e variando a frequência, pode-se empilhar as diversas respostas para formar um espaço de Gabor correspondente a um determinado azimute. A Figura 1-8 mostra um exemplo de espaço de Gabor para um dado bidimensional e um determinado azimute. O eixo vertical é o da frequência. As amplitudes das respostas do filtro são mostradas como voxels no volume. Dessa forma, o espaço de Gabor pode ser interpretado como uma visão explodida do conteúdo espectral local variável de um mapa. Selecionando-se e somando-se determinadas respostas no espaço de Gabor, pode-se obter as estruturas correspondentes a determinadas escalas e orientações.

[1] http://ttsuchi.github.io/2015/08/26/gaborfilters.html, acessado em 24/07/2018.

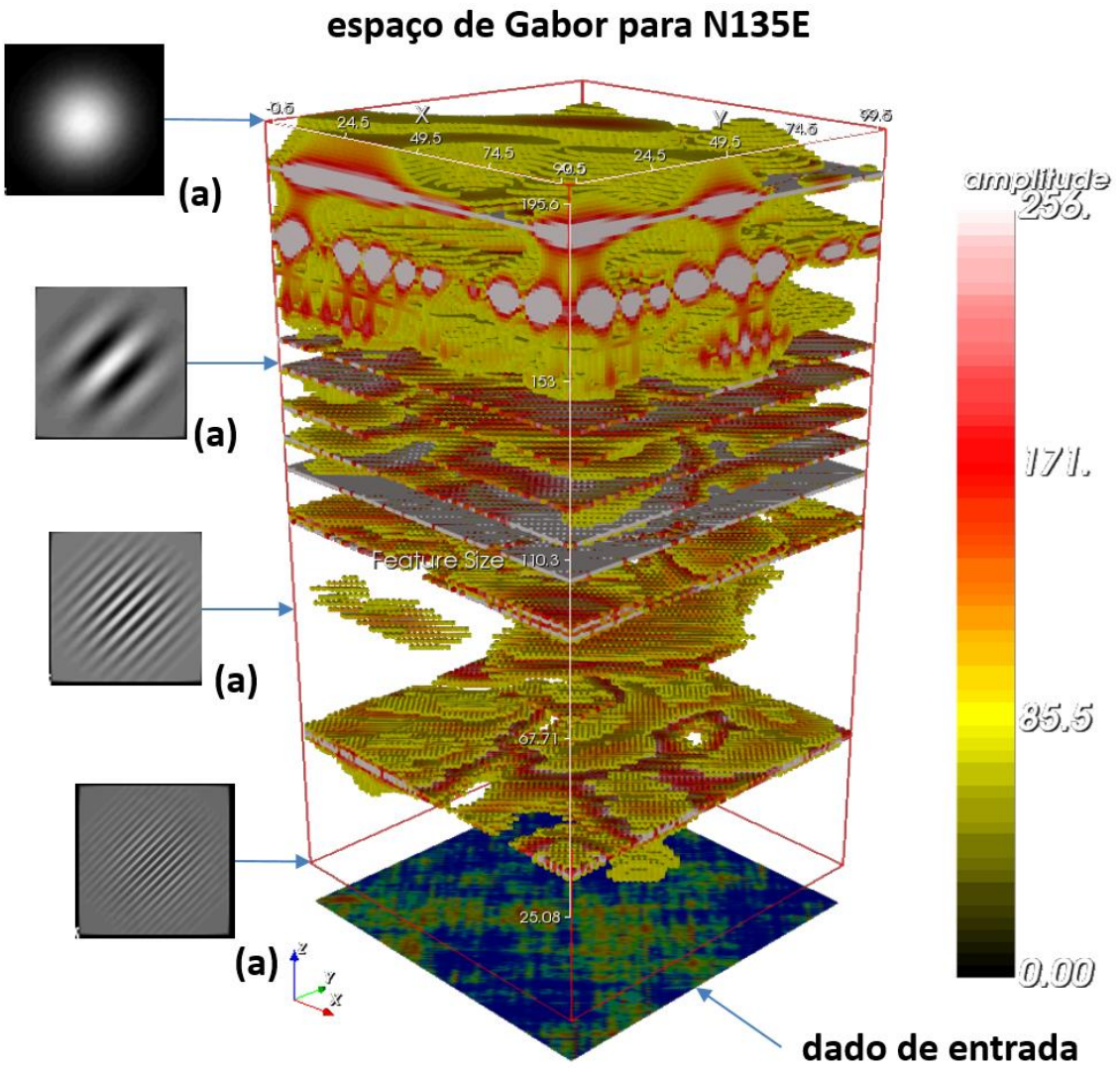


Figura 1-8: O espaço de Gabor como uma visão explodida do conteúdo espectral local variável de um dado 2D. (a): são alguns núcleos de Gabor correspondentes a algumas frequências desse espaço.

1.1.6 Análise com ondaleta (*wavelet*)

A análise estrutural com ondaleta[2] (Seção 2.2.6) também emprega um núcleo de convolução especial, chamado de ondaleta, para "sintonizar" estruturas de determinadas frequências e orientações tal como na análise com Gabor. A diferença reside na forma com que o dado de entrada é "sintonizado".

Na análise de Gabor, é necessário variar continuamente a forma do núcleo de convolução enquanto que na análise com ondaleta, o núcleo é uma forma fixa especificada pelo

[2] Termo lusófono empregado por Morettin (1999), embora os geofísicos usem frequentemente o termo "ondícula", que é um vocábulo do idioma espanhol.

modelador, tal como um variograma na FK. O algoritmo a reescala automaticamente para mapear as correlações com as estruturas em diversas escalas, gerando um modelo chamado escalagrama. Então, a separação de estruturas é obtida por uma operação chamada *thresholding* (Seção 3.6.4). *Thresholding* pode ser realizado com instruções do tipo `if...else` em um *script* de calculadora que é mais simples de executar do que a seleção de respostas do núcleo no espaço de Gabor. A Figura 1-9 ilustra a aplicação em uma imagem 2D.

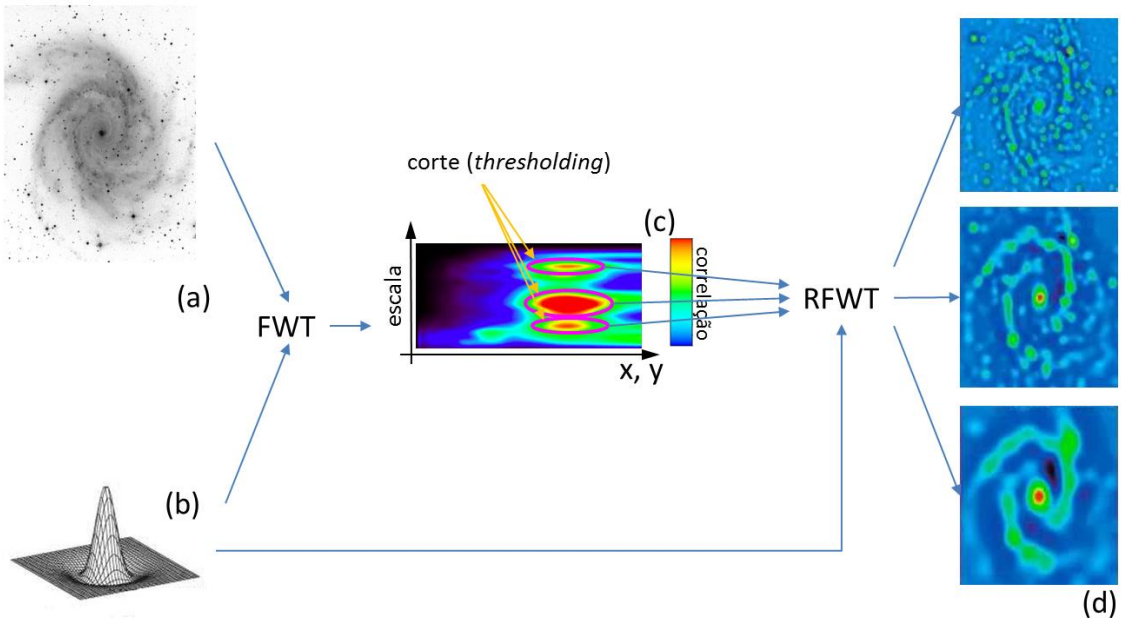


Figura 1-9: Exemplo de aplicação da análise com ondaleta para decompor uma imagem. (a): imagem da galáxia NGC 2997 (modificado de [3]); (b): ondaleta; (c): escalagrama obtido; (d): três imagens obtidas quando se retrotransforma três partes destacadas do escalagrama com a ondaleta. (b) e (d) retirados de [4].

Como em diversas análises vistas nas seções anteriores, há também aqui a limitação imposta pela parametrização global (ondaleta), o que faz o método não capturar variações locais. Uma mesma ondaleta global pode se correlacionar de forma imperfeita conforme a forma das feições muda em um estudo que tenha uma grande extensão geográfica.

1.1.7 Análise com redes convolucionais

Não se sabe de uma aplicação de redes convolucionais com resultado equivalente ao da krigagem fatorial com sucesso. Não obstante, esta tese apresenta diversas arquiteturas de

[3] http://kudzu.astr.ua.edu/devatlas/Sc-gals.html, acessado em 24/07/2018.

[4] https://www.astronomyclub.xyz/wavelet-transform/ji.html, acessado em 24/07/2018.

rede e um novo tipo de camada convolucional com potencial para aplicação ao problema desta tese. As redes convolucionais são apresentadas na Seção 2.5.4.

1.2 Motivação

O problema consiste em decompor uma imagem (por exemplo, um mapa de um fundo marinho ou um horizonte sísmico) como um somatório de fatores geomorfológicos, porém sem parametrização global *a priori* (variograma, ondaleta, etc.). Portanto, a decomposição em fatores geomorfológicos deve ser automatizada.

A ideia é obter um método que seja adaptativo como o Empirical Mode Decomposition, porém sem suas limitações, para que possa capturar características locais. Contudo, o método deve resultar em fatores geomorfológicos cuja estrutura espacial tenha um modelo teórico. Foi escolhido o variograma como modelo, assim, de forma resumida, esta tese visa propor um método cujo resultado sejam os mesmos fatores que seriam obtidos com krigagem fatorial, porém sem a necessidade de modelar um variograma, uma tarefa assaz laboriosa quando se espera muitas estruturas imbricadas.

1.3 Meta

A meta desta tese consiste em propor um novo método de decomposição automatizada em fatores geomorfológicos de dados regularmente amostrados (em malhas regulares) em domínio espacial com menor custo laboral e, idealmente, resultados equivalentes ao do emprego de FK.

1.4 Objetivos

Para alcançar a meta do estudo, os seguintes objetivos devem ser cumpridos:

- Avaliar a decomposição estrutural a partir de diversos algoritmos de decomposição estrutural com parâmetros globais definidos pelo modelador.
- Propor e avaliar uma metodologia para decomposição automatizada da imagem em fatores geomorfológicos, cada qual correspondendo a uma estrutura variográfica.
- Comparar os resultados obtidos através de modelos manuais globais com os resultados obtidos com a metodologia proposta.

1.5 Método

i. Software:

- GSLib (Deutsch e Journel, 1998), conjunto de programas projetados para realização de toda sorte de tarefas em geoestatística, além de utilitários para análise exploratória de dados, pós-processamento e saídas gráficas;
- GammaRay[5], camada software sobre a GSLib[6] que contém utilitários interativos de geoestatística e para aplicação dos algoritmos FFT, SVD, etc. e o Jupyter Notebook[7] necessários (ver Seção 2.2) à execução dos experimentos;
- Anaconda 3[8], pacote de softwares que contém, dentre outros itens, um interpretador Python e o Jupyter Notebook, que é uma interface que executa em um navegador de internet e torna mais amigável a interação com o Python. Os Notebooks requerem que sejam instalados no Python (comandos `conda` do Anaconda Prompt ou `pip install` do interpretador Python) os pacotes NumPy, MatPlotLib, SciKit-Learn, Keras 2 e Numba;
- Os Jupyter Notebooks[9] para execução dos experimentos do Capítulo 4.

ii. Dados:

- Um mapa (malha 2D) com origem em (0, 0) de 100 × 100 células, cada célula de 1m × 1m contendo um registro sintético de uma geologia qualquer contaminado com artefatos espacialmente correlacionados. A malha foi gerada a partir de uma Simulação Sequencial Gaussiana (SGSIM – Deutsch e Journel, 1998, seção V.2.3), incondicional e com o variograma da Tabela 1-1. O efeito pepita representa o ruído puramente aleatório, assumido como previamente removido. As estruturas 2 e 3 representam estruturas geológicas de curto e longo alcances e as estruturas 4 e 5 representam marcas de aquisição sísmica e curto e longo alcances (artefatos com estrutura espacial). Também, não há uma tendência (*trend*) global como uma rampa no sentido leste-oeste.

---

[5] Disponível (executáveis Windows e códigos-fonte para compilar em outras plataformas) para download em https://github.com/PauloCarvalhoRJ/gammaray.

[6] Pode-se baixar executáveis e códigos-fonte da GSLib a partir de http://www.statios.com/Quick/gslib.html.

[7] https://github.com/PauloCarvalhoRJ/pynb_paper_autovarfit2D

[8] Pode-se obter o Anaconda 3 a partir de https://www.anaconda.com/distribution.

[9] Os Python Notebooks podem ser obtidos de https://github.com/PauloCarvalhoRJ/notebooks_tese.

Estrutura	tipo	contribuição	azimute	semieixo maior	semieixo menor
1	efeito pepita	0%	N/A	N/A	N/A
2	esférico	25%	N000E	20m	20m
3	esférico	25%	N000E	50m	50m
4	esférico	25%	N135E	10km	1m
5	esférico	25%	N045E	10km	5m

Tabela 1-1: Parâmetros variográficos usados para gerar o dado para os experimentos.

O dado (arquivo-texto no formato GEO-EAS/GSLib) pode ser baixado de https://github.com/PauloCarvalhoRJ/gammaray/blob/master/data/uncond_SGSIM.grid e está mostrado na Figura 1-10.

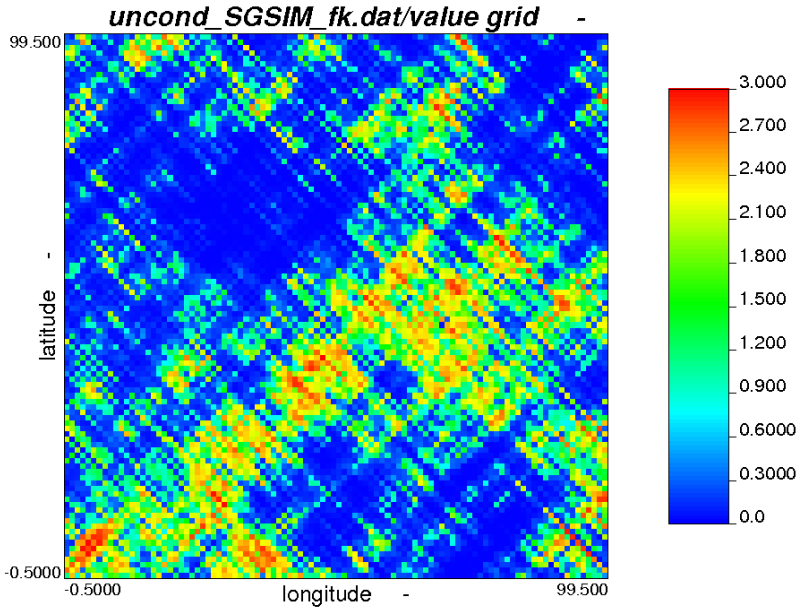


Figura 1-10: Mapa dos dados para os experimentos.

iii. Procedimento:

A parte experimental deste estudo é composta pela decomposição estrutural manual do mapa de uma geologia sintética contaminado com artefatos espacialmente correlacionados através de seis algoritmos: Krigagem Fatorial (FK), Transformada Rápida de Fourier (FFT), Decomposição em Valores Singulares (SVD), Empirical Mode Decomposition (EMD),

Filtro de Gabor, Transformada Rápida de Ondaleta (FWT). A parte experimental também inclui a decomposição estrutural automatizada com técnicas de aprendizado de máquina e com o método automatizado proposto do mesmo mapa. Avaliar os variogramas de cada um dos quatro fatores obtidos com o algoritmo proposto quanto à semelhança com cada estrutura variográfica do dado original (Tabela 1-1). Ainda com o algoritmo proposto, esse procedimento deve ser repetido com cada um dos métodos de otimização implementados (Seção 2.1) e com as duas funções-objetivo (Seções 5.3 e 5.4.1).

iv. Resultados:

Quatro fatores geomorfológicos cujos variogramas, idealmente, correspondem a cada uma das quatro estruturas imbricadas do modelo variográfico usado para gerar o dado original.

1.6 Organização da tese

- Capítulo 2: Apresenta as teorias, conceitos, métodos e algoritmos utilizados no processo de decomposição estrutural de mapas. A leitura deste capítulo permitirá compreender como cada passo dos três modos (manual, automatizado com aprendizado de máquina, automatizado com algoritmo dedicado) de decomposição funciona.

- Capítulo 3: Apresenta a decomposição manual do mapa de teste com cada um dos principais métodos de decomposição estrutural apresentados na Seção 1.1, excetuando-se o uso de redes convolucionais, que é um processo automatizado e está apresentado no Capítulo 4.

- Capítulo 4: Apresenta a tentativa de decomposição automatizada com técnicas de aprendizado de máquina.

- Capítulo 5: Apresenta o método proposto de decomposição automatizada com algoritmo escrito especificamente para esse fim.

- Capítulo 6: Os resultados das decomposições com o método proposto nesta tese são apresentados e discutidos. Os problemas e limitações encontrados durante o emprego do método proposto são discutidos.

- Capítulo 7: São apresentados conclusões e trabalhos futuros para resolução das questões surgidas durante a execução do método proposto.

Capítulo 2

Revisão Bibliográfica

Esta revisão bibliográfica consiste em visitar as teorias, conceitos, cálculos e algoritmos empregados nos experimentos e na construção do fluxo de decomposição estrutural que esta tese propõe.

2.1 Métodos de otimização

Esta seção apresenta os métodos de otimização considerados e implementados para encontrar mínimos globais das funções-objetivo apresentadas nas Seções 5.3 e 5.4.1 para os experimentos.

2.1.1 Simulated Annealing com Gradiente Descendente em tandem (SA+GD)

O método do Gradiente Descendente, proposto por Cauchy (1847) e dado pela Equação 2-1, consiste em encontrar um mínimo de uma função-objetivo *F*. O escalar α é um fator de redução (ou tamanho do passo) para o vetor ∇F a fim de prevenir saltos para além do mínimo, o que pode retardar a convergência. Yuan (1999) discute e propõe métodos para calcular o melhor α para vários tipos de funções-objetivo, no entanto, como neste caso *F* decorre de um processo numérico (software), o passo α também é determinado iterativamente. Assim, para cada passo *i*, o algoritmo implementado para esta tese inicia α com um valor padrão (por exemplo 1,0) e o reduz gradativamente até detectar uma descida ($F([w])_{i+1} < F([w])_i$). Isso assume que o vetor ∇F aponta na direção geral de um mínimo local. As iterações cessam quando um critério de convergência é satisfeito (por exemplo $F([w]_i) / F([w]_{i+1}) < 1 + \varepsilon$) e/ou quando certo número de passos é executado.

$$[p]_{i+1} = [p]_i - \alpha \nabla F([p]_i)$$

Equação 2-1: Atualização iterativa dos parâmetros *p* de uma função-objetivo *F* com o método do gradiente descendente. *i* é o número da iteração; α: ver texto.

No entanto, nesta tese, encontrar o gradiente ∇F não é trivial, pois a expressão analítica de *F* em função dos parâmetros *p* é desconhecida porque *F* resulta de um processamento numérico (software), tal como descrito nas Seções 5.3 e 5.4.1. Portanto, ∇F deve ser

computado numericamente. Assumindo métrica Euclidiana para o subespaço vetorial definido pelos parâmetros livres p e que esses parâmetros são coordenadas Cartesianas nesse espaço, então o gradiente pode ser calculado pela Equação 2-2 onde $\hat{\imath}_k$ são versores[10] do sistema de coordenadas Cartesiano de dimensão mn.

$$\nabla F = \sum_{k=1}^{mn} \frac{\partial F}{\partial p_k} \hat{\imath}_k$$

Equação 2-2: Cálculo do gradiente da função-objetivo F.

As derivadas parciais podem ser calculadas numericamente pelo método amplamente conhecido (método de Euler – Equação 2-3) onde ε é um valor muito pequeno (ex. 6 ou 7 ordens de magnitude abaixo dos valores típicos de F) dado pelo modelador como um parâmetro do algoritmo.

$$\frac{\partial F}{\partial p_k} \sim \frac{F(p_1, \ldots, p_k + \varepsilon, \ldots, p_{mn}) - F(p_1, \ldots, p_k - \varepsilon, \ldots, p_{mn})}{2\varepsilon}$$

Equação 2-3: Derivada parcial em relação ao parâmetro p_k da função-objetivo F calculada numericamente.

Entretanto, dependendo de F, pode haver muitos mínimos (locais e globais). Portanto, o resultado do GD pode depender sensivelmente de onde a busca começa no espaço das variáveis independentes. Assim, para evitar que o processo de otimização fique preso em um extremo local, faz-se necessário utilizar um método que "aceite subir o morro de vez em quando", isto é, um método estocástico. Um método popular para isso é o Simulated Annealing. *Annealing* é uma palavra inglesa que significa "temperar", no contexto da ciência dos materiais. O processo da têmpera consiste em fundir um material e resfriá-lo, de forma controlada, tal que um arranjo molecular específico final, diferente do inicial, seja atingido e assim alguma propriedade mecânica do material seja alterada para um valor-alvo, por exemplo, rigidez de determinado grau.

Resumidamente, SA (Metropolis *et al.*, 1953; Cerny, 1985; Deutsch, 1992; Press *et al.* 2007, seção 10.12) é um método de otimização estocástico (caminho aleatório) que funciona como outros métodos para minimização, porém admite "subir o morro de vez em quando para ver se do outro lado há um vale mais profundo". Em termos formais, SA associa uma probabilidade de aceitação a um conjunto de parâmetros p (ver Equação 2-2) que leve a um

[10] Vetor de tamanho unitário (1,0).

estado mais energético (maior valor da função-objetivo *F*). Essa probabilidade, tão maior quanto seja a "temperatura", é usada em um sorteio para decidir se SA o aceita e diminui conforme as iterações passam (esfriamento). A busca prossegue até que a temperatura mínima seja atingida ou que um certo número de passos seja atingido. O termo *simulated quenching* é usado para quando a temperatura mínima é zero, isto é, a temperatura em que somente estados que levem a menores energias são aceitos.

Assim, SA é usado para inicializar os parâmetros livres *p* próximos do mínimo global (idealmente, pois isso não é garantido) da função-objetivo *F*. SA não faz por si só uma assunção acerca da função-objetivo, mas, quando ela é utilizada em tandem com outro método de otimização, deve-se considerar a assunção deste: GD, como utiliza gradientes, é um método que assume que a função-objetivo é diferençável.

No Anexo 4, i) está o algoritmo SA em sintaxe *quasi*-C++ tal como implementado no software empregado nos experimentos. Depois que o SA encontra parâmetros perto de um mínimo global (teoricamente), o algoritmo de GD prossegue para encontrar o mínimo exatamente, em sintaxe semelhante no Anexo 4, ii). A Figura 2-1 ilustra a forma com que SA e GD funcionam para encontrar o mínimo global de uma função-objetivo.

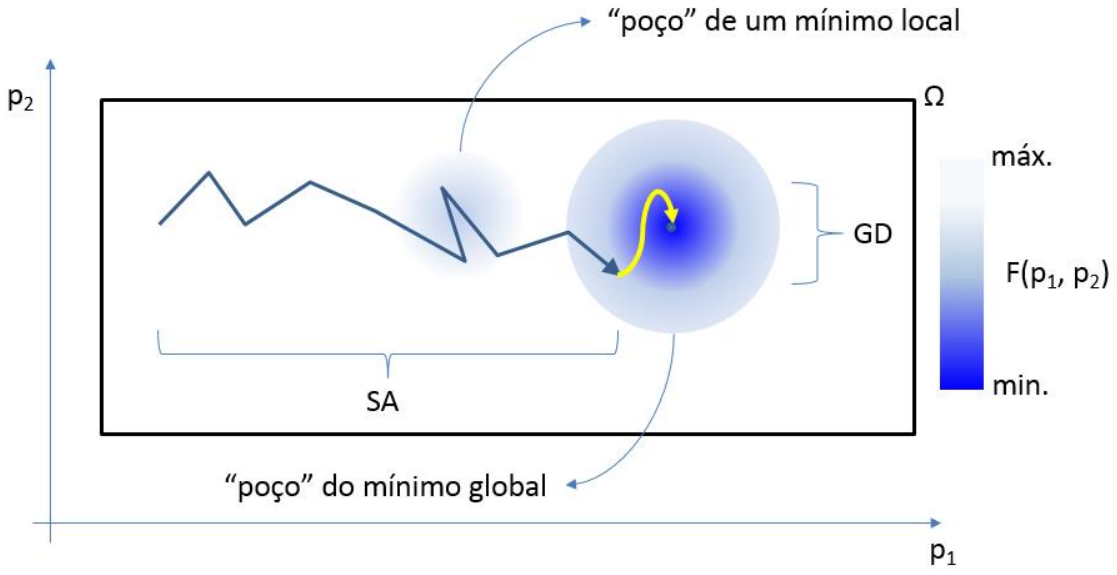


Figura 2-1: Ilustração de como SA com GD em tandem encontram o mínimo global de uma função-objetivo *F* de apenas dois parâmetros (p_1 e p_2). Ω: domínio de busca.

2.1.2 Busca Linear com Reinício (LSRS)

LSRS (Linear Search with Re-Start) é um algoritmo de otimização proposto por Grosan e Abraham (2009) para encontrar um ótimo global rapidamente (componente estocástica) para

uma função-objetivo duplamente diferençável com grande número de dimensões dentro de um domínio limitado.

Um conjunto de pontos, cada qual denotado por vx_i onde *i* é o índice da variável, é iniciado aleatoriamente dentro do domínio. Cada ponto é deslocado para uma nova posição (aleatória ou seguindo um passo de tamanho constante) ao longo de uma linha de direção e sentido ortogonais aos eixos do espaço de parâmetros. A ideia é que a busca ao longo de diversas linhas tem maior probabilidade de encontrar um mínimo do que com apenas pontos. O ponto que resultou na melhor avaliação da função-objetivo é registrado (denotado por $xOtimo_i$).

O gradiente da função-objetivo é então avaliado no entorno de $xOtimo_i$ e os limites do domínio são atualizados de acordo com os valores do gradiente. O algoritmo então reinicia (daí o nome), gerando um novo conjunto de pontos dentro do novo domínio. No Anexo 4, iii) está o pseudocódigo do algoritmo e a Figura 2-2 ilustra seu funcionamento.

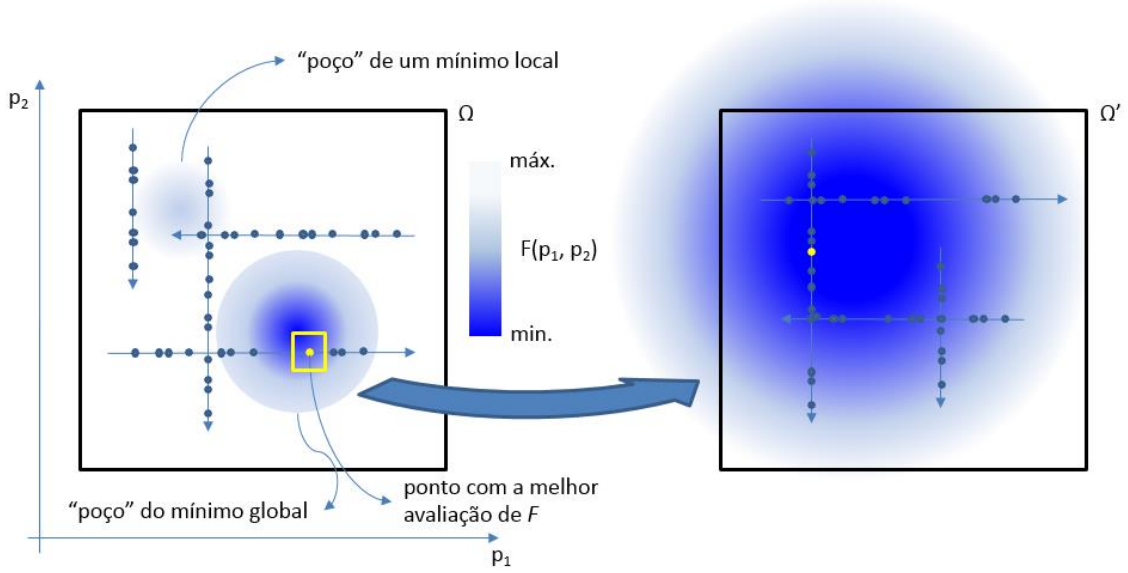


Figura 2-2: Ilustração do funcionamento do algoritmo LSRS para encontrar o mínimo global de uma função-objetivo *F* de apenas dois parâmetros (p_1 e p_2). Ω: domínio de busca; Ω': domínio de busca reduzido após um reinício.

2.1.3 Enxame de Partículas (PSO)

PSO (Particle Swarm Optimization – Kennedy e Ebberheart, 1995) é um método de otimização com uma componente estocástica para encontrar rapidamente um ponto ótimo de uma função-objetivo multidimensional. Embora o método em si não dependa de gradientes da função-objetivo, seu funcionamento é análogo ao do movimento de partículas

estudado pela Mecânica, o que implica em continuidade. O método consiste em inicialmente posicionar um certo número de "partículas" aleatoriamente no domínio. Cada "partícula" tem uma "posição" (os valores dos parâmetros livres – denotados por x_i, onde i é o índice do parâmetro livre) e uma "velocidade" (taxas de variação dos parâmetros – denotadas por v_i). O número de partículas é um parâmetro fornecido pelo usuário.

Cada partícula também tem associada a ela a posição onde obteve a melhor avaliação da função-objetivo ao longo do seu trajeto, denotada por $xMin_i$. O algoritmo também mantém a posição da melhor avaliação global, denotada por $gxMin_i$. As i-ésimas componentes da posição (valor do i-ésimo parâmetro livre) e da velocidade de uma partícula do t-ésimo passo são atualizadas no (t+1)-ésimo passo com a Equação 2-4.

$$v_i(t+1) = \omega v_i(t) + c_1 random_1\big(xMin_i - x_i(t)\big) + c_2 random_2\big(gxMin_i - x_i(t)\big)$$
$$x_i(t+1) = x_i(t) + v_i(t+1)$$

Equação 2-4: Atualização da posição e velocidade das partículas no método de otimização Enxame de Partículas (PSO). ω é um coeficiente "inercial" ou "de massa"; c_1 e c_2 são coeficientes de "aceleração"; $random_1$ e $random_2$ são números aleatórios sorteados a partir de uma distribuição uniforme entre 0,0 e o valor entre parênteses. Demais símbolos: ver texto.

No Anexo 4, iv) está o pseudocódigo do algoritmo e a Figura 2-3 ilustra seu funcionamento.

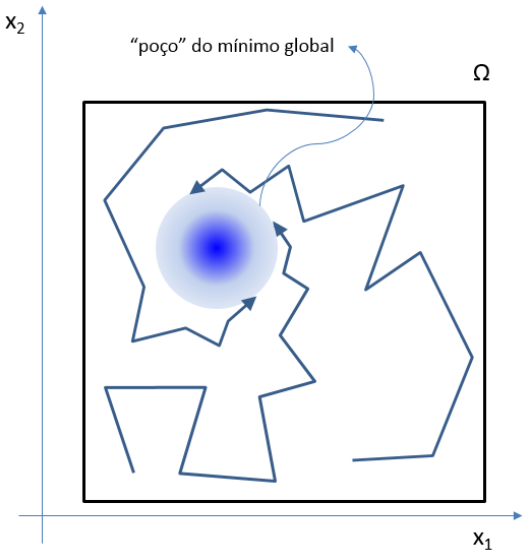


Figura 2-3: Ilustração do funcionamento do algoritmo PSO para encontrar o mínimo global de uma função-objetivo F de apenas dois parâmetros (p_1 e p_2). Ω: domínio de busca.

2.1.4 Algoritmo Genético (GA)

GA é um algoritmo mais sofisticado, diferente de algoritmos de busca estocástica simples conforme visto anteriormente. GA pertence à classe dos algoritmos bioinspirados, dentro

do campo da Computação Natural. As origens de GA remontam ao próprio Allan Turing, um dos pais da computação moderna. Turing (1950) propôs que uma máquina capaz de aprender funcionaria conforme os princípios da evolução de Darwin. Desde Turing até o fim dos anos 1960, uma série de publicações contendo modelos matemáticos e numéricos do processo evolucionário se seguiu até o trabalho de Fraser e Burnell (1970), que já continha todos os elementos de um GA moderno. GA finalmente tornou-se popular com o trabalho de Holland (1975), quando já se havia consolidado como método de otimização (busca de máximo/mínimo de funções).

GA por si só é um termo genérico para uma classe de algoritmos que utilizam os chamados operadores genéticos: seleção, cruzamento e mutação; e usam estruturas de dados para modelar populações, gerações e indivíduos. Os operadores genéticos operam sobre essas estruturas de dados assim como os operadores aritméticos operam sobre números. Os diferentes tipos de GA vêm das diferentes formas com que esses operadores funcionam e de como os genes são representados como estruturas de dados.

O emprego de GA como método de otimização parte do princípio de que os genes da solução ótima se encontram dispersos no *pool* genético de uma população. A solução ótima então naturalmente emerge na população após algumas poucas gerações (iterações) sob ação dos operadores genéticos. O operador seleção trata da eliminação de soluções ruins da população (sobrevivência do mais adaptado). No entanto, deve haver uma probabilidade de sobrevivência de soluções ruins, pois indivíduos fracos garantem a diversidade genética da população. O operador cruzamento trata da troca de genes entre indivíduos para gerar descendentes diferentes e para combinar genes favoráveis que apareceram em indivíduos separados. O operador mutação é a componente estocástica do método e consiste na mudança aleatória de um certo número de genes em um certo número de indivíduos.

O software utilizado nos experimentos desta tese tem uma implementação do pseudocódigo de GA apresentado por Grosan e Abraham (2009, Seção 3.1). A partir da definição da função-objetivo (*fitness function* no jargão de GA) nas Seções 5.3 e 5.4.1, os genes de um indivíduo são os parâmetros variográficos de cada estrutura imbricada, tal como ilustrado na Figura 2-4.

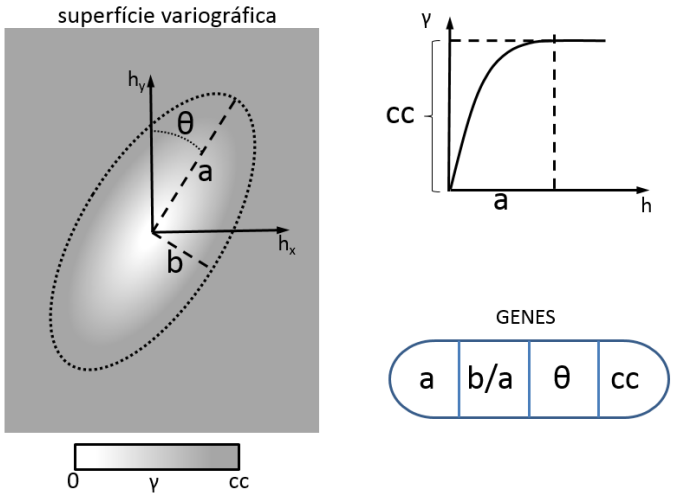


Figura 2-4: Os parâmetros variográficos como uma série de genes de um indivíduo.

A Figura 2-5 ilustra como o problema de otimização desta tese pode ser representado como elementos de GA.

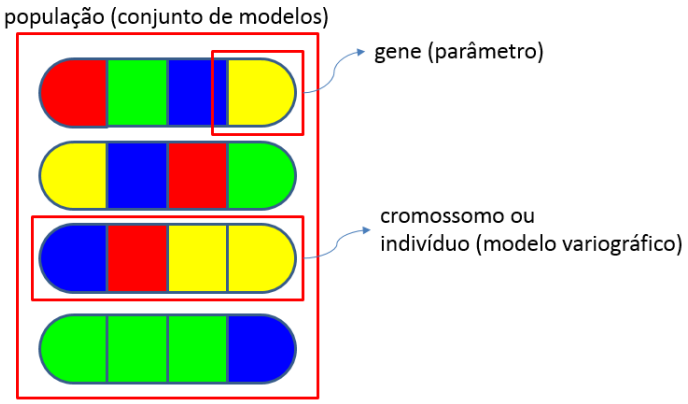


Figura 2-5: Ilustração dos modelos variográficos como elementos de algoritmo genético. As cores representam variações nos valores dos parâmetros.

Apesar de GA ter muitas variações, a Figura 2-6 ilustra a ideia geral de funcionamento de como a solução ótima pode ser encontrada através dos operadores genéticos.

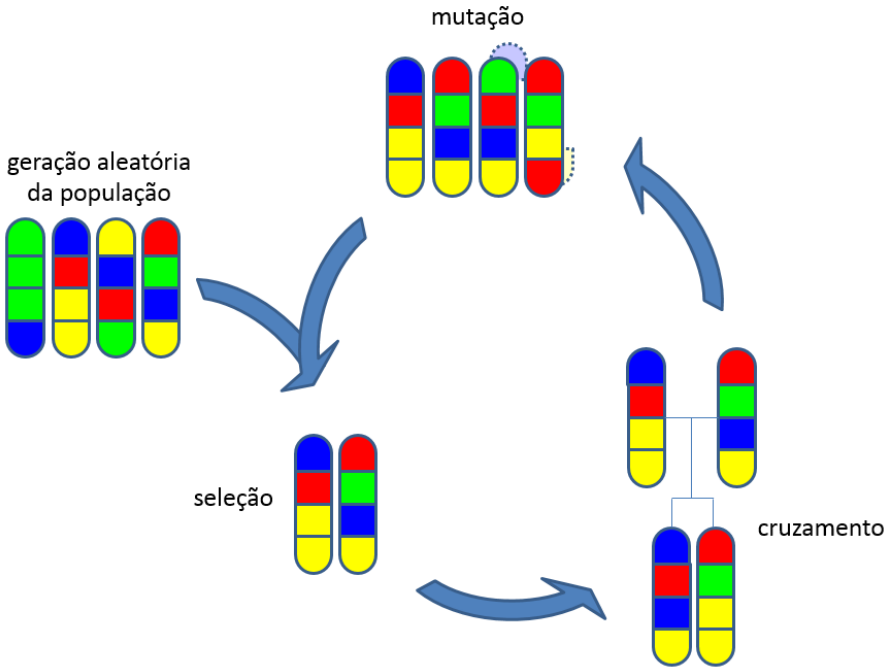


Figura 2-6: Fluxo geral de funcionamento de um algoritmo genético.

No Anexo 4, v) está o pseudocódigo do GA em sintaxe semelhante a C++ tal como está implementado no software empregado nos experimentos. O operador seleção é do tipo torneio binário (*binary tournament*), ou seja, pares de indivíduos são sorteados dentre a população, são confrontados e o pior avaliado de ambos é descartado.

2.2 Métodos de decomposição estrutural

Esta seção discorre em maior detalhe acerca dos métodos de decomposição estrutural testados e apresentados no Estado da Arte (Seção 1.1). O emprego de aprendizado de máquina é apresentado em Seção separada (2.5).

2.2.1 Krigagem Fatorial

A krigagem fatorial (Matheron 1982; Sandjivy 1984; Jaquet 1989; Goovaerts 1997, seção 5.6; Wen e Sinding-Larsen 1997; Wackernagel 2003, cap. 15), por vezes chamada também de análise com krigagem fatorial (FKA) ou análise com krigagem, é um desenvolvimento da krigagem (Matheron, 1963). FK é um filtro espacial baseado no princípio de que a informação em domínio espacial é uma soma de componentes com estruturas independentes, chamados de "fatores regionalizados" ou simplesmente "fatores". FK também pode ser usada para análise com séries temporais (Goovaerts e Sonnet, 1993). Cada fator corresponde a uma estrutura variográfica presente no variograma (ver Figura 1-3).

FK resulta em uma estimativa decomposta nos seguintes fatores:

- 1º fator: a média de krigagem (ver krigagem da média, KM, Wackernagel 2003, cap. 4).
- 2º fator: o efeito pepita, modelado no variograma como o valor em h=0. Corresponde à informação sem estrutura (ruído).
- do 3º fator em diante: modelados individualmente como estruturas imbricadas no variograma. Eles correspondem às componentes contendo as estruturas espaciais, sejam artefatos espacialmente estruturados (ex.: marcas de aquisição sísmica) ou estruturas geológicas.

Se o modelador somar todos os fatores juntos, o resultado equivale ao de uma krigagem exata (simples ou ordinária) feita com o mesmo modelo variográfico.

Wen e Sinding-Larsen (1997) demonstraram as vantagens de FK sobre outros tipos de filtro. No entanto, filtros espectrais são formalmente equivalentes a FK (Galli e Sandjivy, 1985), embora, segundo Telford *et al.* (1990, seção 4.4.8), ruído, artefatos e informação útil podem se sobrepor no domínio de frequência. Contudo, a eficácia do método depende sensivelmente da modelagem do variograma teórico. E, devido ao esforço requerido, normalmente ajusta-se apenas um variograma global aos dados, portanto não modela plenamente características locais variáveis, por exemplo, mudanças na direção da anisotropia. Apesar de não empregada no fluxo de trabalho proposto nesta tese, por suas qualidades, o resultado da krigagem fatorial é utilizado como referência.

2.2.2 Transformada de Fourier

A Transformada de Fourier (Fourier, 1822) muda um sinal (ou uma imagem) do domínio do tempo (ou do espaço) para o domínio da frequência. A Figura 2-7 ilustra a ideia dessa transformação. Segundo ela, uma forma qualquer pode ser expressa como a sobreposição de formas periódicas, resumidas por seus parâmetros (amplitude, frequência, etc.).

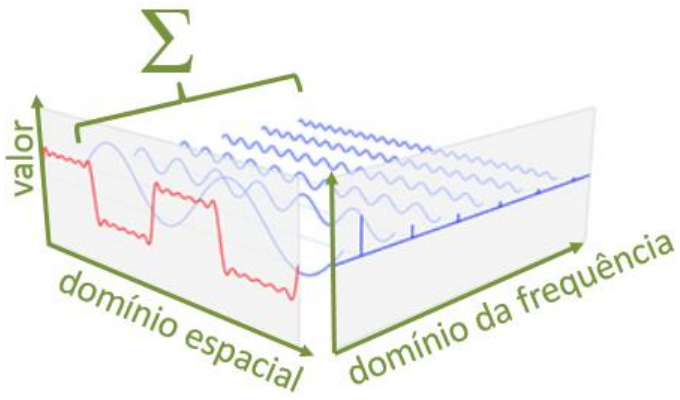


Figura 2-7: Ilustração da ideia da Transformada de Fourier. Modificado de [11].

No caso da Transformada de Fourier, a forma fundamental é uma função trigonométrica, notadamente seno e cosseno, as chamadas funções-base ou *basis functions*. Essas funções-base formam uma base funcional ortonormal (no espaço de Hilbert) para construir qualquer sinal, assim como os vetores ortonormais (1,0) e (0,1) são uma base ortonormal (no espaço Euclidiano) para construir qualquer vetor 2D (x, y).

Em rigor, essa transformada envolve integrais de -∞ a +∞. Na prática, se usa a Transformada Discreta de Fourier ou DFT (Cooley e Tukey, 1965), pois os métodos computacionais requerem dados amostrados discretamente, como é o caso das imagens. Desde então diversos algoritmos de DFT emergiram, dos quais o de Transformada Rápida de Fourier (FFT) de Cooley-Tuckey é mais empregado.

A transformada de Fourier resulta em números complexos, assim um algoritmo de DFT resulta em duas imagens: uma contendo as partes reais *a* e outra, as partes imaginárias *b* de números complexos z expressos na forma retangular z = a + b*i*. O algoritmo, a depender de como foi implementado, também pode resultar na forma polar dos complexos: r cis θ[12], onde *r* é a magnitude e *θ* é o ângulo (fase) do complexo no plano de Argand-Gauss (Figura 2-8).

[11] "Fourier transform," accessado em 15 de julho de 2018, https://en.wikipedia.org/wiki/Fourier_transform

[12] Notação abreviada da forma polar de um número complexo: r cis θ = r(cos(θ) + i sen(θ)).

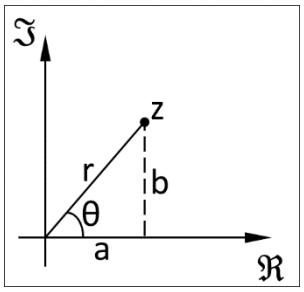


Figura 2-8: O plano de Argand-Gauss mostrando a relação entre as componentes de um número complexo z expresso na forma retangular (z = a + bi) e expresso na forma polar (z = r cis θ). a e b são as componentes real e imaginária de z; r e θ são o módulo (amplitude) e o argumento (fase) de z.

A Figura 1-4 ilustra um exemplo em que o espectro de frequências espaciais é manipulado para separar tendências de larga escala (baixa frequência espacial) das variações locais (altas frequência espaciais). A edição do espectro tem duas limitações: i) normalmente se usa um espectro global, portanto, características variáveis locais não são discerníveis; ii) a transformada de Fourier tende a concentrar a informação, antes espalhada pela imagem, no centro do espectro, o que pode dificultar o discernimento de múltiplas componentes de menor frequência. Não obstante, a transformada de Fourier e sua inversa figuram como passos intermediários no método proposto nesta tese. Nas implementações em software utilizadas neste trabalho, todas as operações envolvendo FFT foram realizadas através da biblioteca fftw (Johnson e Frigo, 2008), considerada a de maior desempenho computacional dentre as opções gratuitas e de código aberto.

2.2.3 Decomposição em Valores Singulares

A Decomposição em Valores Singulares (Singular Values Decomposition – SVD – Forsythe e Moler, 1968) é um dos muitos métodos de decomposição de matrizes (ex.: QR, Gaussiana, etc.). SVD é a extensão da decomposição polar (Richter, 1952; Higham, 1986) para uma matriz não-quadrada qualquer. SVD decompõe uma matriz como na Equação 2-5, em que as colunas de U e de V são vetores que formam bases ortonormais e Σ é uma matriz diagonal que contém, em sua diagonal principal, os chamados valores singulares da matriz em ordem decrescente. O símbolo de estrela (*) denota a matriz transposta conjugada (ou transposta Hermitiana) de uma matriz.

$$A = U\Sigma V^*$$

Equação 2-5: Uma matriz A decomposta com SVD.

Devido às suas características e propriedades, SVD possui muitas aplicações e nesta tese ela é utilizada para decompor uma imagem bidimensional, que pode ser vista como uma matriz,

em uma soma de imagens fundamentais (ver Seções 1.1.3, 3.3 e 4.1). A decomposição de uma matriz como uma soma de matrizes é obtida com a Equação 2-6, onde U_k e V_k são os vetores-colunas correspondentes às k-ésimas colunas das matrizes *U* e *V* resultantes da SVD. O escalar σ_k é o k-ésimo valor singular, isto é, o k-ésimo elemento da diagonal principal da matriz Σ resultante da SVD. O operador $\otimes$ é o produto diático ou tensorial entre dois vetores.

$$M = \sum_{k=1}^{n} \sigma_k U_k \otimes V_k^*$$

Equação 2-6: Decomposição de uma matriz *M* como soma de matrizes com SVD. Ver texto.

Logicamente, há uma infinidade de formas com que uma matriz pode ser decomposta como uma soma de matrizes. A característica que torna essa soma especial é que os membros da soma estão em ordem decrescente de quantidade de informação, pois cada qual corresponde a um valor singular, que está em ordem decrescente de grandeza na matriz Σ. Essa ordem de valores singulares constitui uma assinatura espectral da matriz, ou da imagem 2D.

SVD é um método de decomposição extremamente fácil de empregar, entretanto, da Equação 2-6 depreende-se que a análise com SVD opera sobre linhas e colunas das imagens apenas. Essa característica, portanto, impossibilita discernir duas estruturas de mesma energia alinhadas em diferentes azimutes.

2.2.4 Empirical Mode Decomposition (EMD)

EMD (Linderhed, 2009) é a extensão da Transformada de Hilbert-Huang (HHT – Huang *et al.*, 1996) para imagens bidimensionais. É um algoritmo empírico, isto é, não deriva de uma base teórica como o algoritmo FFT deriva da Transformada Discreta de Fourier (DFT – introduzida por Cooley e Tukey, 1965). Sendo um método empírico, ou seja, dispensa um modelo teórico *a priori*, ele é adaptativo, isto é, considera variações locais. Entretanto, segundo Chen e Feng (2003), o método só distingue feições em banda estreita. Banda estreita, neste contexto, significa que não é possível separar feições em um amplo intervalo de frequências/escalas espaciais.

Nesta tese foi utilizada a versão IEMD (EMD para imagens bidimensionais) proposta por Linderhed (2009) descrita a seguir.

i) Seja $M_{(i,j)}$ uma imagem de entrada onde *i* e *j* são os índices matriciais da imagem.

ii) Localizar os máximos e mínimos locais da imagem $M_{(i,j)}$ (Figura 2-9).

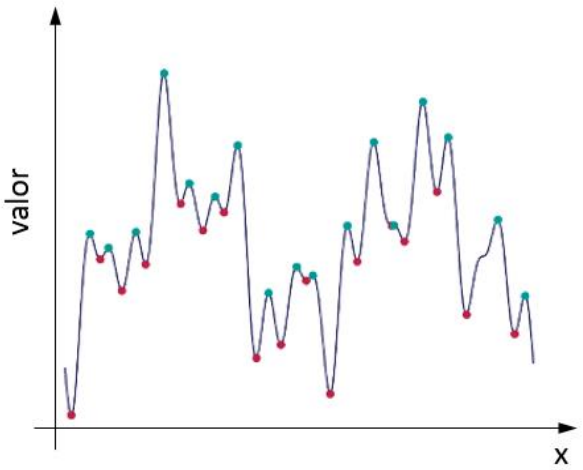


Figura 2-9: EMD: localização dos máximos e mínimos locais (pontos verdes e vermelhos respectivamente). Modificado de [13].

Interpolar uma superfície para os máximos locais, $e^{+}_{(i,j)}$, e outra para mínimos locais, $e^{-}_{(i,j)}$. Linderhed (2009), explicitamente, diz para empregar algum tipo de Spline (Press *et al.* 1992, Seção 3.3). Outro método de interpolação de superfícies como com Radial Basis Functions (RBF – Fornberg e Piret, 2007, Seção 2.1), Discrete Smooth Interpolation (DSI – Mallet, 1989) ou Krigagem (normalmente a Dual – Rasera, 2014, pp16-20 ou a Global – Neufeld e Wilde, 2005 para evitar artefatos de vizinhança de busca) poderia ser experimentado. Essas superfícies são chamadas de envelopes (Figura 2-10).

[13] https://en.wikipedia.org/wiki/Hilbert%E2%80%93Huang_transform, acessado em 04/11/2018.

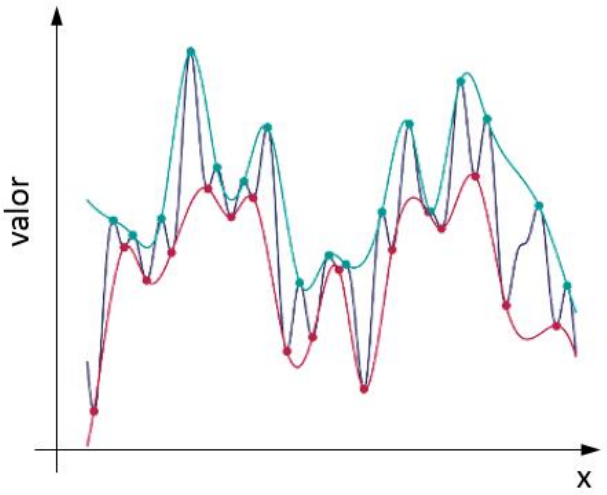


Figura 2-10: EMD: obtenção dos envelopes de máximos e de mínimos locais (linhas verde e vermelha respectivamente). Modificado de [14].

iii) Encontrar o envelope médio, $\bar{e}_{(i,j)} = \frac{e^{+}_{(i,j)} + e^{-}_{(i,j)}}{2} \; \forall \; i, j$ (Figura 2-11).

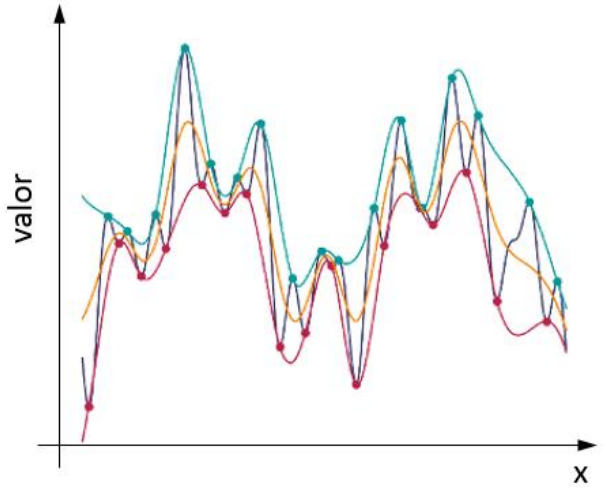


Figura 2-11: EMD: obtenção do envelope médio (linha cor-de-abóbora). Modificado de [15].

iv) Subtrair da imagem $M_{(i,j)}$ o envelope médio, $\bar{e}_{(i,j)}$, formando o sinal candidato: $c_{(i,j)} = M_{(i,j)} - \bar{e}_{(i,j)}$.

v) Se todos os valores do envelope médio $\bar{e}_{(i,j)}$ são próximos de zero (ou zero em teoria), $c_{(i,j)}$ é uma *intrinsic mode function* (IMF, Figura 2-12) e o resíduo $r_{(i,j)} = M_{(i,j)} - c_{(i,j)}$ será o dado de entrada para a próxima iteração. Caso contrário, $c_{(i,j)}$ não é

[14] https://en.wikipedia.org/wiki/Hilbert%E2%80%93Huang_transform, acessado em 04/11/2018.

[15] https://en.wikipedia.org/wiki/Hilbert%E2%80%93Huang_transform, acessado em 04/11/2018.

uma IMF e, ao invés disso, é usado como dado de entrada para a próxima iteração. Em rigor, a última IMF não deve ter mínimos e máximos locais e é a tendência global da imagem, mas esse critério pode ser relaxado para imagens 2D e 3D por motivos práticos.

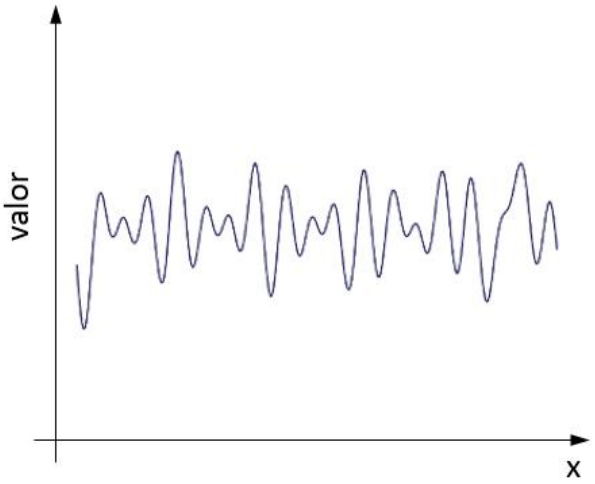


Figura 2-12: EMD: obtenção da primeira IMF (linha azul-escuro). Modificado de [16].

Os passos são repetidos até a satisfação de um critério de convergência (ex.: número de extremos abaixo de um limiar), quando um determinado número de IMFs for encontrado ou quando um número de repetições for executado. O dado original pode ser obtido somando-se todas as IMFs e o útimo resíduo do passo v): $M = \sum_{k=1}^{n} \mathrm{IMF}_k + r_{n+1}$, onde *n* é o número de IMFs encontradas. O método se baseia em *zero crossings* (locais onde a imagem passa pelo valor zero), portanto é necessária alguma transformação tal como subtrair a média global de todos os valores ou extrair a tendência (*trend*) global para trabalhar com valores centrados em zero.

A implementação do método para esta tese (ver software na Seção 1.5, i) foi feita com interpolação dos extremos locais pelo método de Shepard (inverso da distância elevado a uma potência, normalmente 2) e com *Thin Plate Spline* (Linderhed, 2009) com função radial $r^2 \log r$. A implementação também conta com duas opções para os extremos locais: como pontos ou como linhas ("vales" e "divisores de água").

[16] https://en.wikipedia.org/wiki/Hilbert%E2%80%93Huang_transform, acessado em 04/11/2018.

2.2.5 Filtro de Gabor

A transformada de Fourier (Seção 2.2.2) é uma análise no domínio da frequência. O termo "frequência" pressupõe repetição. No caso da análise de Fourier, a repetição é representada por funções periódicas (particularmente senos e cossenos) que se estendem ao infinito. Portanto, não obstante sua substancial utilidade, ser uma análise de frequências constitui uma limitação no que tange à representação de feições localizadas e isoladas.

Para ilustrar esse problema, imagine-se o sinal de áudio de uma ópera com duração de 1 hora. Na análise de Fourier, é fácil isolar componentes dominantes, tal como as notas de um instrumento que tenha tocado o tempo todo, pois a quantidade de informação é grande e as notas musicais têm frequências precisas (banda estreita). Agora, imagine-se que desejemos isolar o canto de um ator que só participou por 30 segundos. No espectro de frequência somente, a contribuição desse ator (a voz humana tem muitos harmônicos, ou seja tem banda larga) seria equivalente ao de um ruído (que também tem banda larga) que estivesse presente durante 1h. Isto é, sinais fortes, mas localizados, podem ser indistinguíveis de sinais fracos, porém estacionários, no espectro de frequências.

Um espectro de Fourier, como visto na Seção 1.1.2, é um modelo que expressa a imagem como amplitudes e fases (exemplo na Figura 2-13) em função da frequência espacial apenas. Assim, feições localizadas, como uma falha ou uma dobra, acabam diluídas no ruído de fundo do espectro de Fourier.

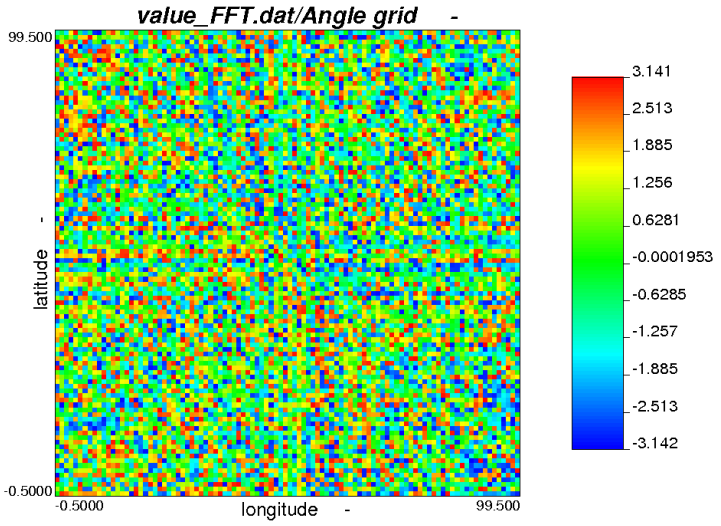


Figura 2-13: O mapa de fases (em radianos) do dado de teste (Figura 1-10).

Para resolver esse problema, é necessário preservar a dimensão espacial no espectro e realizar a análise de frequências localmente, em pequenos intervalos chamados janelas. Em sinais no

tempo, esse tipo de análise é amplamente conhecido por análise tempo-frequência (*time-frequency*).

A Windowed Fourier Transform (WFT - Allen e Rabiner, 1977) é uma Transformada de Fourier feita localmente, por exemplo em áreas de 20 × 20 pixels em um mapa de dimensões maiores, daí o nome "windowed". A chamada Transformada Gabor (Gabor, 1946; Chui, 1992, seção 3.1; Feichtinger e Strohmer, 1998) é um caso de WFT invertível, daí sua importância. A inversibilidade vem do fato de que os valores da janela são ponderados pela extensão significativa de uma função Gaussiana, pois a transformada de Fourier de uma função Gaussiana é ela mesma, assim a mesma janela pode ser usada tanto no domínio espacial quanto no de frequências.

Uma WFT, como a Transformada Gabor, resulta em um espectro de frequências com dimensões extras: as das frequências e tantas outras quantas forem as dimensões espaciais. Por exemplo, a WFT de um mapa 2D resulta em um volume 4D. Esse espectro com dimensões extras recebe o nome de espectrograma (Figura 2-14, c).

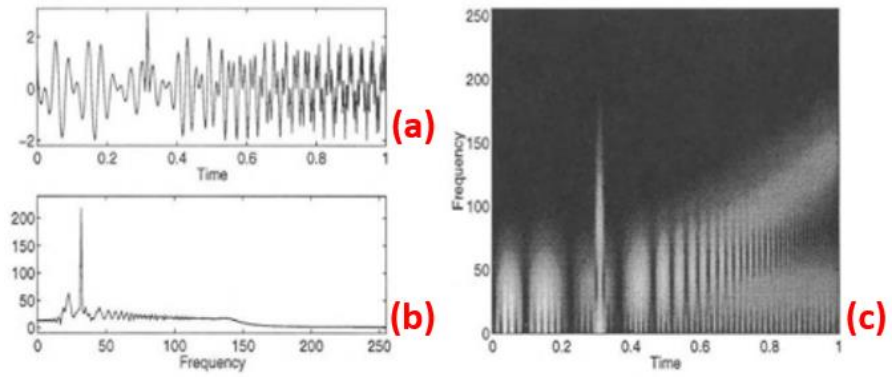


Figura 2-14: WFT de um sinal unidimensional. (a) é uma série temporal; (b) é sua transformada de Fourier com a intensidade do sinal em função da frequência; (c) é sua WFT (espectrograma) como a intensidade do sinal (tons de cinza) em função da frequência e do tempo. Modificado de Chui (1992, p. 9, fig. 4).

Contudo, para dados bidimensionais, a WFT apresenta um problema de ordem prática. A transformada de Fourier de um mapa (ver exemplo na Figura 1-4) já tem duas dimensões, portanto o seu espectrograma teria que ter quatro dimensões, o que complica a análise e a interpretação. Poder-se-ia reduzir o espectrograma a três dimensões caso se tomasse um azimute específico e realizar a análise azimute-a-azimute, porém os dados arrumados em malha regular só estão alinhados em dois azimutes, por exemplo: N000E e N090E se a malha não estiver rotacionada. Para azimutes intermediários, seria necessário extrair amostras ao longo de bandas radiais de comprimento equivalente ao tamanho da janela. Seria necessário ainda interpolar esses valores de forma a regularizá-los para uma transformada de Fourier

unidimensional. Ainda, a sobreposição das bandas, dependendo das suas larguras, perto do ponto de pivotação pode causar confusão dos espectros.

O filtro de Gabor (não confundir com a transformada de Gabor vista anteriormente) é um núcleo (*kernel*) de convolução cujos parâmetros mais relevantes são a frequência e a orientação (azimute), assim também permite levantar o conteúdo espectral local variável. Ora, computacionalmente, a convolução é uma operação natural entre malhas regulares, portanto não há as implicações práticas de computar FFTs 1D ao longo de bandas radiais. Marĉelja (1980) ainda propôs que a resposta desse filtro modela a forma com que o córtex visual dos mamíferos discrimina as diferentes texturas que há nas imagens. Então, de forma equivalente, pode-se usar o filtro de Gabor para extrair da imagem as feições estruturais de frequência espacial e orientação específicos.

Há dois núcleos Gabor: a) para a componente real da resposta, ele resulta da multiplicação entre a função Gaussiana e a função cosseno (Figura 2-15); b) para a componente imaginária da resposta, é a função seno (ver Seções 1.1.2 e 2.2.2). Como a função Gaussiana se estende ao infinito, na prática só uma pequena parte em torno da moda considerada significante é utilizada. Portanto, o papel da função Gaussiana é definir uma localização no espaço (função-janela ou *window function*) e ela modula a função trigonométrica porque a transformada de Fourier de uma função Gaussiana é ela mesma, assim a mesma janela definida no domínio do espaço pode ser usada no domínio da frequência. E a função modulada é a (cos)seno porque é a função-base (*basis function*) do espaço gerado pela transformada de Fourier.

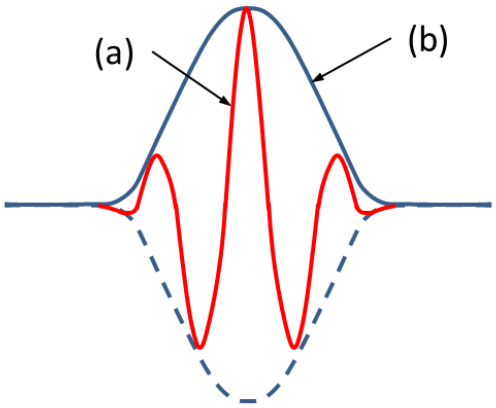


Figura 2-15: Um núcleo de Gabor em 1D para a componente real da resposta do filtro definido como uma função cosseno (a) modulada por uma função Gaussiana (b). Em 1D, bastam três parâmetros para definir esse filtro: frequência e fase da função cosseno e o desvio-padrão da função Gaussiana.

Em termos práticos, se o núcleo for grande, transforma-se o filtro de Gabor e a imagem de entrada com FFT para reduzir a convolução a um produto célula-a-célula (ver Teorema da Convolução – Hunt, 1971) de menor demanda computacional. No caso 1D, basicamente se varia a frequência e, para cada frequência, o núcleo resultante é convolvido contra o sinal ao longo da dimensão espacial (*x*). As respostas são plotadas em um gráfico de três variáveis: a dimensão espacial, a frequência e a correlação (normalmente em uma escala de cores). Esse diagrama é chamado de espaço de Gabor.

A Figura 2-16 ilustra o funcionamento, embora as convoluções aconteçam no domínio da frequência como produtos simples. Pode-se trabalhar apenas com a parte real da resposta (núcleo com cosseno), apenas com a parte imaginária (núcleo com seno) ou ambas, para, por exemplo, obter a resposta como amplitude (ver Seção 3.5.3).

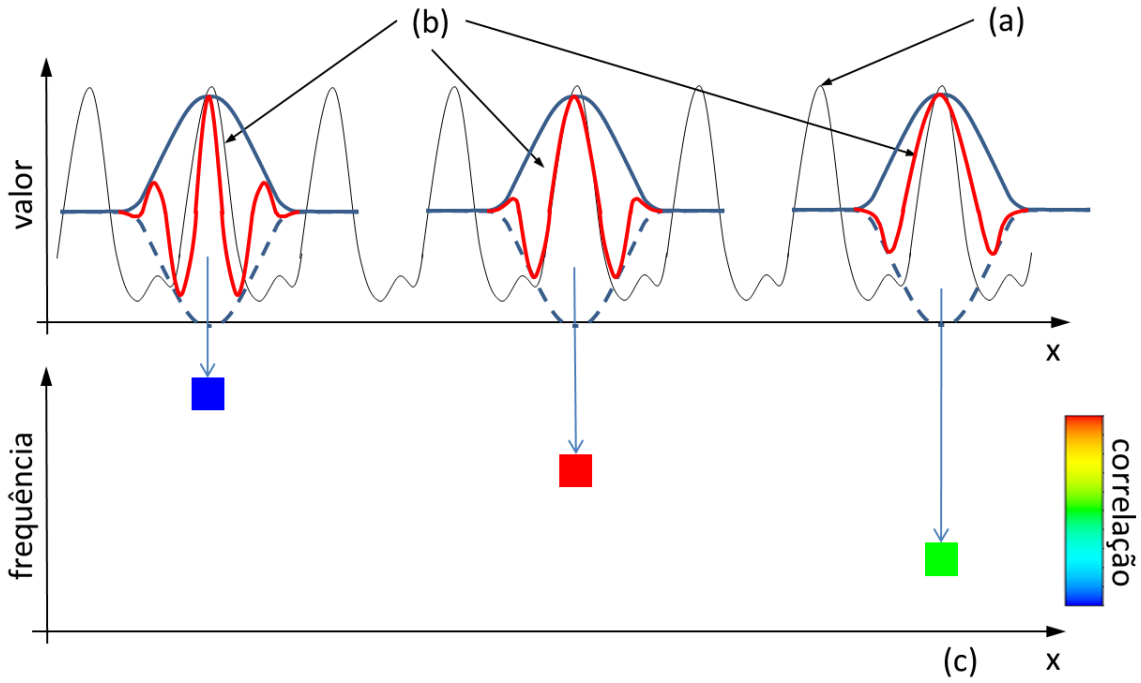


Figura 2-16: Ilustração de funcionamento do filtro de Gabor como convoluções em domínio espacial para facilitar a compreensão. O núcleo de Gabor (uma (cos)senoide modulada por uma função Gaussiana) mostrada em (b) tem sua frequência variada e convolvida contra o sinal (a) dentro de intervalos fixos (janela) centrados na célula da iteração. Os valores local × frequência × correlação são plotados no espaço de Gabor (c) formando uma imagem para interpretação.

Em 2D, o núcleo (Figura 2-17) é modelado através de seis parâmetros, a saber: λ, o comprimento de onda que controla a frequência espacial da função (cos)seno; θ, que controla a orientação das ondas senoidais; m_x e m_y, as médias/modas da função Gaussiana que controlam a sua posição relativa à origem do núcleo; σ_a e σ_b, os desvios padrões da função Gaussiana em seus semieixos maior e menor. A Equação 2-7, a chamada função Gabor, é

usada para gerar a superfície que é avaliada em cada célula de uma malha regular e, com isso, obter um núcleo de convolução utilizável em algoritmos computacionais. O Anexo 3 contém um script de calculadora para gerar um núcleo em um mapa.

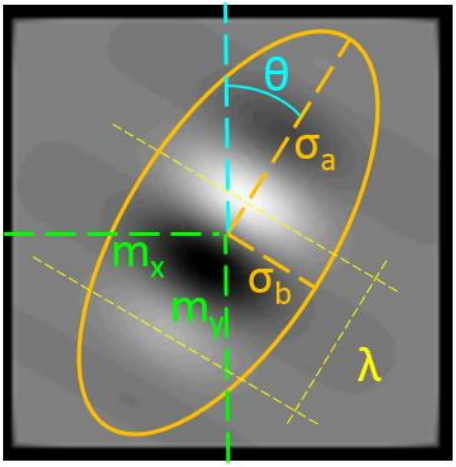


Figura 2-17 Núcleo Gabor 2D (para a componente imaginária da resposta do filtro) e seus parâmetros (ver texto).

$$x' = (x - m_x)\cos\theta + (y - m_y)\sin\theta$$
$$y' = -(x - m_x)\sin\theta + (y - m_y)\cos\theta$$
$$g(x,y) = \exp\left(-\frac{x'^2}{2\sigma_a^2} - \frac{y'^2}{2\sigma_b^2}\right)\cos\left(\frac{2\pi x'}{\lambda}\right)$$

Equação 2-7: função Gabor para obter a componente real da resposta do filtro. λ, θ, m_x, m_y, σ_a e σ_b são os parâmetros do filtro de Gabor (ver texto). Para obter a componente imaginária basta substituir o cosseno na terceira linha da expressão por seno.

Para obter o espaço de Gabor completo de um mapa 2D, faz-se variar os parâmetros mais importantes, λ e θ, que determinam, respectivamente, a frequência espacial e orientação das feições a serem extraídas. Os parâmetros m_x e m_y são normalmente o centro do núcleo (ex.: 120, 120 se o núcleo medir 240 × 240 unidades de comprimento). σ_a e σ_b são fixados dependendo do desejo do modelador de dar maior ou menor peso às amostras mais distantes durante o processo de convolução.

Para o caso 2D, como se varia dois parâmetros (orientação e frequência) e a reposta do filtro é também um mapa bidimensional, o espaço de Gabor completo teria quatro dimensões. Então, por motivos práticos, faz-se a análise fixando um dos parâmetros (por exemplo, o azimute) para obter um espaço de Gabor na forma de um volume 3D (exemplo na Figura 1-8) ou resumindo cada resposta do filtro a uma métrica (ex.: média) e com isso obter uma imagem frequência versus azimute (exemplo na Figura 3-11).

Neste método de decomposição estrutural, efetivamente um espectro de frequências espaciais sob a forma de respostas a um núcleo de convolução é obtido dando origem a um tipo de espectrograma em que feições não estacionárias de um sinal podem ser observadas e trabalhadas. Porém, o princípio da incerteza do processamento de sinais (Equação 2-8 – Chui, 1992, seção 3.2) estabelece que quanto menor a janela (maior resolução espacial), menor a gama de frequências discerníveis (menor resolução em frequência) e vice-versa. Isto, portanto, constitui uma limitação na capacidade de discernir feições caso tenham estruturas espaciais muito diferentes.

$$\Delta x \Delta \omega \geq 1/2$$

Equação 2-8: Princípio da incerteza do processamento de sinais. Δx é a resolução espacial (ou janela); Δω é a resolução em frequência angular (ω=2πf, onde *f* é a frequência espacial).

2.2.6 Transformada Ondaleta

A transformada ondaleta contorna o princípio da incerteza da análise de Gabor (Seção 2.2.5) ao introduzir as janelas (escalas) variáveis e fixando a forma do núcleo de convolução (ondaleta). A Figura 2-18 ilustra a diferença entre o resultado da análise de Gabor (um espectrograma) e o resultado de uma transformada ondaleta (um escalagrama). Assim, obtém-se melhor resolução espacial para as altas frequências e melhor resolução em frequência para as estruturas que variam lentamente (baixa frequência espacial) em relação à análise de Gabor.

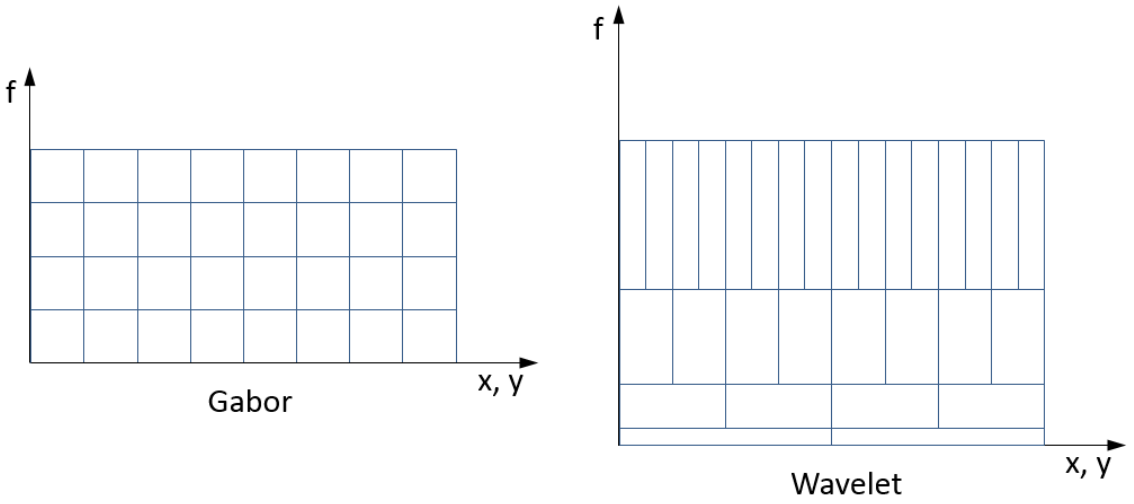


Figura 2-18: Diferença entre um espaço de Gabor (esquerda) e um escalagrama (direita).

Uma ondaleta (Graps, 1995) é um modelo de oscilação limitado (portanto localizável no espaço) que, quando convolvido contra um sinal, realça as oscilações para as quais a ondaleta foi construída. A ondaleta (conhecida como ondaleta-mãe) deve satisfazer determinados

critérios, como, por exemplo, quando reescalada, dá origem a ondaletas-filhas as quais devem formar uma base ortonormal no espaço de Hilbert (espaço de funções brevemente discutido na Seção 2.2.2). O realce de feições acontece pela correlação que ocorre entre a ondaleta e quaisquer estruturas correspondentes que possa haver na imagem.

Em termos práticos, uma ondaleta é escolhida ou construída pelo modelador, com a qual é aplicada uma transformada ondaleta (ver teoria em Chui, 1992, pp.6-9; Meyer, 1995), normalmente através de um algoritmo de Transformada Discreta de Ondaleta (DWT – Haar, 1910), como a Transformada Rápida de Ondaleta (FWT – Mallat, 1989). Em seguida, a imagem resultante (chamada de escalagrama) é editada (similarmente à edição do espectro visto na Seção 1.1.2) de forma a eliminar, atenuar, realçar ou isolar as feições desejadas. O escalagrama traz destacadas as partes da imagem onde houve correlação com a ondaleta, o que permite a separação através de limiares (*thresholds*). Por fim, o escalagrama editado é retrotransformado contra a ondaleta para obtenção da imagem original filtrada.

Para ilustrar o funcionamento, tome-se como exemplo o caso do sinal monótono e unidimensional da Figura 2-19 (a). Primeiro, o modelador escolhe ou constrói a ondaleta-mãe pensando em como ela ressoará com as formas (Figura 2-19 (b)). O algoritmo então a reescala segundo um certo número de escalas (transformada discreta), produzindo as ondaletas-filhas (Figura 2-19 (c)). A seguir, cada ondaleta-filha é convolvida contra o sinal ao longo da dimensão espacial (x). Os valores das correlações obtidos são plotados em um diagrama chamado escalagrama (Figura 2-19 (d)). Em 1D, o escalagrama tem três dimensões: a dimensão espacial; a escala e a correlação (normalmente mostrada em escala de cores).

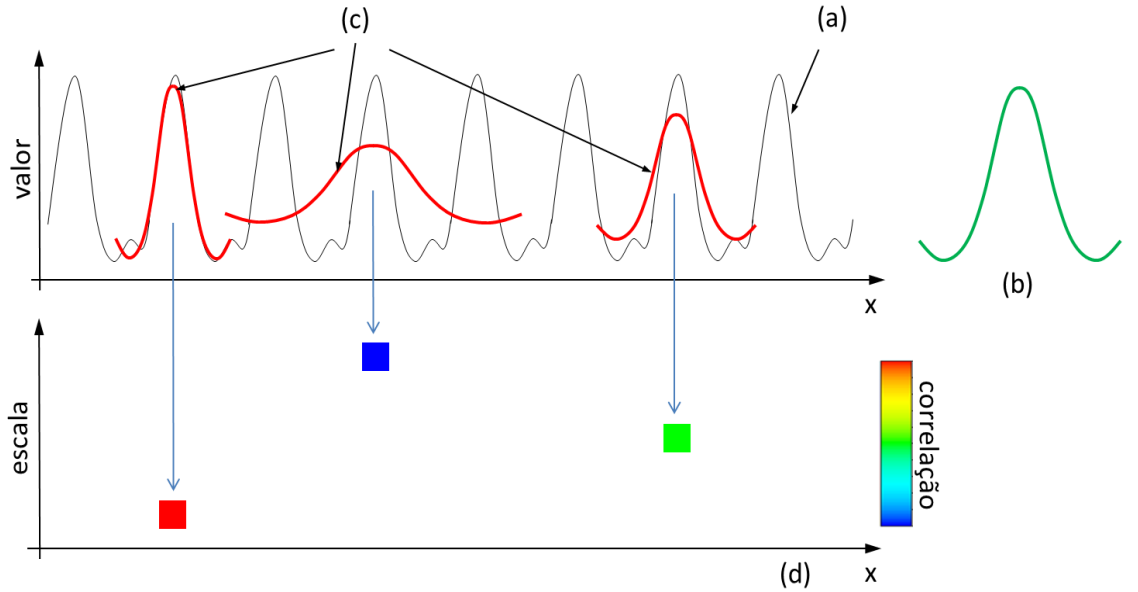


Figura 2-19: Ilustração de funcionamento da transformada ondaleta. (a): um sinal a ser analisado; (b) ondaleta-mãe; (c): ondaletas escaladas; (d): escalagrama. Este exemplo mostra a geração da imagem do escalagrama em apenas três pontos para facilitar a compreensão.

2.3 Cálculo eficiente de mapa variográfico

A função-objetivo F, conforme apresentado mais adiante, passa pela avaliação da variografia experimental. A variografia experimental levantada em todas as direções, formando um mapa, chama-se mapa variográfico (Marcotte, 1996; Deutsch e Journel, 1998, pp. 55-57).

O processo de otimização escolhido invariavelmente é iterativo (ver Seção 2.1), o que certamente requer que F seja avaliada reiteradamente. Além disso, o gradiente de F para os métodos Gradiente Descendente (Seção 2.1.1) e Busca Linear com Reinício (Seção 2.1.2) é calculado numericamente (Equação 2-3), requerendo uma avaliação de F para cada derivada parcial (da ordem de 10). Isto para cada passo de otimização. Portanto, é deveras importante que o cálculo do mapa (ou volume) variográfico seja célere.

O cálculo do variograma experimental de uma variável (ou entre variáveis diferentes do mesmo mapa para variografia cruzada) corresponde a uma convolução. Do Teorema da Convolução (versão discreta deste teorema em Hunt, 1971), essa operação se reduz a um produto de Hadamard (também conhecido por produto elemento-a-elemento ou produto de Schur, ver Figura 2-20) no domínio da frequência (ver Seção 2.2.2).

$$\begin{bmatrix} 10 & 40 \\ 90 & 160 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \circ \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

Figura 2-20: Exemplo de um produto de Hadamard ou elemento-a-elemento entre duas imagens 2x2.

Assim, Marcotte (1996) propõe um método para computar o mapa variográfico em tempo O(n log n) ao invés de O(n^2) requerido por uma convolução direta (*naïve*) em domínio espacial (ver notação O em [17]). Pode-se estender a aplicação deste método a dados irregulares, desde que seja precedido de um passo de regularização (interpolação, estimativa ou simulação) cujo resultado tenha variografia semelhante ao dos dados irregulares. Devido aos passos adicionais de FFT e retrotransformação, o ganho de desempenho se torna perceptível para imagens maiores (ordem de 10.000 células ou maior). A Figura 2-21 ilustra resumidamente o método, embora o mapa obtido em rigor seja um correlograma (valores maiores no centro e decaindo conforme *h* aumenta), que equivale ao variograma "invertido".

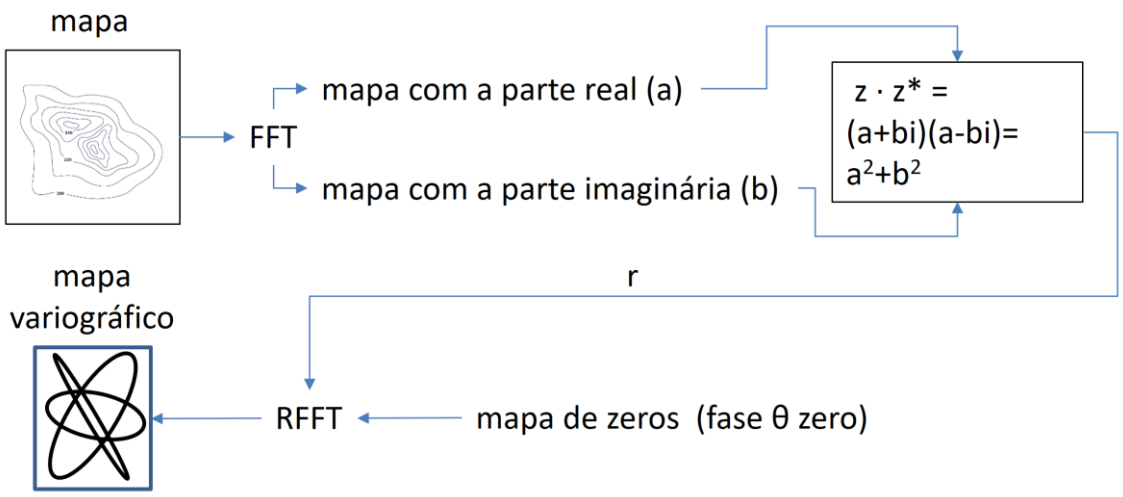


Figura 2-21: Resumo gráfico do fluxo baseado no método proposto por Marcotte (1996) para o cálculo eficiente de mapa variográfico com FFT. z é um número complexo; r e θ são a magnitude e a fase de um número complexo respectivamente. O sinal estrela (*) denota o conjugado de um número complexo. Os valores complexos da FFT devem ser lidos na forma retangular (a + bi) e os valores complexos são passados ao passo de retrotransformação na forma polar (r cis θ).

[17] A notação O-grande (*Big-O*) em Ciência da Computação, especificamente na análise de algoritmos, denota o comportamento de um algoritmo quanto ao tempo de execução (por vezes quanto uso de memória) em função do tamanho do dado de dentrada (n). Por exemplo, um algoritmo cujo tempo de execução aumenta com o quadrado do tamanho da entrada é dito um "algoritmo de complexidade O(n^2)" ou simplesmente "algoritmo O(n^2)".

2.4 Método Integral de Fourier (FIM)

FIM (Pardo-Iguzquiza e Chica-Olmo, 1993) é um método utilizado para produzir realizações em simulações estocásticas rapidamente, pois não é um algoritmo de simulação sequencial. Baseia-se no princípio de que perturbações no mapa de fases resultante da transformada de Fourier (ver Seção 2.2.2), enquanto se fixa o mapa de amplitudes, resulta em imagens com as mesmas formas, porém reposicionadas. Isso é certamente útil para geração de cenários para caracterizar a incerteza.

FIM se baseia nas relações entre uma informação aleatória, sua representação no domínio da frequência, sua densidade espectral e sua autocovariância. Estas relações estão ilustradas na Figura 2-22.

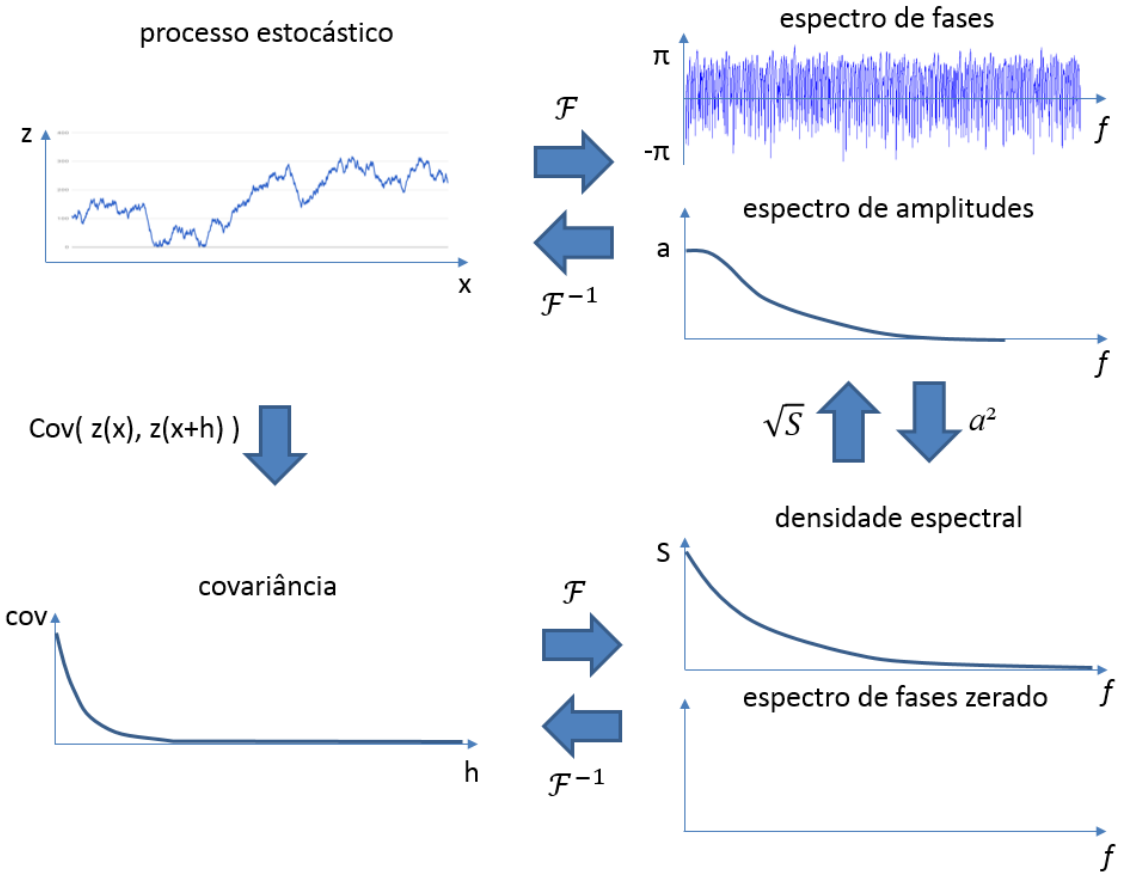


Figura 2-22: Relações entre um sinal aleatório, sua forma em domínio de frequência, sua densidade espectral e sua autocovariância. *z*: valor; *x*: local no espaço; *f*: frequência; *a*: valor de amplitude; *S*: valor de densidade espectral: cov: valor de covariância; *h*: separação no espaço. $\mathcal{F}$ e $\mathcal{F}^{-1}$: transformada Fourier e sua inversa.

Então, de acordo com o mesmo princípio, pode-se fixar o mapa de fases e alterar o mapa de amplitudes para obter imagens com formas diferentes nas mesmas posições. A Figura 2-23 ilustra a aplicação para obter um resultado semelhante à FK (ver seção 2.2.1). O operador raiz quadrada está no fluxo porque o mapa de amplitudes da superfície correlográfica é na

verdade uma densidade espectral. Tirando-se a raiz quadrada desses valores obtém-se amplitudes propriamente ditas para a RFFT com o mapa de fases.

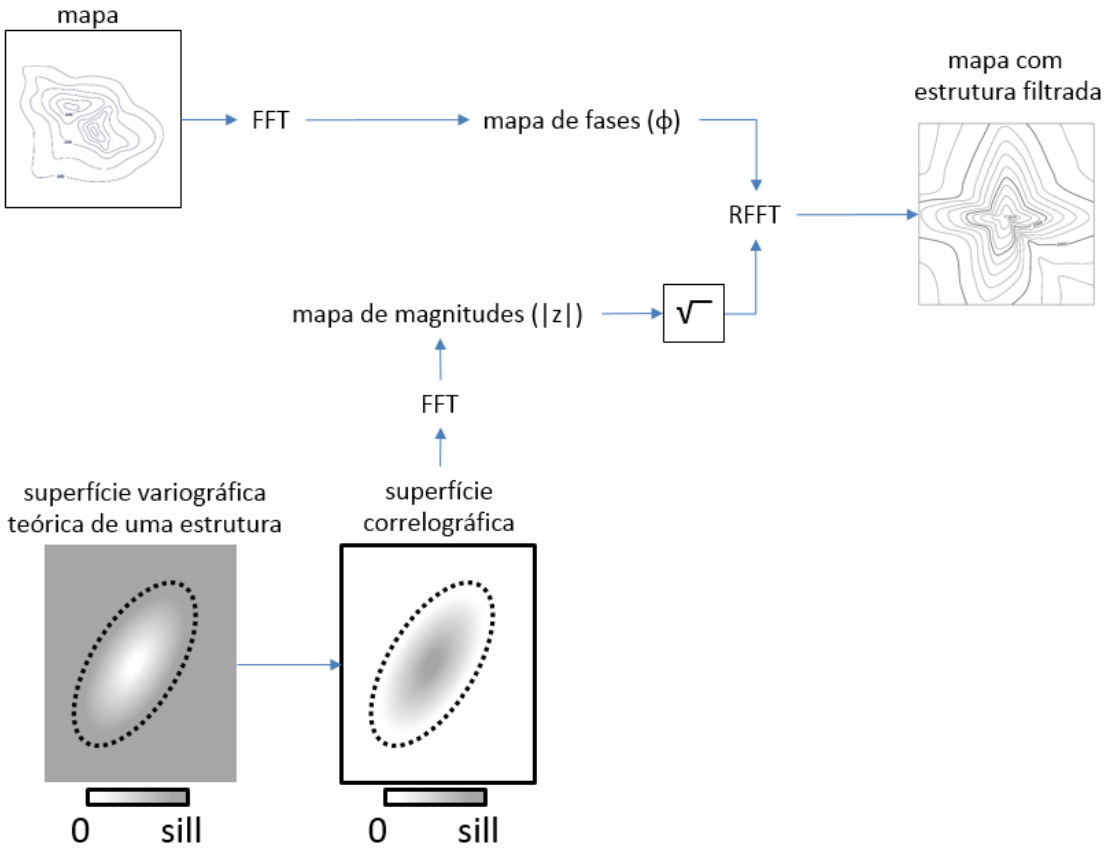


Figura 2-23: Fluxo para síntese de estruturas a partir de um modelo variográfico baseado no FIM.

2.5 Deep Learning

O aprendizado profundo pode ser considerado como outro método de otimização além daqueles apresentados na Seção 2.1. Porém, dados seus conceitos e elementos diversos, o assunto foi destacado em uma Seção dedicada para uma suficiente compreensão.

2.5.1 O neurônio artificial: o perceptron

O campo da Inteligência Artificial pode ser dividido em duas grandes escolas: a simbolista e a conexionista. A escola simbolista consiste em modelar o raciocínio diretamente através da Lógica Matemática, declarando fatos e relações em linguagens de programação de alto nível como Prolog e Lisp. O simbolismo se baseia no fato de que o raciocínio é um comportamento emergente (princípio do todo que é maior que a soma das partes – Green, 1993) de um sistema físico complexo (o cérebro), portanto, não desce ao nível de modelar seus componentes elementares (abordagem de cima para baixo). A escola simbolista é normalmente utilizada para automatizar tarefas que envolvem o raciocínio abstrato como,

por exemplo: prova de teoremas, dedução e processamento de linguagem natural. Seu exemplo mais proeminente nos dias de hoje é o famoso sistema Watson (Lally & Fodor, 2011). Ao abstrair a função cerebral, a abordagem simbolista encontra limitações ao tipo de tarefa que pode executar, pois o cérebro possui outras funções diferentes do raciocínio.

A escola conexionista trata de modelar os constituintes do sistema físico do cérebro – os neurônios[18] e sua interconexão (princípio reducionista) – para que seja possível automatizar, em princípio, qualquer função cognitiva (abordagem de baixo para cima) ao custo de maior demanda computacional. O neurônio artificial é um modelo matemático de um neurônio multipolar (vários dendritos, um axônio) introduzido por Rosenblat (1957), que o chamou de *perceptron*. O perceptron mais simples está representado na Figura 2-24. Seguindo sua contraparte biológica, com um único axônio, o neurônio artificial sempre tem uma única saída. Lembrando que, tal como em um sistema nervoso, um mesmo axônio pode se conectar aos dendritos (conexões de entrada) de mais de um neurônio.

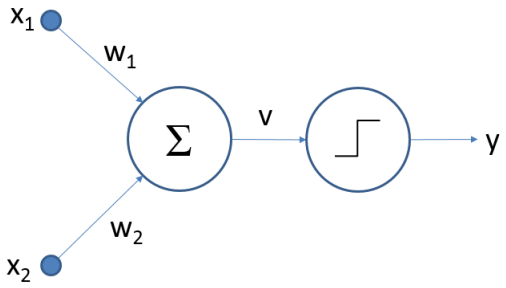


Figura 2-24: Símbolo esquemático do perceptron de Rosenblat (1957) de duas entradas. x_1 e x_2: entradas; w_1 e w_2: pesos; *v*: campo local induzido; *y*: saída. O símbolo Σ denota que as entradas são somadas, após serem multiplicadas pelos pesos. O símbolo em degrau representa a função-degrau de Heaviside.

Algebricamente, o resultado do perceptron pode ser expresso como $y = H(w_1x_1 + w_2x_2)$, ou seja, o resultado do perceptron é uma combinação linear das entradas submetido à função-degrau de Heaviside[19] para ser convertida em um indicador binário. A função que transforma o resultado da combinação linear é conhecida por função de ativação (*activation function* – Ramachandran *et al.*, 2017) que modela como um neurônio natural ativa ou não o axônio

[18] Embora o cérebro possua outros tipos de célula em sua constituição, como por exemplo: astrócitos, oligodendrócitos, etc., a Inteligência Artificial se concentra na modelagem das células envolvidas diretamente na função cognitiva: os neurônios.

[19] A função-degrau é comumente denotada por H( ), θ( ), u( ), **1**( ) ou 𝟙( ).

(saída) em função dos estímulos (entradas). A função de ativação é também conhecida por função de decisão (*decision function*). O valor resultante da combinação linear, antes de ser transformado pela função de ativação, é conhecido por campo local induzido (*induced local field*).

Assim, o perceptron, ou uma combinação de perceptrons, é útil para aprender regras de classificação ou operações booleanas. Modernamente, admite-se que a função de ativação possa ser modificada para outros tipos de tarefa. Para tarefas de regressão, por exemplo, pode-se substituir a função de Heaviside pela função-identidade (y = x) ou outra que retorne uma variável contínua, diferindo do análogo biológico.

De forma geral, para *n* entradas, o resultado do perceptron pode ser expresso na forma matricial y = H(WX), onde *W* é o vetor-linha dos pesos e *X*, o vetor-coluna das entradas. Um perceptron pode, portanto, representar qualquer função binária $y = H(f(x_1, ..., x_n))$ desde que *f* possa ser expressa como combinação linear das variáveis independentes. Em implementações reais em software, costuma-se adicionar um valor constante *b*, chamado de *bias* no jargão de aprendizado de máquina, assim fazendo y = H(WX + b). O valor *bias* é normalmente provido pelo usuário e vale zero por *default*.

O vetor de pesos *W* é obtido por um processo chamado *treinamento*, cujo algoritmo para o perceptron é:

i. Seja *D* o conjunto de treinamento constituído por pares em que cada *j*-ésimo par é formado por entradas X_j e uma saída d_j conhecida. *D* é fornecido pelo usuário (aprendizado supervisionado).

ii. Para cada X_j em *D*, calcular a saída do perceptron: $y_j = H(WX_j + b)$.

iii. Modificar os pesos em *W*: $w_i = w_i + r \cdot (d_j - y_j) \cdot x_{ij}$. Os novos pesos são imediatamente usados, ao invés de esperar que todos os pares de *D* sejam processados. *r* é chamado taxa de aprendizagem (*learning rate*), varia de 0,0 a 1,0 e é um parâmetro fornecido pelo usuário. Um valor maior de *r* resulta em uma convergência mais rápida ao custo de menor acurácia.

iv. Repete-se *ii* e *iii* até que uma condição seja satisfeita. Por exemplos: quando a média dos erros $|d_j - y_j|$ for abaixo de determinado limiar; quando um número de passos de treinamento (chamados de épocas ou *epochs* no jargão de aprendizado de máquina) for executado.

2.5.2 Redes neuronais[20]

Uma rede neuronal é a interconexão de mais de um neurônio artificial. Quando a tarefa de classificação, vista na Seção 2.5.1, tem mais de uma classe, monta-se uma rede neuronal como ilustrado na Figura 2-25. Nesse caso, a rede tem como saída quatro indicadores binários para quatro classes. Essa rede é capaz de aprender regras de classificação em quatro categorias a partir de duas variáveis de entrada. Pode-se facilmente empregar essa rede para aprender uma função (regressão) que relacione duas variáveis independentes e quadro variáveis de resposta, desde que se mude a função de ativação para uma função apropriada.

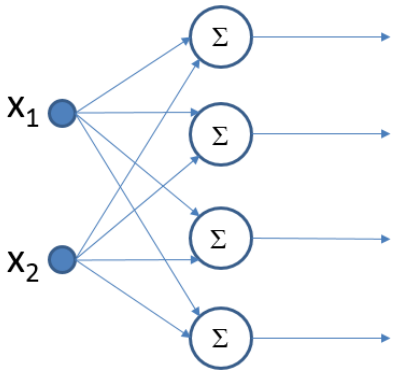


Figura 2-25: Uma rede neuronal composta por quatro perceptrons em uma única camada.

Naturalmente, pode-se conceber uma rede neuronal com arquitetura em duas camadas conforme mostrado na Figura 2-26 à esquerda. Como normalmente as redes são totalmente conectadas (todas as entradas estão conectadas a todos os neurônios) e como os números de entradas, de saídas e de neurônios são grandes na prática, adota-se, por brevidade, a representação da arquitetura da rede conforme mostrado na Figura 2-26 à direta. Nessa forma de representar, o número de saídas é implícito (equivale ao número de neurônios da última camada). O vetor de entrada não é considerado uma camada propriamente dita e normalmente é representada com um estilo diferente das camadas de neurônios.

---

[20] Nesta tese é utilizado o termo "neuronal", que deriva corretamente do vocábulo "neurônio" no lugar do termo mais difundido "neural", advindo diretamente do inglês, usado impropriamente como "relativo a neurônio". O termo "neuronial" (note-se a presença do "i") por vezes também é reconhecido em dicionários.

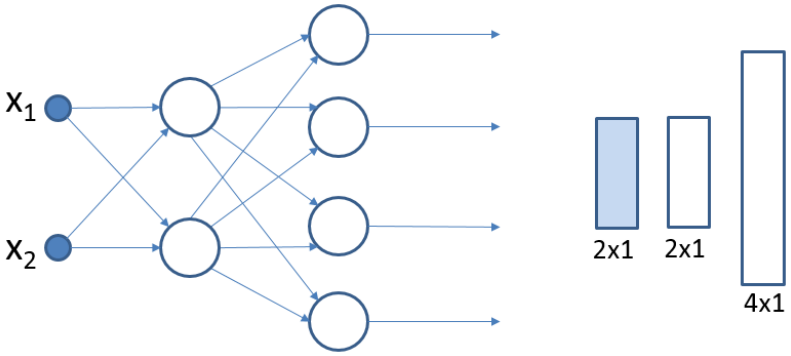


Figura 2-26: Uma rede neuronal de duas camadas com sua representação moderna à direita.

2.5.3 Redes neuronais profundas

Uma arquitetura de rede pode incluir ainda as chamadas *camadas ocultas*, ou seja, camadas de neurônios que não fazem contato nem com as entradas, nem com as saídas. O campo da Inteligência Artificial que estuda este tipo de arquitetura de rede é conhecido como Aprendizado Profundo (*Deep Learning*) e uma rede cuja arquitetura contenha camadas ocultas é chamada de rede neuronal profunda (*deep neural network*). Um exemplo de rede profunda está ilustrado na Figura 2-27. Uma rede profunda de perceptrons é conhecida como Perceptron Multicamada (*Multilayer Perceptron*, MLP), comumente referida por *rede MLP*. Segundo Ramachandran *et al.* (2017), uma função chamada ReLU (*Rectified Linear Unit*) é a função de ativação melhor sucedida e a mais popularmente empregada em redes profundas.

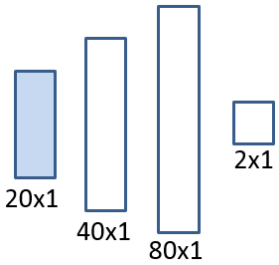


Figura 2-27: Uma rede neuronal profunda com uma camada oculta (80x1).

Por causa das camadas ocultas, o algoritmo de treinamento para uma rede MLP é diferente: o processo de retropropagação (*backpropagation* – Rumelhart *et al.*, 1986):

i. Para um *l*-ésimo neurônio de saída e para um *n*-ésimo dado de treinamento, calcular o erro da saída $e_l(n) = d(n) - y_l(n)$, onde *d* é o valor de resposta esperado e *y*, a resposta do neurônio.

ii. Calcular o erro quadrático total de todos os neurônios de saída para um *n*-ésimo dado de treinamento $\varepsilon(n) = \frac{1}{2} \Sigma e_l^2(n)$. Uma função como essa é chamada de função-perda

(*loss function*) segundo a praxe de aprendizado de máquina, sendo o mesmo que função-objetivo. Implementações em software podem empregar outras funções-perda diferentes do erro quadrático total.

iii. Atualizar o peso atual (w_{ij}^{0}) para a próxima época (w_{ij}^{1}) entre o *i*-ésimo neurônio da camada anterior e o *j*-ésimo neurônio da camada atual segundo a Equação 2-9, obtida desenvolvendo-se a partir da fórmula geral do gradiente descendente (Equação 2-1). Nessa equação, *r* é a taxa de aprendizagem, φ' é a derivada da função de ativação, v_j é o campo local induzido do *j*-ésimo neurônio da camada atual. O somatório itera sobre todos os *K* neurônios da camada anterior. y_i é a saída do *i*-ésimo neurônio da camada anterior. Na prática, as implementações em software usam um tipo de gradiente descendente estocástico, um processo semelhante ao SA (ver Seção 2.1.1), para que a otimização não fique presa em um mínimo local. O termo entre colchetes é efetivamente o gradiente utilizado no processo de otimização (ver Equação 2-1).

$$w_{ij}^{1}(n) = w_{ij}^{0}(n) + r\left[\phi'(v_j(n))\sum_{k=1}^{K} -\frac{\partial \varepsilon(n)}{\partial v_k(n)} w_{kj}(n)\right] y_i(n)$$

Equação 2-9: Atualização dos pesos de uma rede MLP (ver texto).

Aplicando-se uma rede MLP ao problema desta tese, pode-se definir uma arquitetura de rede (Figura 2-28) cujo objetivo seja aprender a relação entre as formas presentes em um mapa variográfico experimental e os parâmetros variográficos de um variograma teórico de até quatro estruturas imbricadas (dezesseis parâmetros no total). Como o mapa de teste tem 100×100 pixels, a camada de entrada deve ter 10.000 entradas para integrar toda a informação contida no mapa visando a obtenção dos 16 parâmetros variográficos (a última camada deve ter 16 neurônios). Estão também indicadas as funções de ativação (φ) para cada camada.

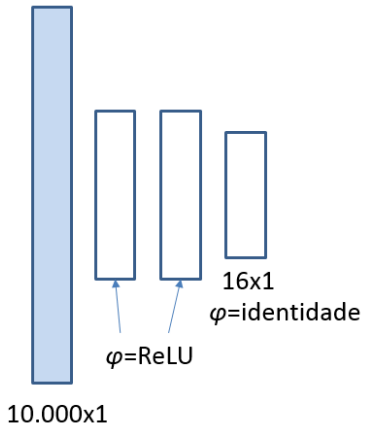


Figura 2-28: Uma possível arquitetura para uma rede MLP capaz de encontrar os parâmetros variográficos de um modelo que melhor se ajusta a um mapa variográfico experimental de 100x100 *pixels*. Ver texto.

Dependendo do tamanho da primeira camada, cria-se um problema combinatório. Se a primeira camada tiver 100 neurônios, serão 100 × 10.000 = 1.000.000 de pesos. A etapa de treinamento pode se tornar um sério problema de desempenho, sobretudo porque, para melhores resultados, é necessária uma grande quantidade de mapas para treinamento. Para 1.000 mapas de treinamento, seria necessário avaliar, para cada época, a Equação 2-9 1.000.000.000 de vezes. Um bilhão de cálculos não chega a ser um problema muito sério com o hardware de hoje, mas a equação da retropropagação contém um somatório que itera por *todos* os neurônios de uma camada. Entre a camada de entrada e a primeira camada, isso significa um bilhão vezes um milhão, o que é um número grande mesmo para os padrões de hoje.

2.5.4 Redes neuronais convolucionais profundas (CNN)

A CNN, outro mecanismo computacional bioinspirado, foi concebida a partir da investigação de Hubel e Wiesel (1968) acerca do funcionamento do córtex visual em mamíferos (notar esse paralelo com a análise de Gabor vista na Seção 2.2.5). Esse trabalho demonstra a existência de neurônios que respondem a estímulos limitados a campos visuais pequenos, que se conectam a outros neurônios. Esses neurônios respondem integrando a informação vinda de um pequeno grupo dos primeiros, formando elementos de imagem mais complexos. Esses neurônios, por sua vez, se conectam a outros de maior ordem ainda, estabelecendo uma hierarquia no processamento visual. Cada nível hierárquico forma elementos de maior complexidade até a interpretação completa da cena que está sendo captada pela retina. Em outras palavras, uma CNN estabelece uma hierarquia de feições em função da escala espacial, de forma semelhante à hierarquia resultante da WFT (ver Seções 2.2.6 e 3.6). Assim, se permite o leitor uma reflexão: o que nós vemos é um modelo construído pelo cérebro e não exatamente a cena opticamente projetada na retina, portanto, não confiemos tanto no sentido da visão.

Os principais elementos de uma CNN foram introduzidos no trabalho de Fukushima (1980) que chamou sua rede de *neocognitron*, a primeira CNN. Uma CNN, atualmente em voga no campo do aprendizado de máquina, é um constructo computacional composto por estruturas de dados e algoritmos arranjados para formar um *pipeline* de processamento (entrada → processamento → saída/entrada → processamento → ...). Esse *pipeline* pode assumir uma miríade de formas, a depender do objetivo do processamento. O arranjo de como as estruturas de dados e algoritmos estão conectados é denominado *arquitetura de rede*

(Kalogirou, 2014, cap. 11) – não confundir com a arquitetura de rede estudada em Telecomunicações (Tanembaum, 2003, Seção 1.2).

Por causa do problema de dimensionalidade (*curse of dimensionality*), as CNNs foram adotadas para tarefas que usem informações de imagens. Uma CNN é uma rede neuronal (normalmente profunda) em que as chamadas *camadas convolutivas* são inseridas entre as entradas e as camadas neuronais na arquitetura. Isto causa dois efeitos: integração da informação da imagem sem o problema combinatório e discriminação das estruturas espaciais na imagem, pois a operação de convolução, a depender do tamanho do núcleo (*kernel*) de convolução, só integra informações localmente.

Em rigor, a convolução é um operador integrativo entre duas funções, mas, no domínio discreto, procede-se como ilustrado na Figura 2-29. No exemplo, o valor do *pixel* de saída é $y = w_1x_{k+1} + w_2x_{k+2} + ... + w_9x_{k+9}$ onde $w_1...w_9$ são os pesos no núcleo de convolução e $x_{k+1}...x_{k+9}$ são valores em uma *k*-ésima vizinhança da imagem de entrada. Em notação matricial: $y = WX$ ($y = WX+b$ com o valor *bias*), sendo, portanto, matematicamente equivalente a um neurônio artificial. E, tal como o neurônio artificial, a convolução resulta em um único valor de saída.

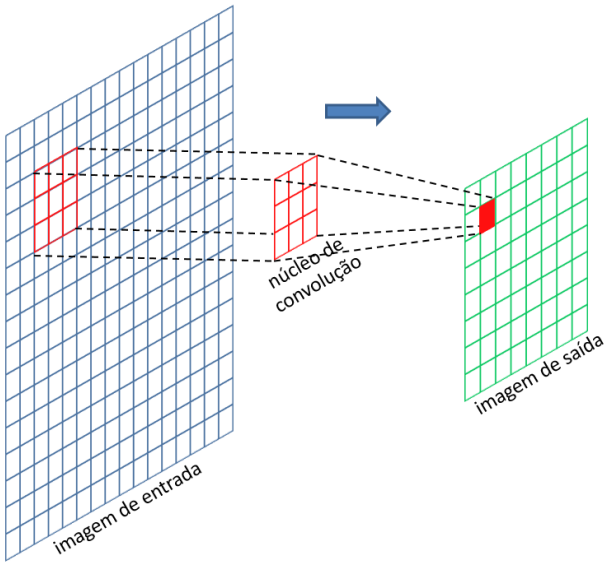


Figura 2-29: Ilustração do funcionamento de uma convolução de uma imagem de entrada com um núcleo de convolução resultando em outra imagem. Ver texto.

A imagem de entrada é, então, percorrida conforme ilustrado na Figura 2-30. A passada da varredura é configurável pelo usuário, cujo valor é conhecido por *stride*. O tamanho do *stride*, portanto, define o tamanho da imagem de saída. A configuração de *padding*, isto é, dilatar a imagem de entrada preenchendo as células extras com algum valor (*zero padding*, por exemplo,

as preenche com zeros) também determina o tamanho final da imagem de saída. Em implementações para aplicação industrial, o processamento da convolução não ocorre de maneira sequencial, mas massivamente paralela no processador gráfico (GPU) ou com hardware dedicado de processamento tensorial (TPU) desenvolvido pela Google.

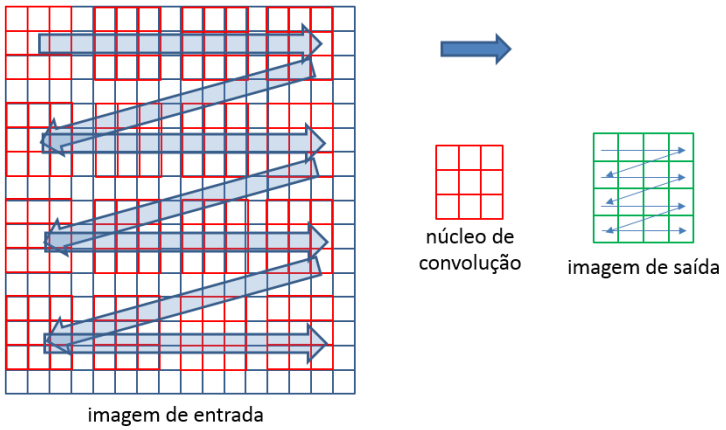


Figura 2-30: Varredura de convolução com passada (*stride*) de 5x5.

Sendo o núcleo de convolução equivalente a um neurônio artificial, pode-se definir uma camada convolutiva com um determinado número de núcleos. O número de núcleos define o número de imagens de saída (chamadas de *feature maps*), ou seja, a profundidade do volume de saída. O valor do *stride* e do *padding* definem a largura e altura desse volume. Coletivamente, esses três valores são conhecidos como hiperparâmetros (*hiperparameters*). O tamanho das imagens de saída em número de células, ao longo da largura ou da altura, pode ser estimado com o cálculo simples: $n_S = (n_E + padding * 2) / stride + 1$ onde n_E é o número de células da imagem de entrada ao longo da altura ou da largura. A Figura 2-31 ilustra uma camada convolutiva com quatro núcleos, fazendo o papel de quatro neurônios artificiais.

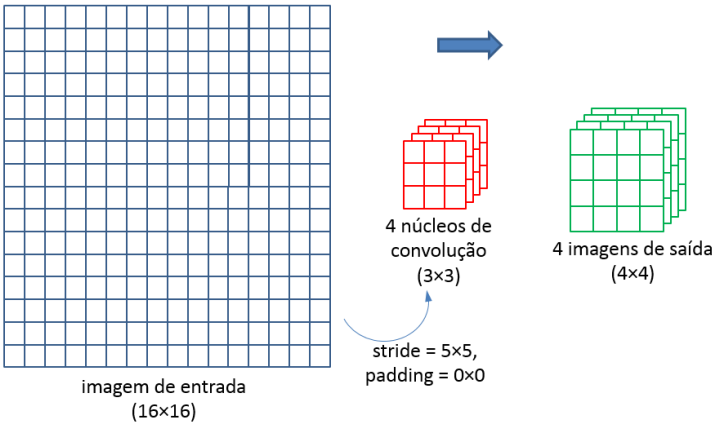


Figura 2-31: Uma camada convolutiva de quatro núcleos (em vermelho).

Tal como uma MLP, uma CNN pode ter múltiplas camadas convolutivas. Assim, entre uma camada e outra, os núcleos passam a ser tensores (matrizes com mais do que apenas linhas e colunas), cada qual com a dimensão ao longo da profundidade equivalente ao número de imagens na entrada da camada. Núcleos em forma de tensor integram a informação contida nas múltiplas imagens de entrada em uma imagem apenas.

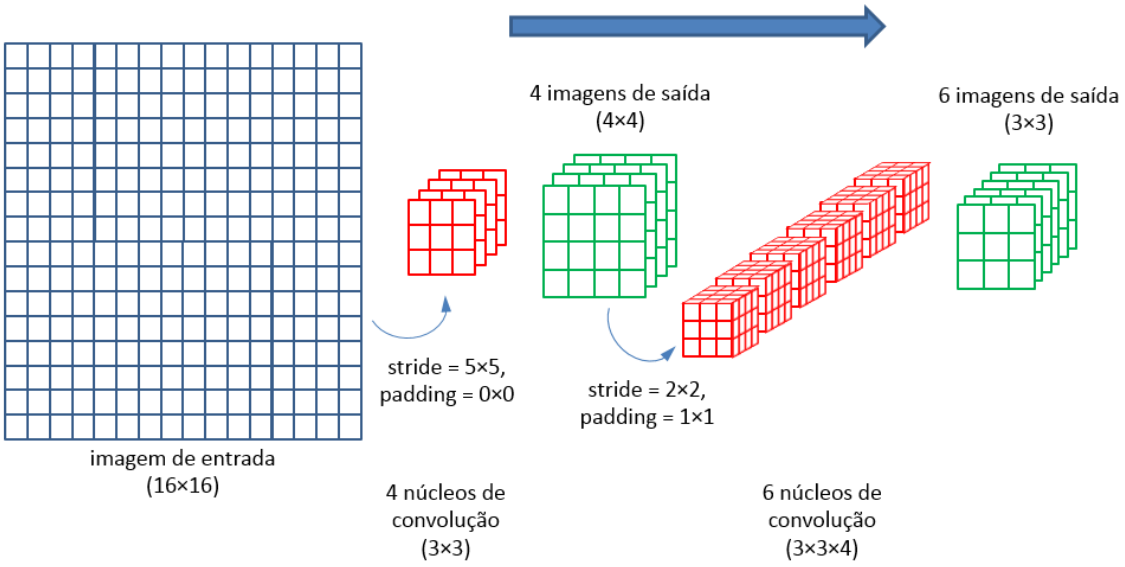


Figura 2-32: Uma CNN com duas camadas convolutivas.

Assim como no caso das MLPs, por brevidade, a arquitetura das camadas convolutivas costuma ser representada, com variações, conforme mostrado na Figura 2-33. Essa arquitetura é a da CNN ilustrada na Figura 2-32. Essa representação destaca como a informação da imagem de entrada está sendo canalizada pela rede. A largura e altura dos tensores (paralelepípedos) é tão maior quanto a quantidade de informação espacial (resolução) na camada. A profundidade dos tensores é tão maior quanto a quantidade de feições (*features*) encontradas na camada.

Etapas não-convolutivas podem ser inseridas em uma camada, por exemplo, uma de máximo de janela móvel (*max pooling*), de uso bastante difundido. A entrada e as etapas com funções diferentes de convolução costumam ser representadas com cores diferentes para destacar quais camadas são "inteligentes", ou seja, aquelas que se envolvem com o processo de aprendizado com os pesos. As camadas devem ser contadas apenas quando há fluxo de pesos entre elas (Figura 2-34), pois é uma decorrência do algoritmo de aprendizado. Por exemplo, uma convolução seguida de um *max pooling* contam como uma camada só.

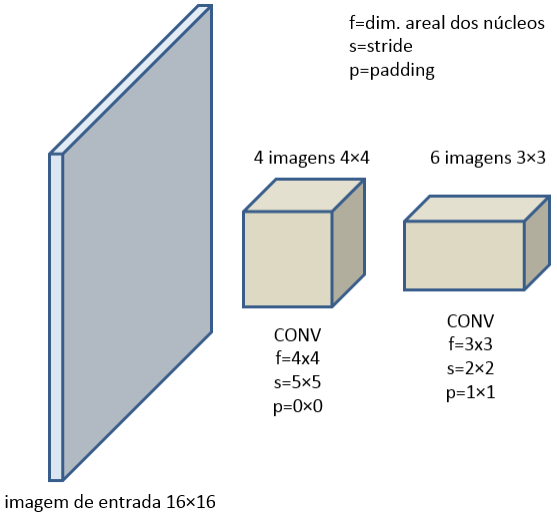


Figura 2-33: Arquitetura das camadas convolutivas da CNN da Figura 2-32.

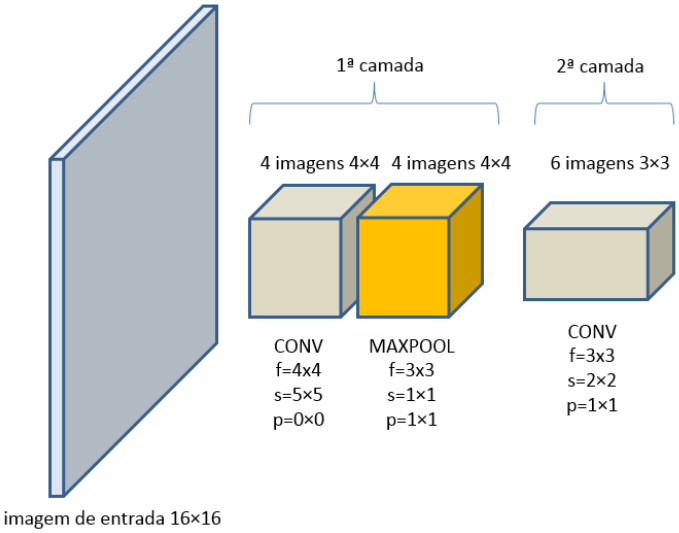


Figura 2-34: Uma arquitetura de CNN de duas camadas. A primeira camada tem uma etapa de pós-processamento do tipo máximo de janela móvel (*max pooling*).

Por fim, com a parte convolucional da arquitetura tendo reduzido a dimensionalidade o suficiente, conecta-se a parte neuronal da arquitetura para o "alto raciocínio" que uma rede totalmente conectada proporciona. A conexão entre a última camada convolucional e a primeira camada neuronal dá-se por meio de uma "camada" de linearização (*flattening*) que consiste simplesmente em serializar todas as células do último volume de mapas de saída de forma que elas sirvam como entradas individuais para uma MLP. O que uma CNN aprende é o conjunto de filtros (núcleos de convolução) capaz de decompor uma imagem em elementos de geometria da mais simples à mais complexa.

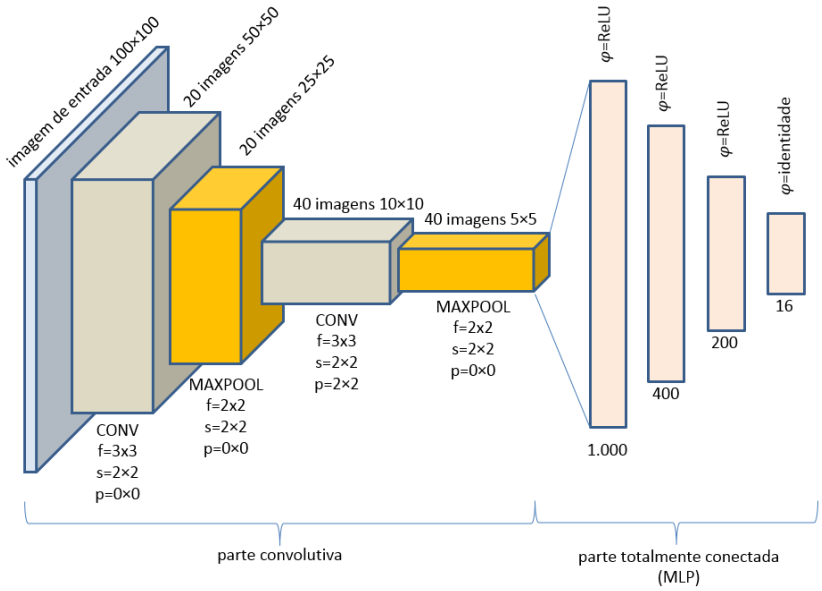


Figura 2-35: Uma arquitetura completa de CNN para obtenção de 16 parâmetros variográficos de um modelo que melhor se ajusta a um mapa variográfico experimental de 100×100 pixels.

O treino, ou seja, a atualização dos pesos dos núcleos de convolução, se dá de forma semelhante ao já visto para o treinamento de uma rede MLP. Em princípio, a arquitetura proposta pode aprender a relação entre as formas observadas em um mapa variográfico e os parâmetros do variograma teórico. O conjunto de treinamento pode ser construído ao gerar imagens de superfícies variográficas a partir de parâmetros variográficos aleatórios.

2.5.5 Redes autocodificadoras ou totalmente convolucionais

Uma CNN autocodificadora (*autoencoder*) ou totalmente convolucional (Fully Convolutional Network – FCN – Long *et al.*, 2014) é composta por uma parte convolutiva convencional (parte codificadora) conectada a uma parte convolutiva especial (*up convolution*) que aumenta a contagem de pixels da imagem em sua saída (parte decodificadora). A forma geral de arquitetura está ilustrada na Figura 2-36. Uma característica importante desse tipo de CNN é que a saída deve ter a mesma topologia da entrada, por exemplo, se a entrada é uma imagem monocanal de 100×100 pixels, a saída deve ser também monocanal de 100×100 pixels, embora as camadas ocultas não precisem ser simétricas. A razão disso é que uma CNN autocodificadora, como o nome sugere, aprende por autossupervisão.

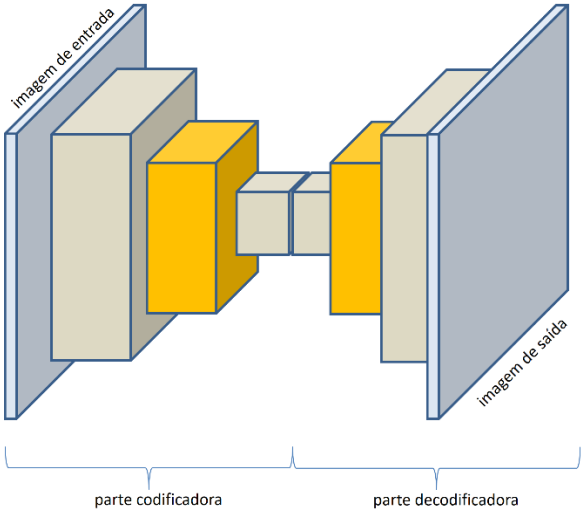


Figura 2-36: Forma geral de arquitetura de uma CNN autocodificadora.

Foram abordados até aqui exemplos de redes que aprendem por supervisão em que o usuário fornece um conjunto de mapas associados a parâmetros variográficos tidos como corretos *a priori*. No aprendizado autossupervisionado, o usuário fornece um número apropriado de imagens e a rede aprende reduzindo a diferença entre as imagens originais e as resultantes da rede em cada época. A ideia é que, ao final do treinamento, os pesos da parte convolutiva formem um codificador (parametrizador) de imagens de determinada característica. Mapas variográficos podem ser vistos como imagens que têm características em comum. Lembrar que aprendizado autossupervisionado não é o mesmo que aprendizado não-supervisionado.

A saída da parte convolutiva, se adequadamente treinada, fornece então uma forma compactada da entrada. Hinton e Salakhutdinov (2006) demonstram que uma rede autocodificadora é uma generalização não-linear da Análise de Componentes Principais (PCA), sendo, portanto, uma transformação redutora de dimensionalidade e reversível. Após o treinamento, a parte decodificadora pode ser substituída por uma parte neuronal para o aprendizado supervisionado convencional da relação entre parâmetros de imagem e parâmetros variográficos.

2.5.6 Redes U

Uma rede U (Ronneberger *et al.* 2015) é uma FCN com canais de informação entre camadas que normalmente não fariam contato. A ideia é transmitir informação de feições de maior resolução para a parte decodificadora da arquitetura, pois essa aumenta a definição das imagens (*upscaling*) mas não a resolução. Lembrando que definição se refere à contagem de pixels de uma imagem enquanto que resolução é relativa à informação que ela transporta, ou seja, uma imagem de alta definição não necessariamente contém informação em alta

resolução. A Figura 2-37 mostra a forma geral de uma rede U. O nome U se deve à forma da arquitetura, que tende a se parecer com a letra "U".

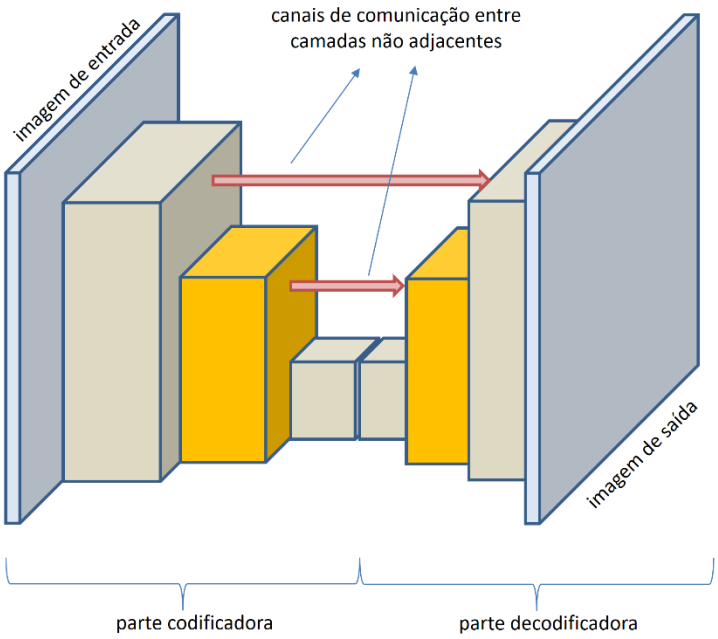


Figura 2-37: Forma geral de arquitetura de uma rede U.

2.5.7 *Data augmentation*

Data augmentation é uma técnica popular em aprendizado de máquina para que as redes não sejam sensíveis a determinado parâmetro dos objetos que se pretende reconhecer, por exemplo orientação e escala. *Data augmentation* consiste simplesmente em gerar dados a partir de um mesmo original, através da variação aleatória de um ou mais parâmetros. A Figura 2-38 ilustra o processo.

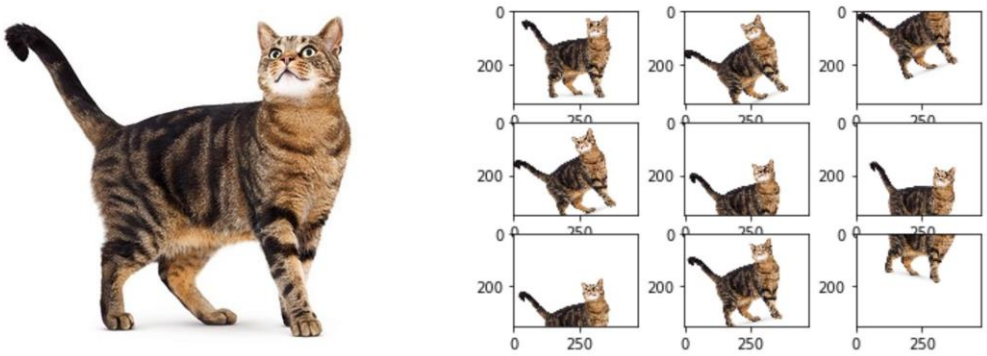


Figura 2-38: Exemplo de *data augmentation*. A imagem da esquerda é a imagem original. As imagens da direita são novos dados sintetizados a partir do original aplicando, sobre este, variações aleatórias na orientação e deslocamento. Retirado de [21]

O objetivo é treinar a rede com uma grande quantidade de dados de forma que seu desempenho seja robusto. Entretanto, Laptev *et al.* (2016) argumentam que o tempo de treinamento aumenta substancialmente com ainda o risco de *underfitting* (pouca acurácia nas predições) ou de *overfitting* (a rede, ao invés de aprender, "decora" os dados e não faz predições satisfatórias).

2.5.8 CNN de Gabor (GCN)

Conforme visto nas Seções 2.2.5 e 2.5.4, tanto a análise com filtro de Gabor quanto a CNN são modelos de como o córtex visual de mamíferos funciona. Segundo Luan *et al.* (2018), obter robustez das CNNs quanto à escala e orientação requer técnicas avançadas que atualmente têm algum comprometimento: i) resultam em grande tempo de processamento; ii) têm limitações quanto à complexidade das transformações que uma imagem possa ter.

Uma GCN (Luan *et al.*, 2018) é uma CNN que aprende núcleos de Gabor (Figura 2-17) ou núcleos modulados (multiplicados) por núcleos de Gabor ao invés de núcleos de convolução arbitrários. A ideia da camada modulada por Gabor é "explodir" os mapas de feições em suas componentes estruturais, tal como mostrado na Figura 1-8. Em tese, com as componentes não mais sobrepostas, a camada teria melhor eficácia, mas ao custo de uma dimensão extra. Assim, a rede pode ter um treinamento robusto quanto à orientação e escala das feições com menos imagens de treinamento e/ou com camadas convolutivas mais finas,

[21] https://stackoverflow.com/questions/44819922/data-augmentation-is-shifting-needed (acessado em 25/12/2019)

resultando em menores tempo de execução e uso de memória ao custo de uma pequena penalidade na acurácia (Sarwar *et al.,* 2017).

No entanto, há um segundo custo, computacional, assaz impactante. Esse custo pode ser visualizado na Figura 2-39. Nesse exemplo, variou-se somente o azimute em apenas quatro ângulos, resultando em um número quatro vezes maior de filtros para cada filtro aprendido. Gerando núcleos Gabor ao variar o azimute e a escala tal como descrito no experimento apresentado na Seção 3.5.3, chega-se à cifra de 62.640 filtros modulados para cada filtro aprendido.

Figura 2-39: Modulação de filtros aprendidos pelo processo normal (retropropagação) com núcleos de Gabor para formar Filtros de Orientação Gabor (GoF). Retirado de Luan *et al.* (2018), fig. 2.

Para contornar este problema, pode-se aprender os parâmetros dos núcleos de Gabor diretamente, ao invés de modular os núcleos aprendidos por todos os possíveis núcleos de Gabor. Para isso, um novo tipo de camada convolutiva é apresentado e proposto na Seção 4.8, com implementação.

2.5.9 Os conjuntos de treinamento, teste e validação

Os algoritmos de treinamento requerem o cômputo de métricas de incerteza e de acurácia para que o usuário possa julgar se o treinamento foi insuficiente, suficiente ou em excesso. O conjunto de treinamento (*training set*) já foi apresentado nas seções anteriores. O conjunto de validação (*validation set*) é usado em cada passo (época) para avaliar a função-objetivo e, com isso, controlar a convergência. O conjunto de teste (*test set*) é usado pelo algoritmo para testar a acurácia da predição durante o treinamento ao apresentar as entradas e comparar as predições da rede com as saídas tidas como verdadeiras *a priori*.

2.6 Métodos de análise de agrupamento (*clustering analysis*)

Métodos de análise de agrupamento são usados para encontrar grupos (*clusters*) de dados segundo um critério de semelhança como, por exemplo, a distância Euclidiana. Por exemplo, amostras de rocha que exibam medidas de porosidade e de permeabilidade semelhantes indicam que podem ser do mesmo tipo de rocha. Outros critérios podem ser usados, tais como mínima variância intragrupo e máxima variância intergrupo (Figura 2-40).

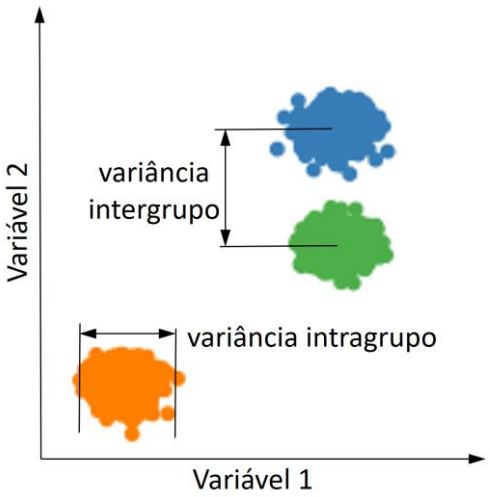


Figura 2-40: Um *cross plot* entre duas variáveis contendo três agrupamentos (*clusters*) ilustrando os conceitos de variâncias inter- e intragrupo.

Junto com as redes neuronais, os algoritmos de agrupamento são componentes importantes em tarefas que envolvem aprendizado de máquina. As seções seguintes apresentam os algoritmos de agrupamento considerados para fazer parte do método proposto nesta tese.

2.6.1 Tipos de agrupamento (*clusters*)

Antes de apresentar os algoritmos de agrupamento investigados, convém apresentar brevemente os tipos de agrupamentos que costumam surgir com mais frequência. A Figura 2-41 ilustra os principais tipos que podem ocorrer. (a) são dois agrupamentos cujos centros são muito próximos; (b) são dois agrupamentos chamados não convexos, ou seja, o centro está fora do agrupamento; (c) são três agrupamentos com variâncias (tamanhos) diferentes; (d) são três agrupamentos de mesma variância e (e) são três agrupamentos de mesma variância, mas anisotrópicos. Ainda: (a) e (b) são ditos agrupamentos com geometria não-linear; (c), (d) e (e) são ditos agrupamentos com geometria linear. Por fim: (a) e (b) são agrupamentos que podem ser separados por algoritmos que usem a conectividade (ex.: vizinho mais próximo) como critério; (c), (d) e (e) são agrupamentos que podem ser separados por algoritmos que usem compacticidade (ex.: variância em torno de uma média) como critério.

A compreensão dos caráteres que os agrupamentos podem assumir é importante para a seleção de um algoritmo de agrupamento adequado. Por exemplo, algoritmos que se baseiam no centro para diferenciar os agrupamentos, não funcionarão bem nos casos (a) e (b).

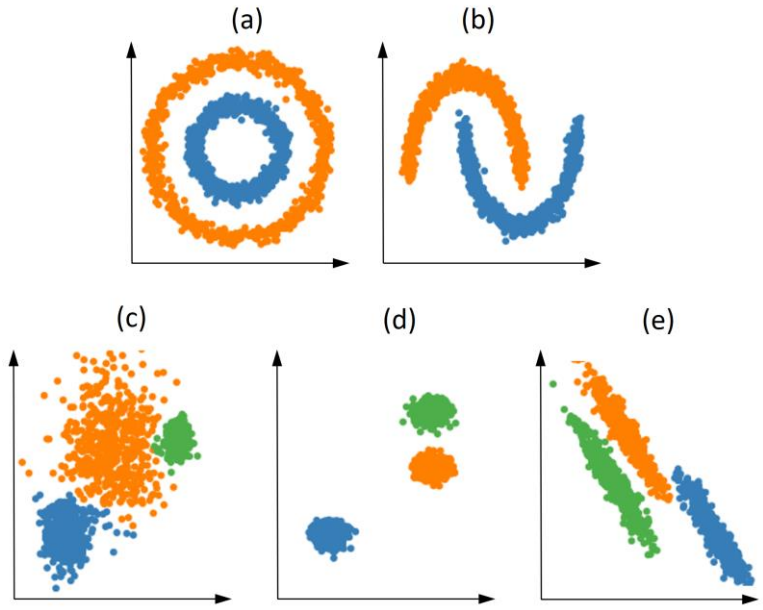


Figura 2-41: Tipos de agrupamento representados graficamente como *cross plots*. Ver texto.

2.6.2 K-means

K-means (Forgy, 1965), por vezes conhecido por algoritmo de Lloyd, poderia ser traduzido como "k médias". Tem esse nome devido ao parâmetro *k* que é o número de grupos que o usuário deseja encontrar. E os grupos são definidos pelos seus centros (médias). Trata-se de um algoritmo que busca minimizar as variâncias intragrupo. O número de grupos é fixo e dado pelo usuário, isto é, não consegue aprender o número de grupos ou reconhecer um número de grupos diferente do informado.

O K-means funciona bem com grande quantidade de amostras, mas é contraindicado quando o número de dimensões é alto. O uso de médias implica que se trata de um algoritmo orientado a compacticidade cujo critério de semelhança é a distância Euclidiana (Equação 2-10), que tende a inflacionar com número alto de dimensões. O K-means também é contraindicado quando os agrupamentos variam muito em tamanho (variância intragrupo).

$$d_{E01} = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$$

Equação 2-10: Distância Euclidiana em três dimensões entre dois pontos: (x_0, y_0, z_0) e (x_1, y_1, z_1).

K-means possui variações, mas a forma geral do algoritmo está descrita a seguir:

i. Escolher *k* amostras aleatoriamente. Elas serão os *k* centros iniciais que representam os *k* agrupamentos.

ii. Para cada amostra, determinar qual o centro mais próximo. Marcar a amostra como sendo do agrupamento desse centro.

iii. Para cada um dos *k* agrupamentos, calcular a média das amostras marcadas como sendo dele. Essa média é o novo centro representativo do agrupamento.

iv. Para cada um dos *k* agrupamentos, calcular a diferença entre o novo e o velho centros.

v. Se todas as *k* diferenças forem menores que uma tolerância, terminar o processamento ou voltar ao passo ii) caso contrário.

O resultado do algoritmo é dependente dos centroides iniciais, dado que eles podem se dirigir para locais diferentes dependendo dos locais de partida. Esse comportamento se assemelha ao do Gradiente Descendente (Seção 2.1.1).

2.6.3 Agrupamento espectral

Agrupamento espectral (*Spectral Clustering* – Ng *et al.*, 2002) tem esse nome porque na verdade é um macro-algoritmo que emprega uma decomposição de matriz em autovalores e autovetores (conhecida por decomposição espectral) para realizar uma redução dimensional e, depois, chamar um algoritmo de agrupamento propriamente dito em um espaço de menor dimensão. A matriz em questão é uma matriz de semelhança ou de afinidade *A* das amostras. A matriz de adjacência, por exemplo, é uma matriz de semelhança quadrada cujas linhas e colunas são em quantidade igual à de amostras e suas células contêm 1 para uma conexão entre dois pontos (como em um grafo ou um agrupamento) e 0 para não. Naturalmente, todos os elementos da diagonal principal de *A* são iguais a zero.

A forma geral do macro-algoritmo é:

i. Calcular os elementos de *A*: $A_{ij} = \exp(-\alpha \cdot d_{Eij}^2)$, onde α é um parâmetro dado pelo usuário e d_{Eij} é a distância Euclidiana (Equação 2-10) entre as *i*-ésima e *j*-ésima amostras. exp() é a função exponencial (e^x). Um ponto de corte r_{cutoff} pode ser usado tal que $A_{ij} = 0$ se $d_{Eij}^2 > r_{cutoff}$. Assim, *A* conterá elementos tendendo a 1 para pares de amostras próximas entre si e tendendo a 0 para pares afastados. A razão para usar $\exp(-\alpha \cdot d_{Eij}^2)$, chamado de núcleo Gaussiano (*Gaussian kernel*), é mitigar o impacto das irregularidades em amostras (*outliers*) em aplicações do mundo real, tal é o caso do pacote SciKit-Learn empregado nesta tese.

ii. Calcular uma matriz-diagonal, *D*, chamada de matriz de grau: $D_{ii} = \sum_{j=1}^{n} A_{ij}$. Os demais elementos de *D* são iguais a zero. Grau aqui é um termo que alude à Teoria de Grafos, que significa a quantos nós um determinado nó está conectado. Assim, como os elementos de *A* tendem a 0 ou a 1, somando elementos ao longo das linhas

e colocando resultado na diagonal principal, D_{ii} pode ser interpretado como uma aproximação de quantas amostras estão próximas a uma *i*-ésima amostra. Portanto, *D* pode ser usado como um fator de normalização para *A* no sentido de equilibrar agrupamentos com números diferentes de elementos.

iii. Definir uma matriz Laplaciana *L* (ver Teoria de Grafos). A literatura dá conta de formas diferentes de defini-la: a Laplaciana propriamente dita: L = D – A; a Laplaciana normalizada: $L_N = \sqrt{D^{-1}}L\sqrt{D^{-1}}$; a Laplaciana generalizada: $L_G = D^{-1}L$; a Laplaciana relaxada: $L_\varrho = L - \varrho D$, onde ϱ é um coeficiente dado pelo usuário; e outras. O importante é que, se as amostras formarem grupos de fato, *L* terá blocos de valores diferentes de zero ao longo de sua diagonal principal. Esses blocos indicam os agrupamentos, fazendo o algoritmo orientado a conectividade.

iv. Encontrar os autovetores, v, e autovalores, λ, de *L*.

v. Selecionar os k autovetores correspondentes aos k menores ou maiores autovalores. k é o número de agrupamentos que o usuário deseja encontrar. Esses k autovetores são as colunas de uma matriz *P*.

vi. Reduzir a dimensão das amostras, cada qual com, por exemplo, 6 medidas, para, digamos, $k = 3$, multiplicando cada vetor-coluna amostral pela transformação P^TLP.

vii. Realizar a separação dos agrupamentos nesse novo espaço de menor dimensão com um algoritmo de análise de agrupamento propriamente dito (K-means, por exemplo). O efeito dessa redução dimensional é mitigar a influência de ruído e de *outliers* caso a análise de agrupamento fosse feita em espaço com todas as dimensões.

2.6.4 Propagação de afinidade

Propagação de afinidade (Affinity Propagation – Frey e Dueck, 2007) é um algoritmo de agrupamento que consiste em um mecanismo de troca de informações (daí o nome do algoritmo) entre os diversos pontos que representam as amostras. Assim, é um algoritmo que realiza o agrupamento pelo critério de conectividade, podendo resolver grupos com centros coincidentes ou de geometria complexa com os da Figura 2-41, (a) e (b). O estabelecimento de uma relação entre as amostras tais como se estivessem em rede para propagação de mensagens cria uma demanda de memória proporcional ao quadrado do número de amostras. A troca de informações entre todos os pontos que representam as amostras implica que o tempo de execução também é quadrático, por uma questão

combinatória. Consequentemente, o algoritmo não é recomendado para um grande número de amostras.

Eis o algoritmo:

i. Inicializar a matriz de similaridade *S* entre todos os pares de amostras. Os elementos de *S* podem ser as distâncias Euclidianas (Equação 2-10): $S_{ij} = d_{Eij}$.

ii. Inicializar as matrizes de responsabilidade *R* e de disponibilidade *A* com todos os elementos iguais a zero.

iii. (Re-)Calcular *R* entre todos os pares de amostras: $R_{ij} = S_{ij} - \max[(A_{ik} + S_{ik}) \; \forall \; k \neq j]$.

iv. (Re-)Calcular *A* entre todos os pares de amostras: $A_{ij} = \min[0, R_{jj} + \sum_{k=1}^{n, \notin \{i,j\}} R_{kj}]$. Notar a interdependência entre os cálculos de *R* e de *A*, pois é o mecanismo de propagação de informações a que alude o nome do algoritmo.

v. Atualizar *R* da iteração atual *t* com o coeficiente de amortecimento λ configurável pelo usuário e que varia entre 0,0 e 1,0: $R_{ij}^{t} = \lambda R_{ij}^{t-1} + (1-\lambda)R_{ji}^{t}$. R^{t-1} é *R* da iteração anterior. O amortecimento evita oscilações numéricas durante a troca de informações de uma iteração para outra.

vi. Atualizar *A* da iteração atual *t* com o fator de amortecimento λ: $A_{ij}^{t} = \lambda A_{ij}^{t-1} + (1-\lambda)A_{ji}^{t}$. A^{t-1} é *A* da iteração anterior.

vii. Para cada amostra *x*, definir o índice da amostra que a representa: $\varphi(x_i) = \arg\max_k\{R_{ik} + A_{ik}\}$. O operador *arg max k*{} quer dizer "o índice *k* que corresponde ao maior valor de". Uma amostra que representa outras se chama exemplar. Naturalmente, espera-se que o número de exemplares seja pequeno e todas as amostras que se referem a um mesmo exemplar formam um agrupamento. Assim, o algoritmo é capaz de determinar o número de agrupamentos de forma automática.

viii. Se os exemplares não mudarem em relação à iteração anterior, parar; caso não, voltar ao passo iii).

Como os exemplares tendem a ficar no centro dos agrupamentos, o algoritmo é orientado a compacticidade.

2.6.5 Deslocamento da média

Deslocamento da média (Mean Shift – Cheng, 1995), como o nome implica, é um algoritmo de agrupamento orientado a compacticidade como o K-means, mas com a vantagem de determinar o número de agrupamentos automaticamente.

Eis o algoritmo:

i. Inicializar um certo número de centros c_i.

ii. Para cada centro c_i, calcular a média ponderada $m(c_i)$, dada pela Equação 2-11. $N(c_i)$ contém todas as amostras x_j da vizinhança no entorno do centro c_i. *K* é tipicamente o núcleo (*kernel*) Gaussiano (ver Seção 2.6.3, alínea i.). A diferença $m(c_i) - c_i$ é o deslocamento do centro c_i em relação à média de sua vizinhança, daí o nome do algoritmo.

$$m(c_i) = \frac{\sum_{x_j \in N(c_i)} K(x_j - c_i) x_j}{\sum_{x_j \in N(c_i)} K(x_j - c_i)}$$

Equação 2-11: Cálculo da média ponderada na vizinhança de um centro candidato. Ver texto.

iii. Atualizar os centros para a iteração: $c_i = m(c_i)$.

iv. Descartar os centros que estiverem muito perto de outro (redundantes).

v. Parar se os deslocamentos $m(c_i) - c_i$ forem desprezíveis ou voltar a ii) caso não.

2.6.6 Agrupamento por aglomeração

Agrupamento por aglomeração (Zhang *et al.*, 2012) é um tipo de algoritmo de agrupamento hierárquico (o outro sendo Agrupamento por partição). Aglomeração alude à estratégia de baixo-para-cima do algoritmo em que cada amostra começa como sendo um agrupamento de uma única amostra. A cada iteração, o algoritmo junta pares de agrupamentos que sejam próximos segundo uma métrica de distância. A informação de quais agrupamentos deram origem a outro mais abrangente são retidas para formar uma estrutura hierárquica em árvore. As iterações terminam quando restar apenas um agrupamento global, representado pelo nó-raíz da árvore. O critério para juntar dois agrupamentos se chama critério de ligação (*linkage*) e é determinante na estrutura final da árvore que, em última análise, determinará os agrupamentos. Um critério famoso e empregado com frequência é o critério de Ward (Ward, 1963), que busca minimizar a variância intragrupo. Naturalmente, este é um algoritmo orientado a conectividade.

Uma vez construída a árvore hierárquica, obtém-se os agrupamentos "cortando"-se a árvore em um determinado nível. Na Figura 2-42, o corte (linha tracejada) formou duas subárvores cujos nós-raízes são os dois agrupamentos representados no *cross plot* à esquerda por elipses tracejadas. O corte na árvore pode ser por um critério como número de nós-raízes ou limiar (*threshold*) de distância.

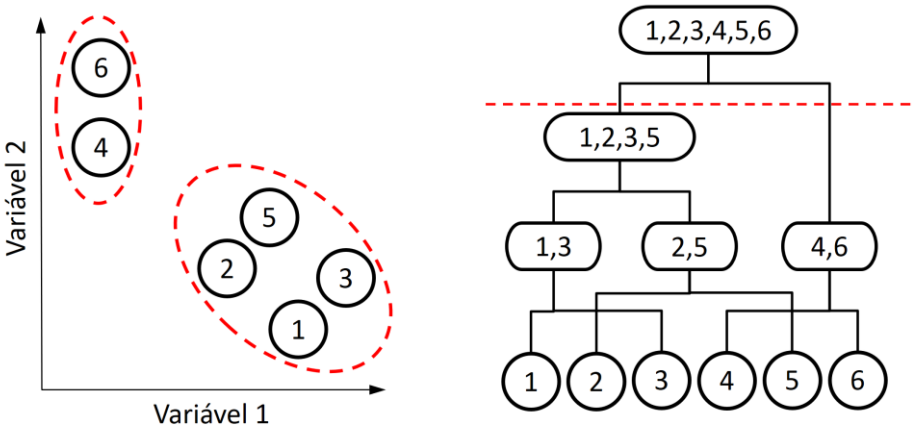


Figura 2-42: Ilustração de funcionamento do agrupamento por aglomeração. Ver texto.

2.6.7 DBSCAN

Agrupamento Espacial Baseado em Densidade para Aplicações com Ruído ou, no inglês, *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN – Ester *et al.*, 1996) é um algoritmo pautado no conceito de amostras centrais (*core samples*). O seu comportamento é fortemente sensível a dois parâmetros: n_{MIN}, que é o número mínimo de amostras que deve haver até uma distância d_{MAX} para que uma amostra possa ser considerada central. O algoritmo determina automaticamente a quantidade de agrupamentos. O DBSCAN também é capaz de descartar automaticamente amostras que são ruído (*outliers*), uma característica que o distingue de outros algoritmos de agrupamento.

Eis o algoritmo:

i. Percorrer as amostras até encontrar uma que seja central, isto é, que tenha ao menos n_{MIN} amostras dentro da distância d_{MAX}. A busca espacial de amostras vizinhas é normalmente feita com um índice espacial rápido como árvore R para evitar o recálculo de uma matriz de distâncias completa, o que seria dispendioso.

ii. Arrolar todas as amostras encontradas no entorno da amostra central.

iii. Para o conjunto de amostras obtidas no passo ii), recursivamente realizar os passos i), ii) e este. Todas as amostras, centrais ou não, encontradas na mesma recursão são marcadas como sendo de um mesmo agrupamento. As recursões eventualmente cessam quando alcançarem todas as amostras não centrais na periferia do agrupamento. Esse comportamento faz do DBSCAN um algoritmo de agrupamento orientado a conectividade.

iv. Iniciar outro cluster, voltando ao passo i) e considerando todas as amostras ainda não marcadas. Se não forem encontradas mais amostras centrais, as amostras que ainda remanescerem como não marcadas são então marcadas como sendo ruído e o algoritmo termina.

2.6.8 BIRCH

Agrupamento Hierárquico Balanceado Iterativo ou, no inglês, *Balanced Iterative Reducing and Clustering using Hierarchies* (BIRCH – Zhang *et al.* 1996) é um algoritmo de agrupamento do tipo hierárquico. Mas, diferentemente dos algoritmos hierárquicos clássicos (apresentados na Seção 2.6.6), constrói sua árvore iterativamente, isto é, a estrutura da árvore pode mudar a cada amostra adicionada, resultando em uma árvore final balanceada.

BIRCH é na verdade um algoritmo de construção de árvore balanceada em altura genérico, mas com um critério adicional que determina a divisão de um nó: o diâmetro do agrupamento representado pelas amostras ou pelos agrupamentos-filhos. Assim, o BIRCH é um algoritmo de agrupamento orientado a compacticidade. Um algoritmo de construção de árvores balanceadas tem muito detalhes que ultrapassam o escopo desta tese, assim, caso o leitor deseje conhecê-lo, recomenda-se Knuth (1998, Seção 6.2.3) e Cormen *et al.* (2009, Cap. 12).

Capítulo 3

Decomposição manual

Este capítulo apresenta os resultados de decomposições estruturais do dado de teste apresentado na Seção 1.5 com diversos algoritmos úteis a esse propósito e com parametrização manual. Ou seja, a tarefa de encontrar os parâmetros ótimos dos algoritmos e julgar os resultados recai sobre o modelador humano. Os algoritmos utilizados nos experimentos estão apresentados na Seção 2.2 e implementados no software GammaRay (ver Seção 1.5).

3.1 FK com variograma global

Como o variograma teórico do mapa é conhecido *a priori* (Tabela 1-1), é relativamente fácil empregar FK para decompô-lo em fatores geomorfológicos perfeitamente distintos. Normalmente, é necessário esforço considerável para modelar o variograma teórico a partir dos dados para obter uma decomposição satisfatória, principalmente se houver muitas estruturas imbricadas.

Foram usadas as seguintes estratégias de busca de amostras:

- Fator 2: elipse 20m × 20m; limitado a 32 amostras; espaçamento de 5m entre amostras.
- Fator 3: elipse 50m × 50m; limitado a 32 amostras; espaçamento de 11m entre amostras.
- Fator 4: elipse 5m × 5m; limitado a 32 amostras.
- Fator 5: elipse 5m × 5m; limitado a 32 amostras.

Para não haver um quinto fator (a média), a opção de krigagem é krigagem simples com média zero. Os quatro fatores geomorfológicos estão mostrados na Figura 3-1. A escala de cores em comum é a do mapa original (Figura 1-10), com que se constata visualmente que cada um dos fatores têm cerca de 25% da informação, tal como modelado no variograma. Os fatores 3 e 4 foram submetidos a um filtro de média móvel com uma pequena janela de

3 × 3 células para remoção de artefatos introduzidos pelo uso de vizinhança de busca. Esses fatores serão usados como referência para avaliar o desempenho do método proposto nesta tese.

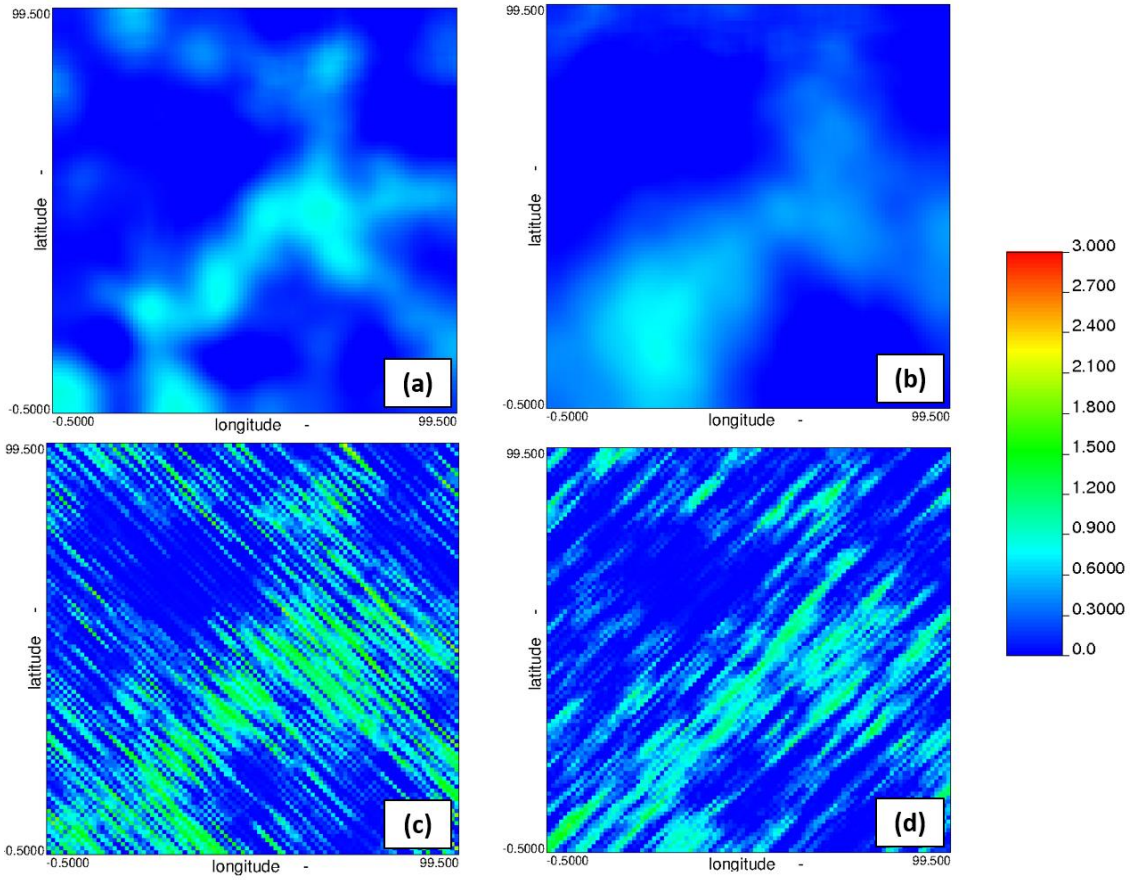


Figura 3-1: Decomposição da imagem de teste em quatro fatores geomorfológicos com FK. (a): fator 2, com feições geológicas na escala de 20m × 20m; (b): fator 3, com feições geológicas na escala de 50m × 50m; (c): fator 4, com artefatos de 1m de largura alinhados com o azimute N135E; (d): fator 5, com artefatos de 5m de largura alinhados com o azimute N045E.

3.2 FFT com espectro global

A decomposição com FFT é realizada através da manipulação do espectro de amplitudes. Para alcançar resultados morfologicamente semelhantes aos da FK, sugere-se os passos a seguir.

i. Criar um campo novo chamado `az` no mesmo mapa onde estão as variáveis que contém a magnitude e a fase resultantes da FFT.

ii. Calcular o campo de meios-azimutes com o *script* de calculadora que se segue:

```
var x0 := 50;
var y0 := 50;
var dx := X_ - x0;
var dy := Y_ - y0;
```

```
az := atan (dy / dx);
if( isNaN( az ))
  az := 0;
az := 90 - az * 180/3.1416;
```

O campo de meios-azimutes resultante está retratado na Figura 3-2. Usa-se meios-azimutes porque o espectro de valores reais é simétrico. O valor do azimute na singularidade da função arcotangente (centro do mapa) é mapeado para N090E.

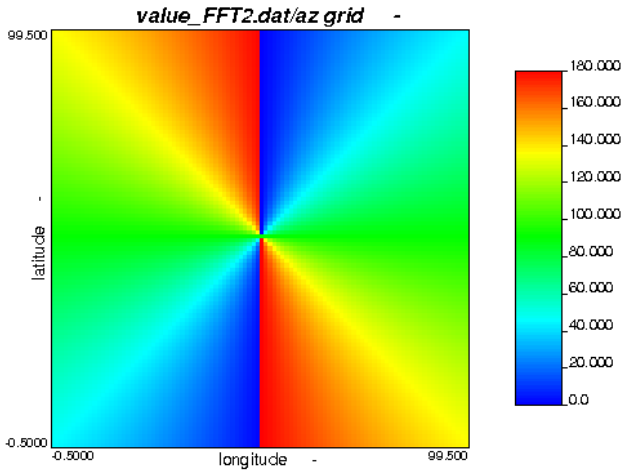


Figura 3-2: Mapa de meios-azimutes.

iii. Criar novas variáveis: `mask2`, `mask3`, `mask4` e `mask5` no mesmo mapa onde estão as variáveis que contém a magnitude e a fase da FFT. Essas variáveis auxiliares marcam as porções do espectro correspondentes a cada estrutura.

iv. Executar o *script* de calculadora a seguir para criar as marcações:

```
var x0 := 50;
var y0 := 50;
var dx := X_ - x0;
var dy := Y_ - y0;
var r := sqrt( dx*dx + dy*dy );
mask2 := 0;
mask3 := 0;
mask4 := 0;
mask5 := 0;
if( r < 1 ){
   mask2 := 1;
   mask3 := 1;
   mask4 := 1;
   mask5 := 1;
}
if( r < 7 )
   mask2 := 1;
if( r < 3 )
   mask3 := 1;
if( az > 45-10 and az < 45+10 )
   mask4 := 1;
if( az > 135-20 and az < 135+20 )
   mask5 := 1;
```

Reparar que todas as estruturas estão designadas na célula central do espectro. O ponto central do espectro corresponde à frequência zero, que é o valor correspondente à média global do mapa. Poder-se-ia atribuir o valor central a uma das estruturas, que seria a

portadora da média e as demais estruturas teriam a mesma forma, mas seriam resíduos com média zero.

v. Criar novas variáveis: `fft2`, `fft3`, `fft4` e `fft5` no mesmo mapa onde estão as variáveis que contém a magnitude e a fase da FFT. Essas variáveis conterão as parcelas da magnitude que lhes cabe, conforme particionamento definido pelo modelador.

vi. Executar o *script* seguinte para distribuir a informação entre as estruturas:

```
fft2 := 0;
fft3 := 0;
fft4 := 0;
fft5 := 0;
var n := mask2 + mask3 + mask4 + mask5;
if( n ){
   var magRatio := Magnitude / n;
   if( mask2 )
      fft2 := magRatio;
   if( mask3 )
      fft3 := magRatio;
   if( mask4 )
      fft4 := magRatio;
   if( mask5 )
      fft5 := magRatio;
}
```

A informação do espectro é distribuída em partes complementares onde as partições se sobrepõem para que, após a retrotransformação, a soma dos fatores geomorfológicos resulte no mapa original. A Figura 3-3 mostra a partição do espectro, feita de forma a resultar em fatores geomorfológicos aproximadamente na mesma escala dos obtidos com FK (Figura 3-1).

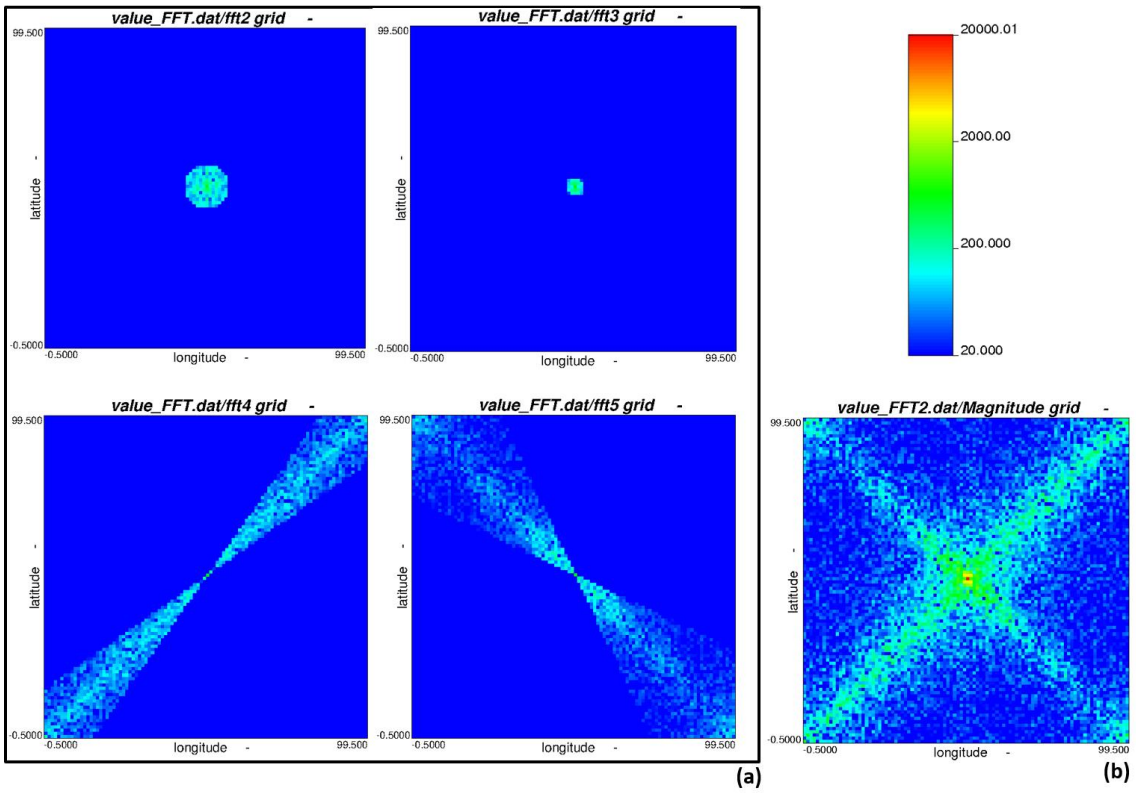


Figura 3-3: Partições do espectro de amplitude (b) para os fatores geomorfológicos (a). Os valores de amplitude estão em escala logarítmica.

vii. Para cada fator, aplicar RFFT com o mesmo mapa de fases original. A Figura 3-4 mostra os fatores geomorfológicos obtidos.

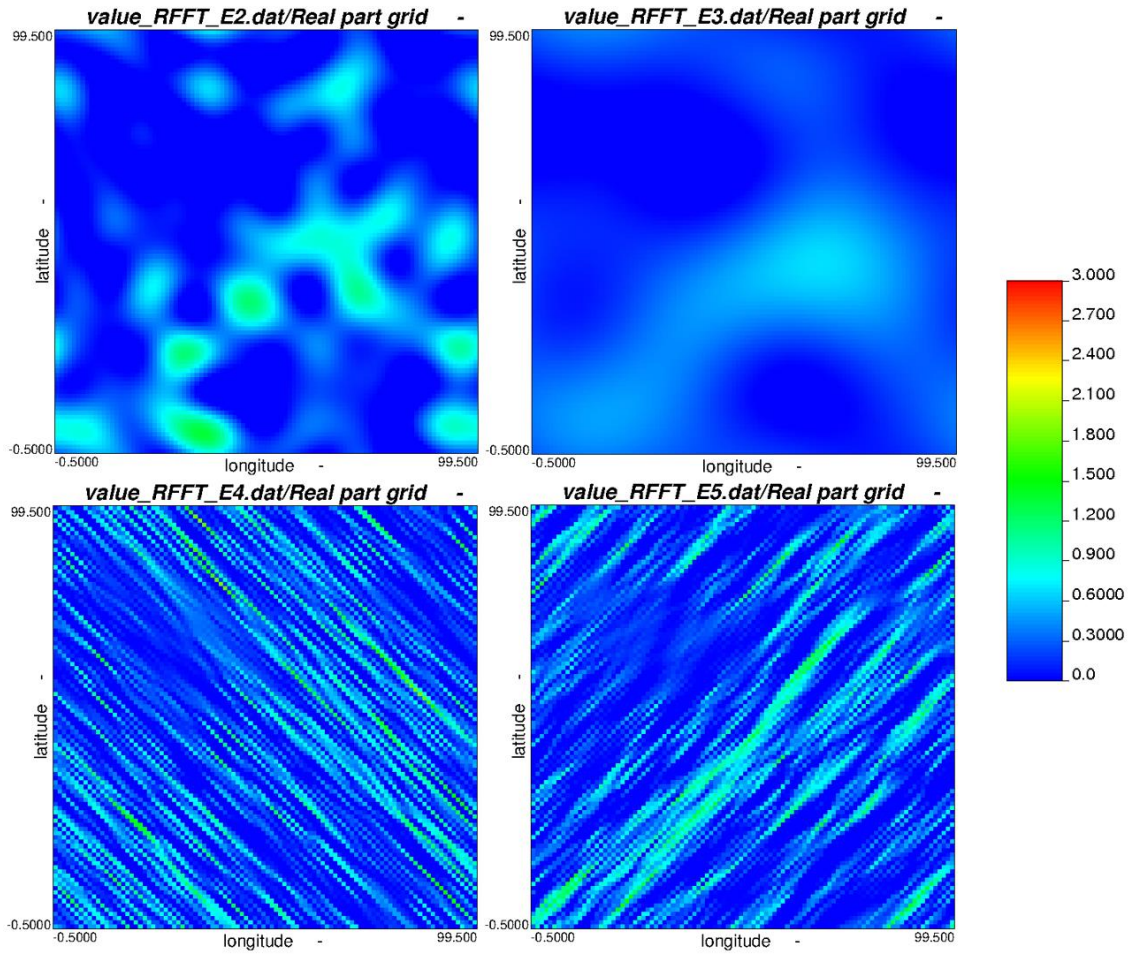


Figura 3-4: Fatores geomorfológicos obtidos com FFT.

3.3 SVD com combinação linear global

Conforme apresentado na Seção 1.1.3, a decomposição da imagem de exemplo através de SVD não é uma decomposição propriamente espacial. É necessário encontrar uma combinação de fatores singulares tal que seja satisfatória. Os pesos da combinação linear de imagens fundamentais (obtidas com SVD) vista na Figura 1-5 podem ser definidos manualmente pelo modelador a fim de obter diretamente os fatores geomorfológicos desejados. A curva de contribuição acumulada de informação (Figura 3-5), para os fatores singulares, mostra que todos os 100 fatores são informativos.

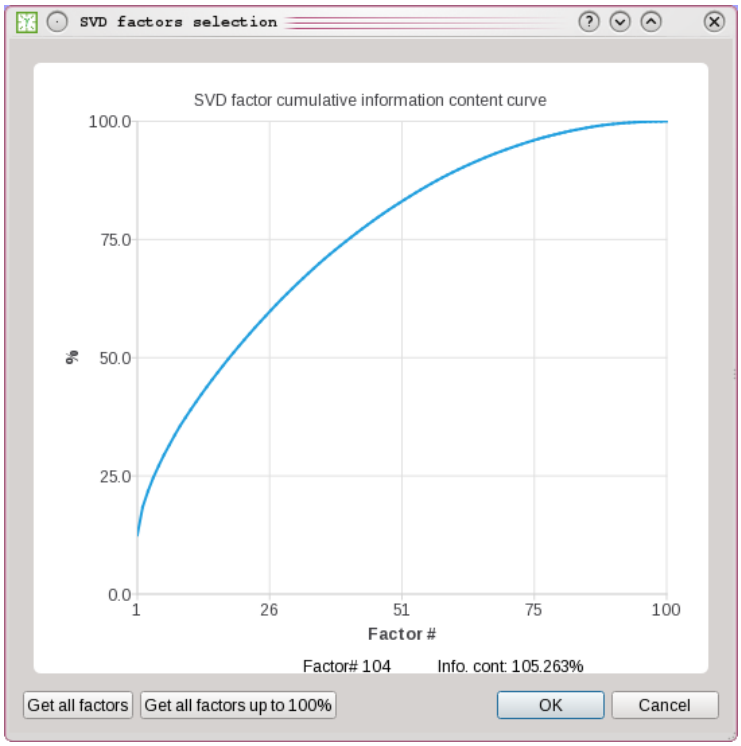


Figura 3-5: Curva de contribuição acumulada de informação para os fatores singulares do mapa de teste.

Dividindo o total de informação em quatro partes aproximadamente iguais, tal como modelado no variograma do experimento feito com FK, chega-se aos fatores geomorfológicos mostrados na Figura 3-6. Na Seção 1.1.3, foi visto que as maiores escalas tendem a aparecer nos primeiros fatores singulares. Assim, eles foram agrupados de forma a refletir as escalas das estruturas modeladas no variograma, indo da de textura mais suave à de textura mais áspera. Os 400 pesos (100 imagens fundamentais × 4 fatores geomorfológicos) usados para obter esses resultados estão no Anexo 2 e foram especificados de forma a obter fatores contendo estruturas de escalas semelhantes às dos fatores obtidos com FK (Figura 3-1).

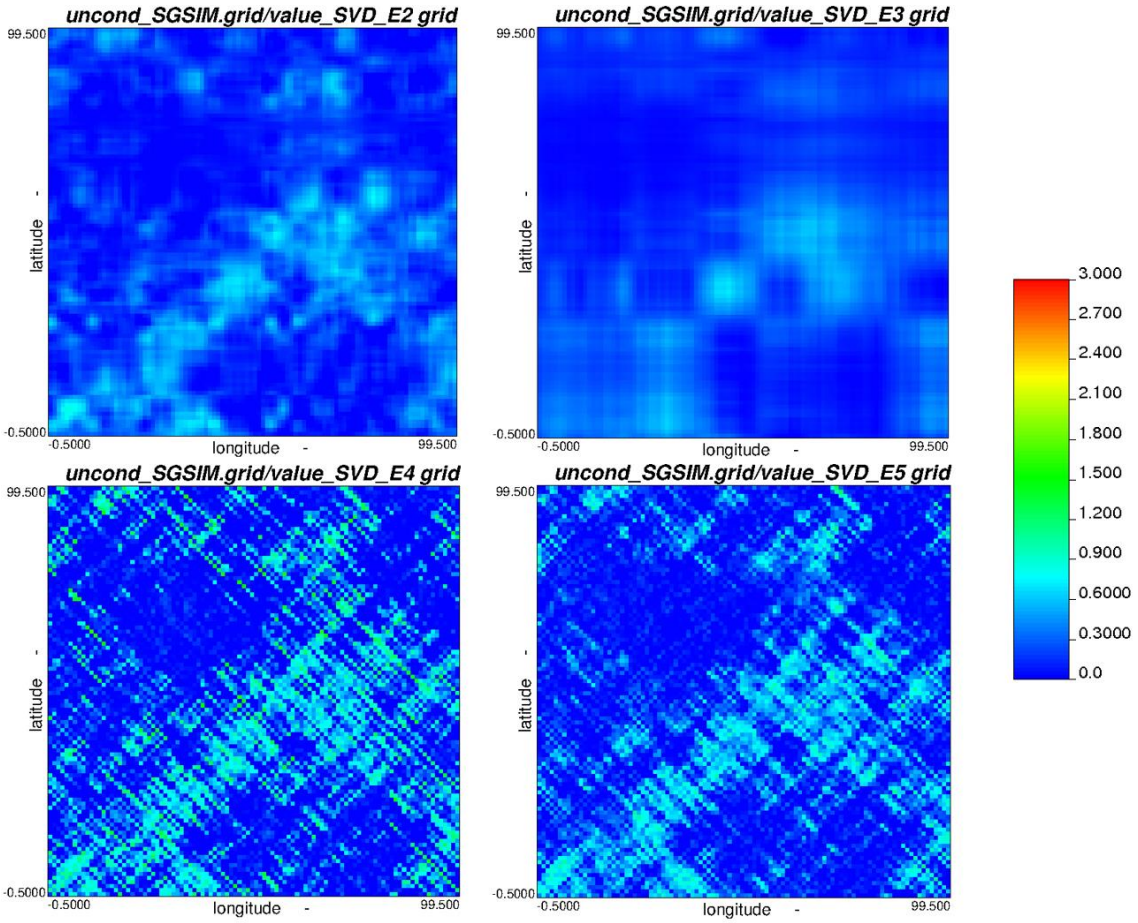


Figura 3-6: Fatores geomorfológicos obtidos com SVD.

3.4 EMD

O dado de teste foi então submetido à decomposição em fatores geomorfológicos pelo algoritmo EMD apresentado na Seção 1.1.4 com os parâmetros mostrados na Figura 3-7 e apresentados na Tabela 3-1. Um mapa com as médias locais foi estimado com krigagem da média (Wackernagel 2003, cap. 4). Essas médias foram subtraídas para que o dado de teste ficasse centrado em zero tal como requisitado para o funcionamento do algoritmo. Esse resultado centrado em zero foi submetido ao algoritmo propriamente dito. Lembrando que EMD é um algoritmo empírico, isto é, não tem um modelo como FK tem o variograma.

Como esperado (ver resultados na Figura 3-8), o algoritmo é eficaz ao decompor o dado em fatores geomorfológicos contendo, cada qual, elementos de mesma escala. Entretanto, tal como SVD, EMD não foi capaz de distinguir os elementos em função de suas orientações. Além disso, EMD resultou em uma distribuição de quantidade de energia (informação) em oposição ao obtido com os métodos anteriores. No caso, a informação está concentrada nos elementos de menor escala ou de maior frequência espacial. Não obstante, os elementos

isotrópicos de escalas equivalentes às estruturas variográficas 2 e 3 (ver Tabela 1-1) estão bem separados na IMF_2 e no resíduo respectivamente.

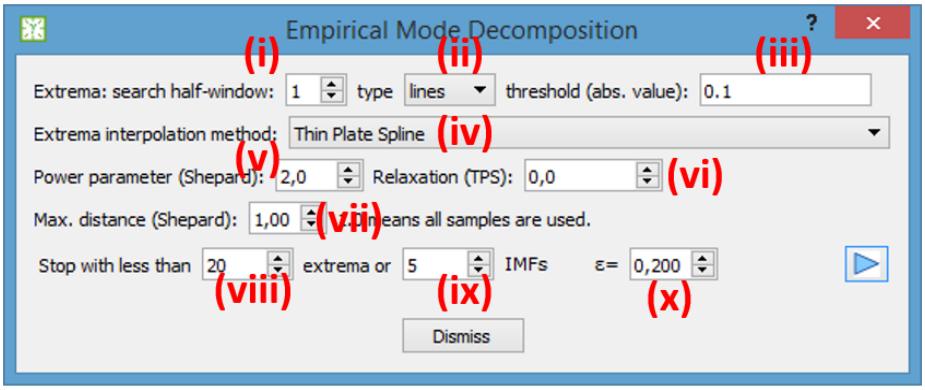


Figura 3-7: Parâmetros para o algoritmo EMD implementado para esta tese.

parâmetro	descrição
(i)	Tamanho da meia-janela para busca de extremos locais. O valor 1 significa uma janela de busca de 3x3x3 células.
(ii)	Tipo de extremo. O valor "line" (linha) significa que os extremos locais são os eixos de "vales" e "divisores de água" ao invés de pontos isolados.
(iii)	Limiar de valores extremos. O valor 0.1 significa que os valores absolutos de extremos locais menores que 0.1 são ignorados, ou seja, não considerados como máximos ou mínimos locais.
(iv)	Tipo de interpolação dos extremos locais. Escolhido *Thin Plate Spline* (com função radial $r^2 \log r$).
(v)	Ignorado para o tipo de interpolação escolhido.
(vi)	Relaxamento da *spline*. O valor zero significa que a superfície interpolada deve passar por todos os extremos locais exatamente. Um valor tendendo ao infinito resulta em um plano equivalentemente ajustado por mínimos quadrados.
(vii)	Ignorado para o tipo de interpolação escolhido.
(viii) e (ix)	Condições de parada forçada do algoritmo.
(x)	Épsilon. Em teoria, o algoritmo deve parar apenas quando o envelope médio é igual a zero. Na prática, se considera que todos os valores absolutos sejam menores que um valor pequeno.

Tabela 3-1: Significados dos parâmetros da Figura 3-7 para o algoritmo EMD aplicado ao dado de teste.

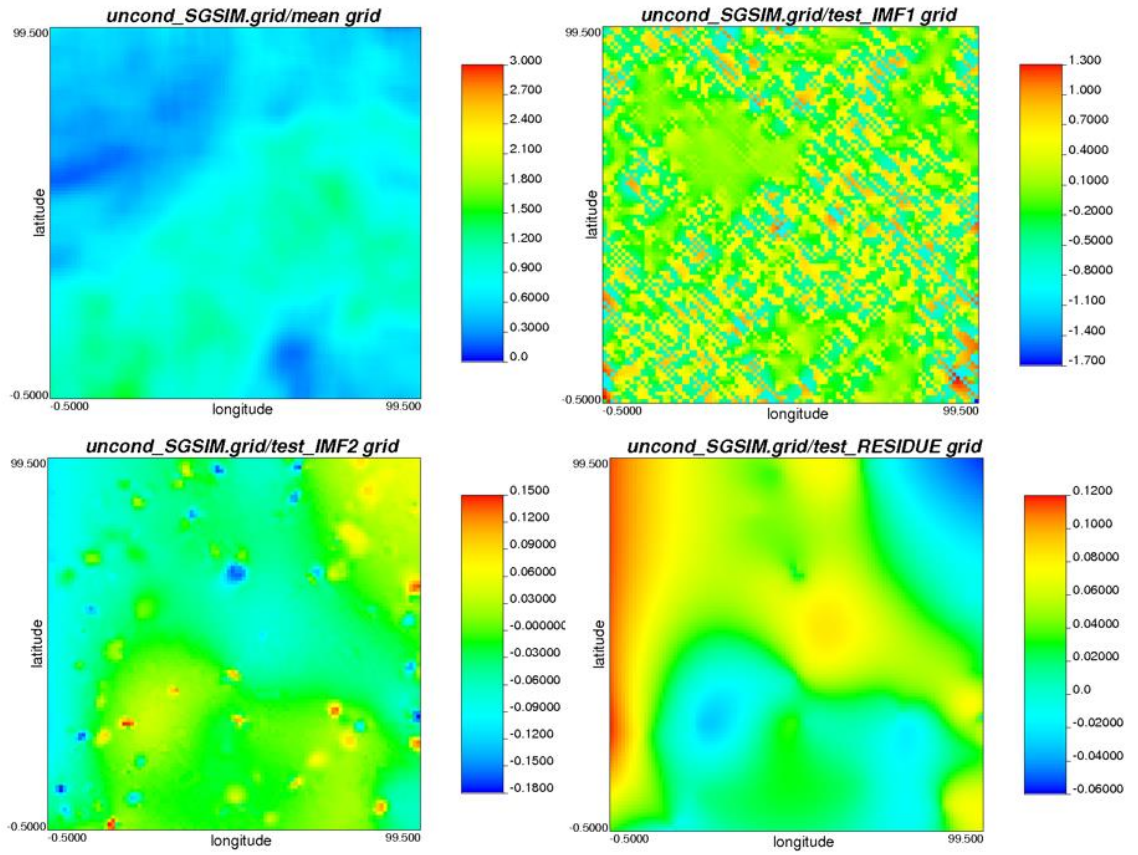


Figura 3-8: Fatores geomorfológicos (IMF_1, IMF_2 e resíduo) obtidos com EMD. O mapa superior à esquerda são as médias locais extraídas do dado antes de executar a decomposição.

3.5 Análise com núcleo Gabor global

Esta seção assume que o leitor esteja familiarizado com os conceitos apresentados na Seção 2.2.5. O dado de teste foi então submetido ao processo de análise com filtro de Gabor. Foram variados a frequência e o azimute e fixados os seguintes parâmetros: tamanho da janela de convolução e os parâmetros do núcleo: m_x e m_y, σ_a e σ_b. Considerando que as maiores estruturas do dado de teste desta tese têm 50m (ver estruturas variográficas na Tabela 1-1), faz sentido fixar o tamanho da janela para 50m × 50m. Como cada célula tem 1m × 1m, o núcleo Gabor será uma malha de 50 células × 50 células. Os parâmetros fixos do núcleo são m_x = 127,5m; m_y = 127,5m; σ_a = 50m; σ_b = 50m.

3.5.1 Conversão de frequência topológica para comprimento de onda

Como a convolução entre malhas regulares é uma operação célula-a-célula, ou seja, ignora a geometria das malhas, o valor da frequência usado para parametrizar um núcleo de Gabor é na verdade uma frequência topológica, isto é, um valor que equivale ao inverso do comprimento da onda em número de células do núcleo original. Então, uma frequência

topológica de 0,003 significa que o comprimento da onda será de aproximadamente 333 células.

O software desenvolvido para os experimentos dispõe de um recurso de redução de células (*upscaling*) do núcleo (o usuário pode querer que o cálculo seja rápido para uma pré-avaliação), então o número de células correspondente a uma frequência topológica deve ser reduzido de acordo. O tamanho máximo do núcleo gerado pelo software é 255 × 255 células. Assim, para um núcleo de 50 × 50 células, a redução é por um coeficiente de 5,1 nas duas direções. Portanto, a frequência topológica de 0,003 corresponde a 333/5,1 = 65,3 células nas duas direções. Como o tamanho da célula é de 1m × 1m, essa frequência topológica corresponde a um comprimento de onda de 65,3m nas duas direções. Em resumo, um núcleo Gabor cujo parâmetro frequência seja 0,003 consoará com estruturas de 65,3m ao longo do azimute configurado.

Para facilitar a parametrização da frequência dos núcleos, o software mostra (Figura 3-9) os comprimentos de onda resultantes em função do tamanho máximo do núcleo, do tamanho de núcleo configurado pelo usuário, do tamanho espacial das células e dos valores de frequência.

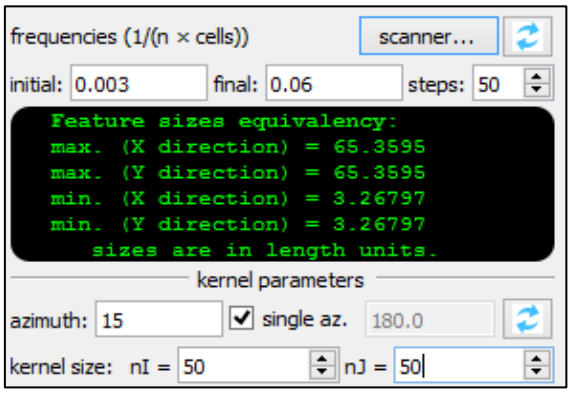


Figura 3-9: Recurso que converte frequências topológicas em comprimentos de onda. O usuário configura as frequências inicial e final e o tamanho do núcleo em células e o programa mostra (painel com fundo preto) os tamanhos máximo e mínimo que poderão ser sintonizados em cada direção principal.

3.5.2 Processo de extração de fatores geomorfológicos

A Figura 3-10 ilustra o processo para obter um mapa contendo estruturas de determinada frequência e azimute a partir do mapa frequência × azimute das médias das amplitudes das respostas do filtro de Gabor. Para cada frequência e cada azimute, dois núcleos Gabor (um para a componente real da resposta e outro para a imaginária) são gerados a partir desses parâmetros. Em cada iteração, o dado de entrada é convolvido com os núcleos. Um mapa

de amplitudes é computado a partir das respostas real e imaginária (ver relação entre as formas Cartesiana e polar dos números complexos na Figura 2-8). A média dos valores de amplitudes é calculada. Essa média é plotada no mapa frequência × azimute.

O modelador seleciona uma área do diagrama, representando um intervalo de frequências e de azimutes. O programa então percorre esse intervalo e, para cada azimute e frequência desse intervalo, gera um núcleo Gabor. Esse núcleo Gabor é submetido à FFT, de onde se obtém seu espectro de amplitudes. Essas amplitudes são unitizadas e combinadas (prevalência do maior valor) com as amplitudes unitizadas das iterações anteriores. Essas amplitudes combinadas são multiplicadas célula-a-célula com o espectro de amplitudes do dado de entrada. O resultado é, junto com o espectro de fases do dado de entrada, submetido à FFT reversa para obtenção do mapa filtrado.

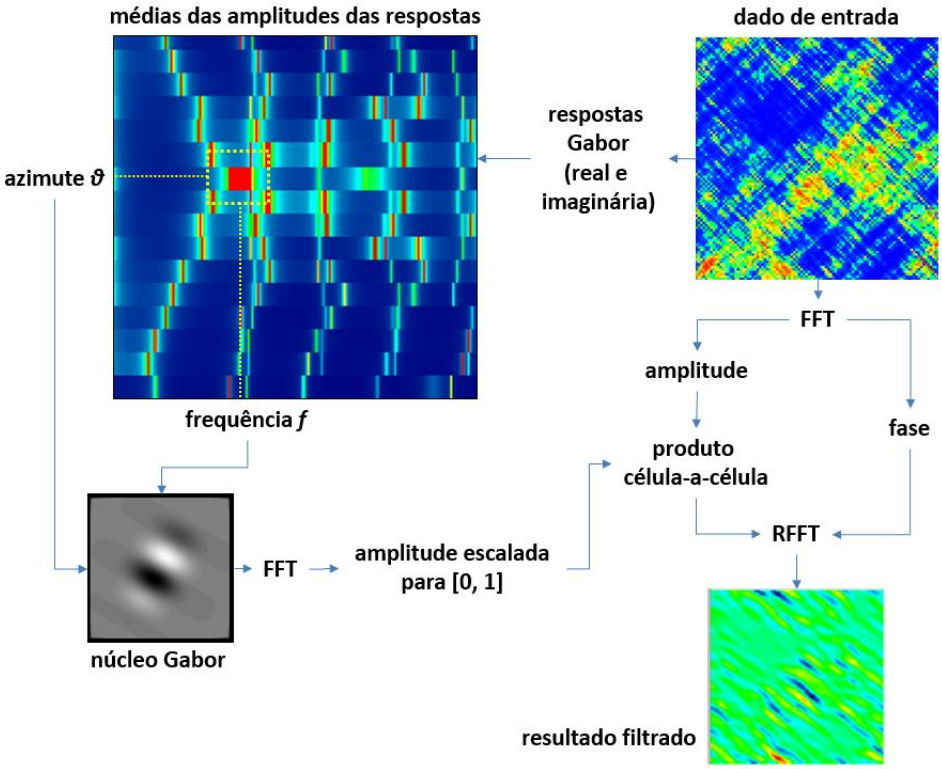


Figura 3-10: Fluxo de obtenção de um mapa contendo estruturas de determinados frequência e azimute utilizando o filtro de Gabor.

3.5.3 Procedimento e resultado

O azimute é variado de N000E a N180E em passos de 5° e a frequência topológica (ver Seção 3.5.1), variada de 0,003 (λ = 65,3m) a 0,09 (λ = 2,1m) em passos de 0,0001. A razão para o valor mínimo de frequência é que ele corresponde a um comprimento de onda maior do que a maior estrutura (50m) modelada com o variograma da Tabela 1-1. A razão para o

valor máximo de frequência topológica utilizado é que ele é muito próximo do limite de Nyquist (Tanembaum, 2003, p. 94) correspondente ao comprimento de onda de 2m = 2 × o tamanho de uma única célula de 1m. Portanto, valores acima de 0,09 resultariam em truncamento (*aliasing*) das formas de onda no núcleo.

Para cada reposta do núcleo Gabor, cada qual uma malha 100 × 100 células, foi calculada a média dos valores de amplitude (módulo das respostas real e imaginária do filtro). Essas médias então foram plotadas em um mapa frequência versus azimute (Figura 3-11, parte de cima).

Nesse mapa foram destacadas as bandas correspondentes às estruturas: (a) são as estruturas isotrópicas na banda de frequência correspondente a corpos de 50m; (b) são as estruturas isotrópicas na banda de frequência correspondente a corpos de 20m; (c) são as estruturas anisotrópicas de banda larga correspondente aos artefatos de 5m ortogonais ao azimute N135E; (d) são as estruturas anisotrópicas de banda larga correspondente aos artefatos de 1m ortogonais ao azimute N045E. As estruturas foram destacadas no diagrama e as frequências/azimutes correspondentes selecionadas para uma transformação reversa (detalhes desse processo estão apresentados na Seção 3.5.2) para extração dos fatores geomorfológicos (Figura 3-11, parte de baixo).

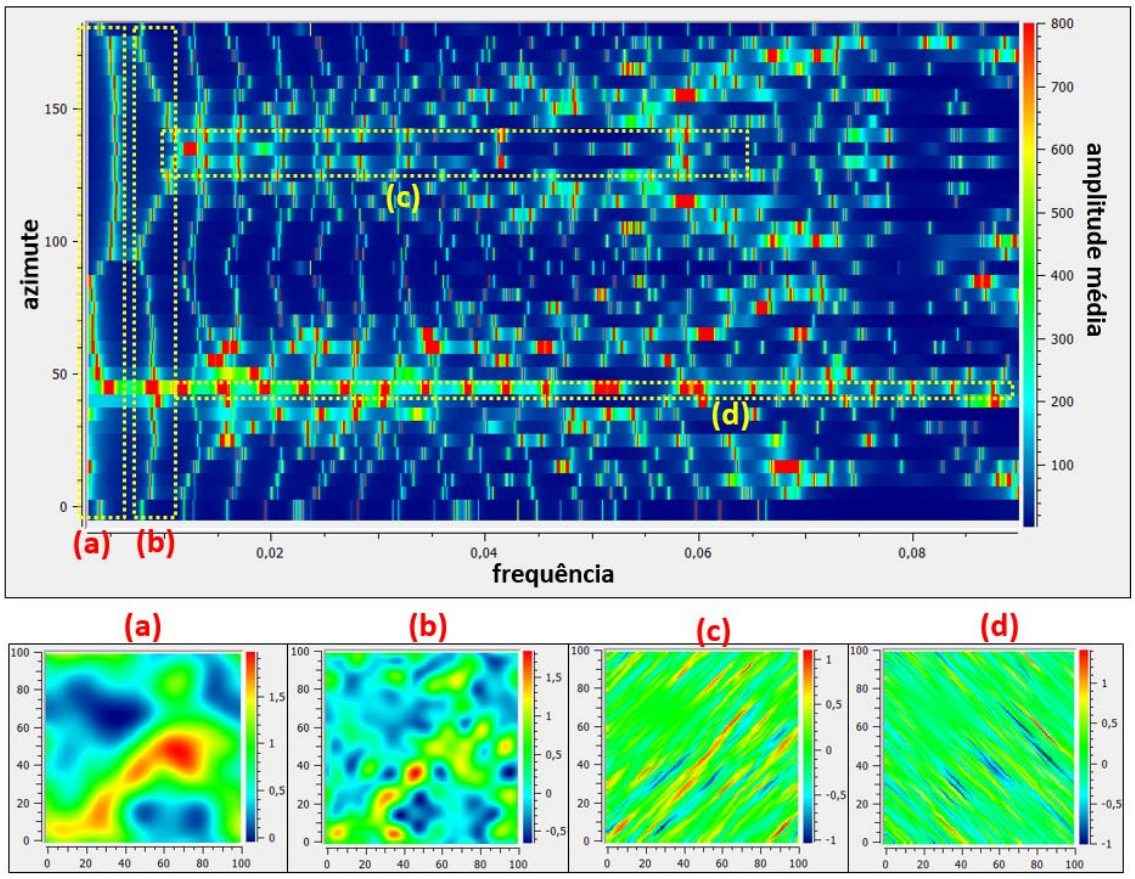


Figura 3-11: No alto, as amplitudes médias das respostas do filtro de Gabor nas diversas frequências e azimutes. Em baixo, os mapas gerados com FFT reversa a partir das frequências e azimutes selecionados no diagrama de cima. (a), (b), (c) e (d): ver texto.

3.6 FWT com ondaleta global

FWT, assim como a análise de Gabor, não obstante mais simples, também resulta em um modelo que possui uma dimensão extra: o escalagrama. Porém, diferente do espectrograma, o escalagrama é uma pilha de mapas que representa sucessivamente níveis de detalhe, isto é, não é um volume regular. Portanto, antes de proceder ao trabalho de filtragem faz-se mister compreender como o FWT estrutura o resultado e como o escalagrama é construído.

3.6.1 Preparação do dado de entrada

Primeiramente é necessário preparar o mapa no sentido de torná-lo compatível com o algoritmo FWT. O programa usado nos experimentos executa este passo automaticamente, porém pode ser preciso executar manualmente em outros ambientes. O mapa é aumentado até que se torne uma malha quadrada cujo tamanho seja uma potência de 2. No caso do dado de teste, de 100 × 100 células, ele é expandido para 128 × 128 células.

Os dados originais são copiados para o canto superior esquerdo do mapa expandido e as células excedentes são preenchidas por espelhamento, uma técnica conhecida como *mirror-padding*. O espelhamento faz com que as formas do mapa sejam periódicas e não formem bordas secas. A presença de descontinuidades pronunciadas pode introduzir artefatos no resultado, o que pode atrapalhar a interpretação. A Figura 3-12 ilustra o processo de preparo do *grid* de trabalho.

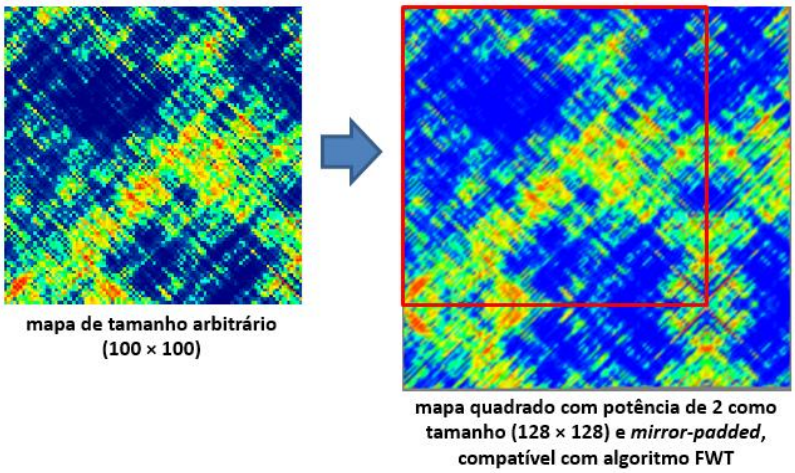


Figura 3-12: Compatibilizando o mapa de entrada com o algoritmo FWT.

3.6.2 Estruturação do resultado

FWT reaproveita o mapa de entrada para armazenar os coeficientes resultantes (os escalagramas), que são dispostos em um arranjo piramidal refletindo o caráter hierárquico dos níveis de detalhe (escalas). A Figura 3-13 mostra como a saída do FWT é estruturada: o primeiro mapa é o resultado da transformada; o segundo mapa delimita os resultados por nível (escala); o terceiro mapa delimita os resultados por direção.

FTW, assim como FFT, realiza a transformada 2D em dois passos 1D. Porém, como a ondaleta é unidimensional, o resultado se restringe a três direções: N-S (direção 1); diagonais (direção 2) e E-O (direção 3). Cada escala corresponde a um nível crescente de detalhe, assim, quanto maior o nível, maior a frequência espacial das estruturas nele representadas.

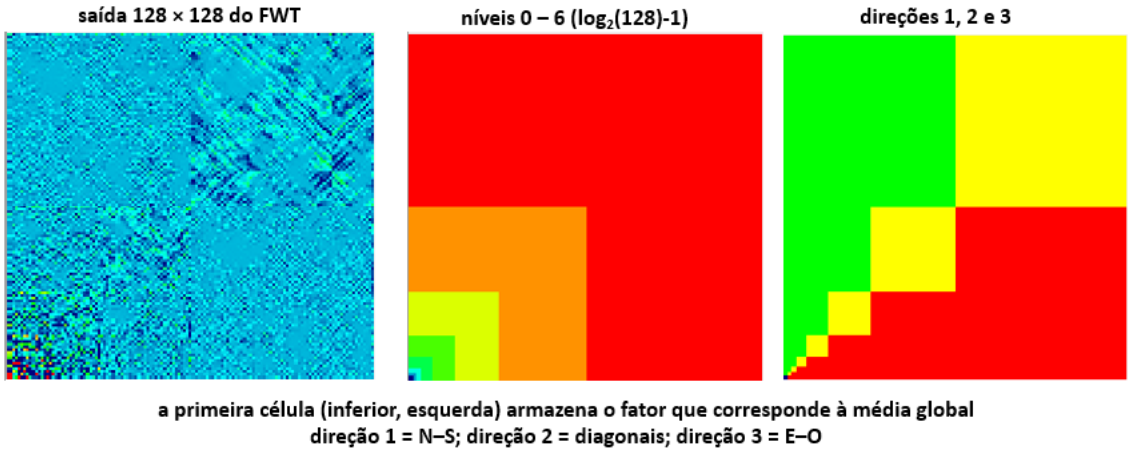


Figura 3-13: Organização da saída do algoritmo FWT (ver texto).

3.6.3 Construção dos escalagramas

A partir da estruturação do resultado do FWT, abordada na Seção 3.6.2, procede-se à construção dos escalagramas, para que o conteúdo espectral do mapa seja posicionado no espaço corretamente para interpretação. A Figura 3-14 ilustra o processo de construção dos escalagramas a partir do resultado da FWT, que é feito automaticamente pelo software empregado nos experimentos.

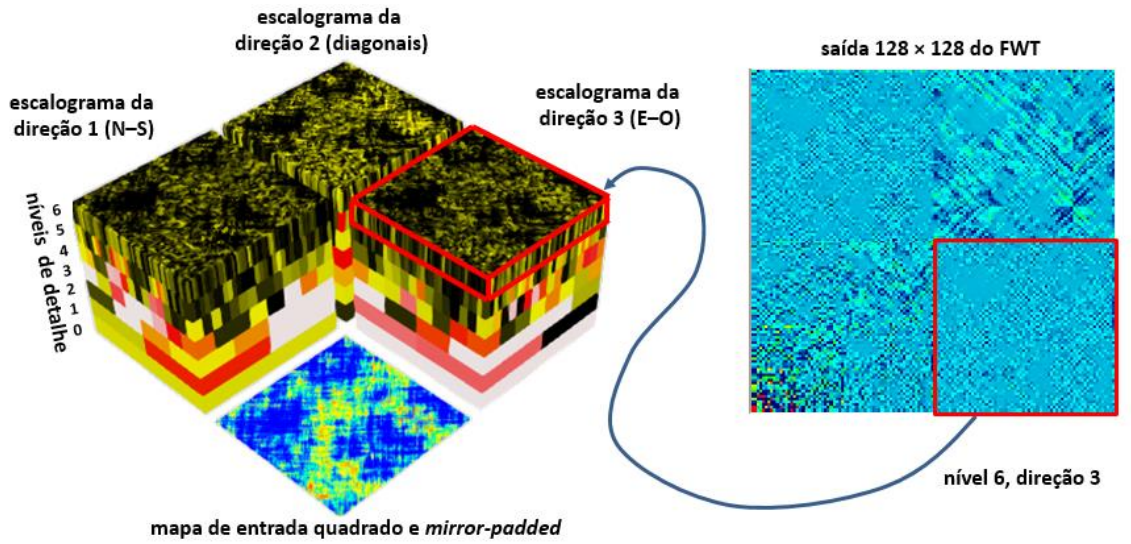


Figura 3-14: Processo de construção dos escalagramas.

3.6.4 Obtenção dos fatores geomorfológicos

O dado de entrada foi então submetido ao FWT com a ondaleta de Daubechies (Daubechies, 1990) do tipo 18 (conhecida por D18) e centralizada. A centralização da ondaleta serve para criar um contraste no mapa 2D que contém o resultado cru do FWT (ver Figura 3-13) a fim de facilitar a interpretação quando uma visualização 3D dos escalagramas não está disponível.

Nota-se ainda a forma com que o FWT 2D deve ser executado. No caso em questão, os passos FWT 1D são executados alternadamente (entrelaçado), por linhas e por colunas (a outra opção é executar primeiro nas linhas, depois nas colunas). A Figura 3-15 mostra a parametrização, assaz simples, do algoritmo FWT tal como implementado no software utilizado nos experimentos. O resultado da transformação está mostrado na Figura 3-16.

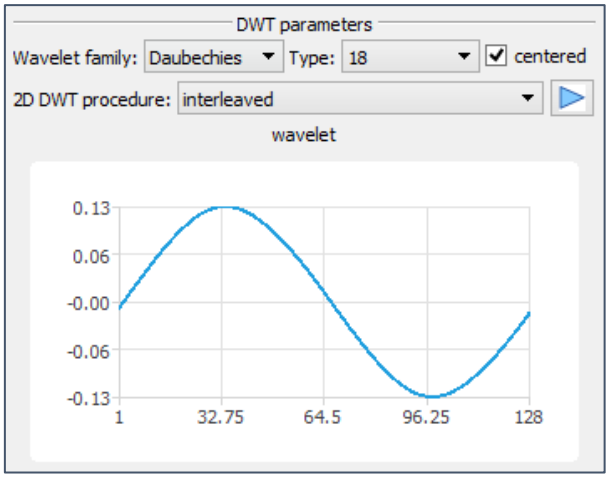


Figura 3-15: Parametrização da ondaleta para o algoritmo FWT implementado no software para os experimentos.

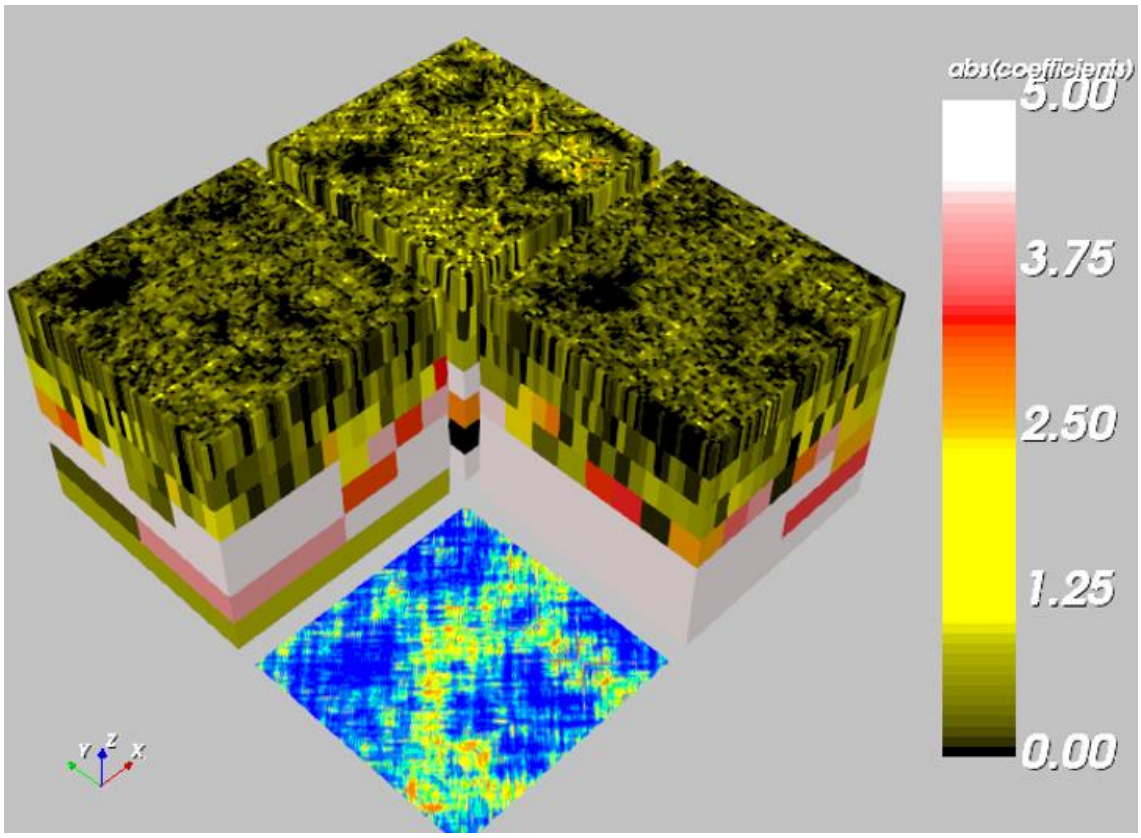


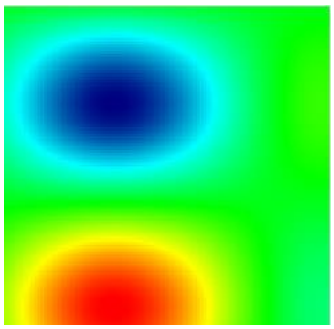
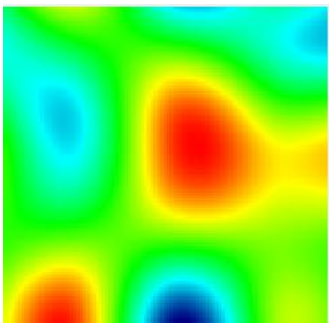
Figura 3-16: Escalagramas obtidos com FWT com a ondaleta configurada como na Figura 3-15.

Consistentemente com os demais métodos, a maior quantidade de energia (informação) está concentrada nas estruturas de maior escala (níveis de detalhe mais baixos). Em seguida, o resultado do FWT foi salvo com um mapa regular, junto com os campos mostrados na Figura 3-13, a fim de que os coeficientes dos escalagramas possam ser manipulados facilmente através de instruções condicionais em *scripts* de calculadora de propriedades. Essa manipulação dos coeficientes é chamada de *thresholding* no jargão do processamento de imagens.

Com o *script* simples a seguir é possível criar filtros por escala (frequência espacial):

```
if( I_ != 0 or J_ != 0 ){
   if( level >= 3 )
      coefficient:=0.0;
   else
      coefficient:=coef_original;
} else
    coefficient:=coef_original;
```

Depois de manipulados, os coeficientes foram trazidos de volta para os escalagramas e foi aplicada o FWT reverso. A Tabela 3-2 mostra as filtragens por cada nível e suas respectivas retrotransformações. Assim, a obtenção de fatores geomorfológicos é muito simples com FWT e *scripts* de calculadora.

níveis	**resultado da retrotransformação**
nenhum	mapa com valor constante (média global = 0,77)
0	
1	

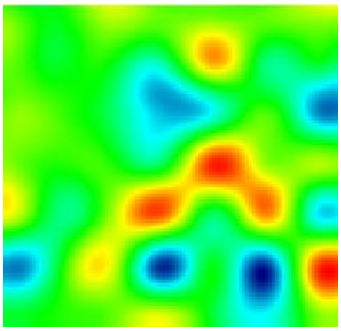
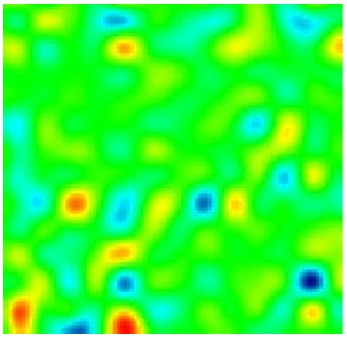
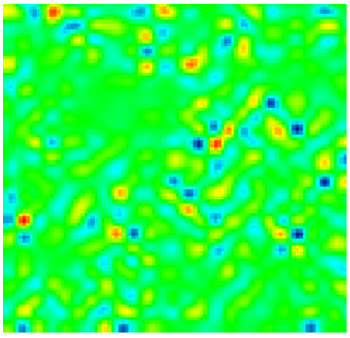
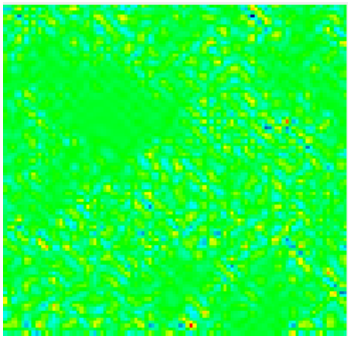
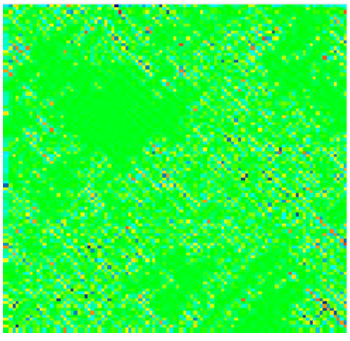
2	
3	
4	
5	
6	

Tabela 3-2: Os níveis de 0 a 6 da transformada ondaleta sobre o dado de teste e suas retrotransformações (fatores geomorfológicos).

Somando a média global (0,77) e os fatores 0, 1 e 2 tem-se o resultado mostrado na Figura 3-17. Esse resultado mostra os corpos de maior escala (50m). Somando o nível 3, acrescenta-se os corpos de menor escala (20m), mostrados na Figura 3-18. A partir do nível 4 figuram os artefatos. A separação dos artefatos (Figura 3-19) foi possível porque estão em escalas diferentes (níveis 4 a 6) das da informação geológica. Não seria possível, por exemplo, separar os dois componentes de artefatos em função dos seus azimutes, uma vez que o algoritmo FWT só distingue três direções: N-S, diagonais e E-O.

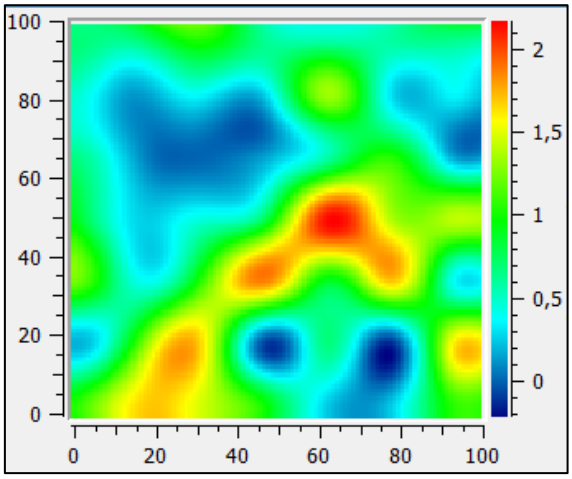


Figura 3-17: Soma da média global (0,77) e os níveis de 0 a 2 mostrados na Tabela 3-2.

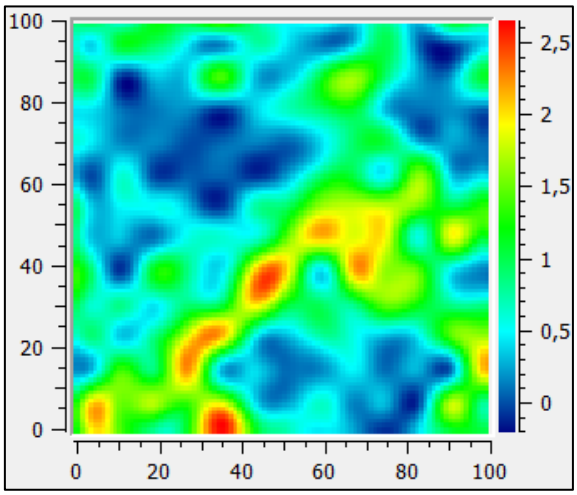


Figura 3-18: Soma da média global (0,77) e os níveis de 0 a 3 mostrados na Tabela 3-2.

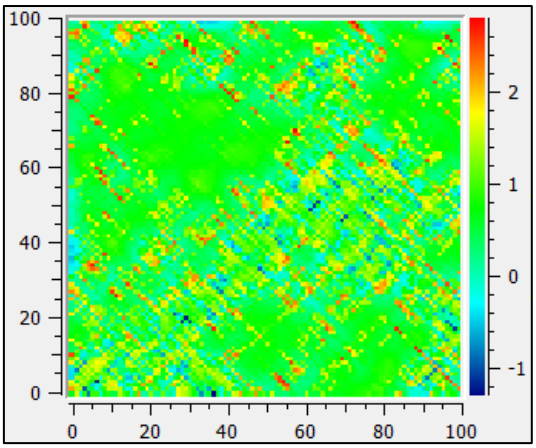


Figura 3-19: Soma dos níveis de 4 a 6 mostrados na Tabela 3-2

3.7 Comparativo e discussão

À luz dos resultados obtidos, os diversos algoritmos avaliados para decomposição estrutural foram avaliados qualitativamente. Os critérios são:

i. **Decomposição**: refere-se ao grau de decomposição em relação ao objetivo proposto nesta tese. Uma decomposição completa que dizer que o algoritmo consegue separar feições tanto quanto à escala quanto ao azimute.

ii. **Facilidade de uso**: refere-se à percepção de facilidade ou de dificuldade ao utilizar o algoritmo para alcançar o objetivo proposto. Um algoritmo fácil de usar é aquele em que se tenta poucas configurações de parâmetros para chegar a um resultado satisfatório.

iii. **Desempenho computacional**: refere-se à percepção de rapidez no processamento dos algoritmos tais como implementados no software utilizado para os experimentos. Deve-se considerar que pode haver implementações melhores (ou piores) desses algoritmos.

iv. **Parametrização**: refere-se à quantidade de parâmetros para ajustar.

v. **Qualidade do resultado**: refere-se à percepção de qualidade dos fatores geomorfológicos obtidos. Por exemplo, os fatores obtidos com SVD são "pouco geológicos", no sentido de que podemos encontrar linhas retas neles (ver Figura 3-6). Deve-se considerar que a qualidade do resultado depende sensivelmente do dado de entrada. Os algoritmos avaliados negativamente podem ter resultados melhores para dados com outros caráteres espectrais.

algoritmo	decomposição	facilidade de uso	desempenho computacional	parametrização	qualidade do resultado
FK	completa	difícil	médio	grande	excelente
FFT	completa	difícil	bom	pequena	boa
SVD	não separa por azimute	fácil	bom	pequena	ruim
EMD	inconclusivo	fácil	ruim	média	ruim
Gabor	completa	média	bom	média	boa
FWT	azimutes restritos a N-S, E-O e diagonais.	fácil	excelente	média	boa

Tabela 3-3: Comparativo dos algoritmos avaliados para decomposição estrutural quanto a alguns critérios qualitativos.

O resultado do FK é esperado que tenha sido o melhor, pois os variogramas teóricos do mapa são conhecidos (não modelados a partir do variograma experimental), para que sejam o resultado de referência para avaliação do método proposto nesta tese. SVD teve um resultado ruim certamente devido ao fato de que esse algoritmo, sendo um método de fatoração de matrizes, não tem o propósito de realizar uma decomposição estrutural. EMD conseguiu separar os artefatos, mas os outros fatores tiveram aspecto pouco geológico; coincidentemente o único dos métodos investigados que não emprega um modelo teórico. FWT, com complexidade de tempo de execução O(n), é o mais rápido de todos os algoritmos avaliados, no entanto só consegue discernir três orientações.

Capítulo 4

Decomposição automatizada com *Deep Learning*

Neste capítulo são investigadas diversas arquiteturas de rede profunda para decomposição estrutural do dado de teste apresentado na Seção 1.5. O tópico *Deep Learning* está apresentado na Seção 2.5 e os experimentos estão implementados como Python Notebooks (ver Seção 1.5). As arquiteturas apresentadas neste Capítulo são facilmente implementáveis a partir dos diagramas através da sintaxe da camada de alto nível Keras (Chollet *et al.* 2015) que abstrai uma camada de software de mais baixo nível (*backend*), por exemplo o TensorFlow.

4.1 Arquitetura 1

A Figura 4-1 mostra a arquitetura usada nesta Seção. CONV são as camadas convolutivas propriamente ditas. MAXPOOL são "camadas" cujos núcleos (*kernels*) retornam o valor máximo encontrado ao invés de uma convolução. As funções φ são as funções de ativação (ver Seção 2.5.1), presentes apenas nas camadas convolutivas e neuronais. Os valores *f* são as dimensões dos núcleos; *s* são as dimensões do *stride* e *p*, as do *padding* (sobre os três hiperparâmetros ver Seção 2.5.4). A intenção dessa arquitetura é reduzir a informação espacial (contagem de pixels por imagem) concomitantemente ao aumento da informação de feições (número de imagens).

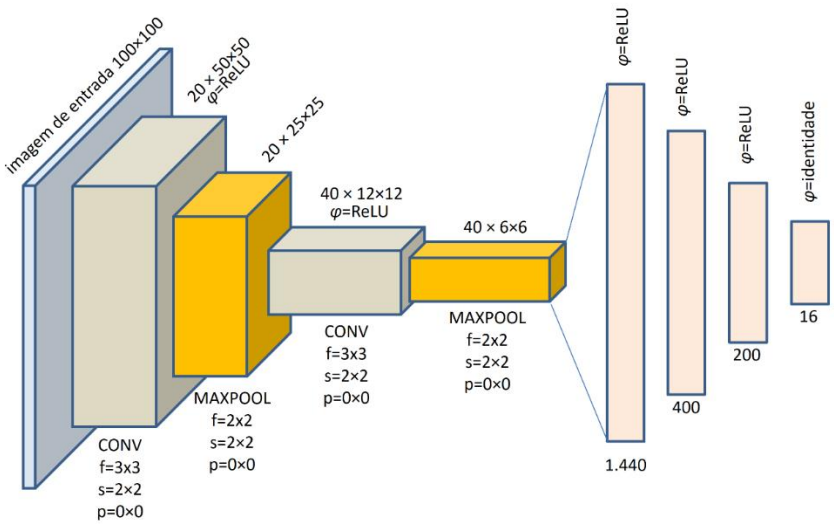


Figura 4-1: Arquitetura 1. Ver texto.

4.1.1 Treinamento com superfícies variográficas

Nesta abordagem foram gerados 160.000 modelos variográficos aleatórios de quatro estruturas. A Figura 4-2 mostra 25 desses variogramas teóricos com seus respectivos parâmetros variográficos. Esse número de variogramas requer 26GB de RAM livres, tanto para as imagens em si, quanto para os tensores (volumes contendo os pesos entre as camadas) gerados no treinamento. Dependendo do sistema, o número de variogramas deve ser ajustado a menor. A geração de 160.000 variogramas leva cerca de 7 minutos[22] com a compilação JIT (*just in time*) proporcionada pelo pacote Numba do Python.

[22] Sistema com processador Intel Xeon de 12 núcleos lógicos, clock de 2.90 GHz e 32GB de memória.

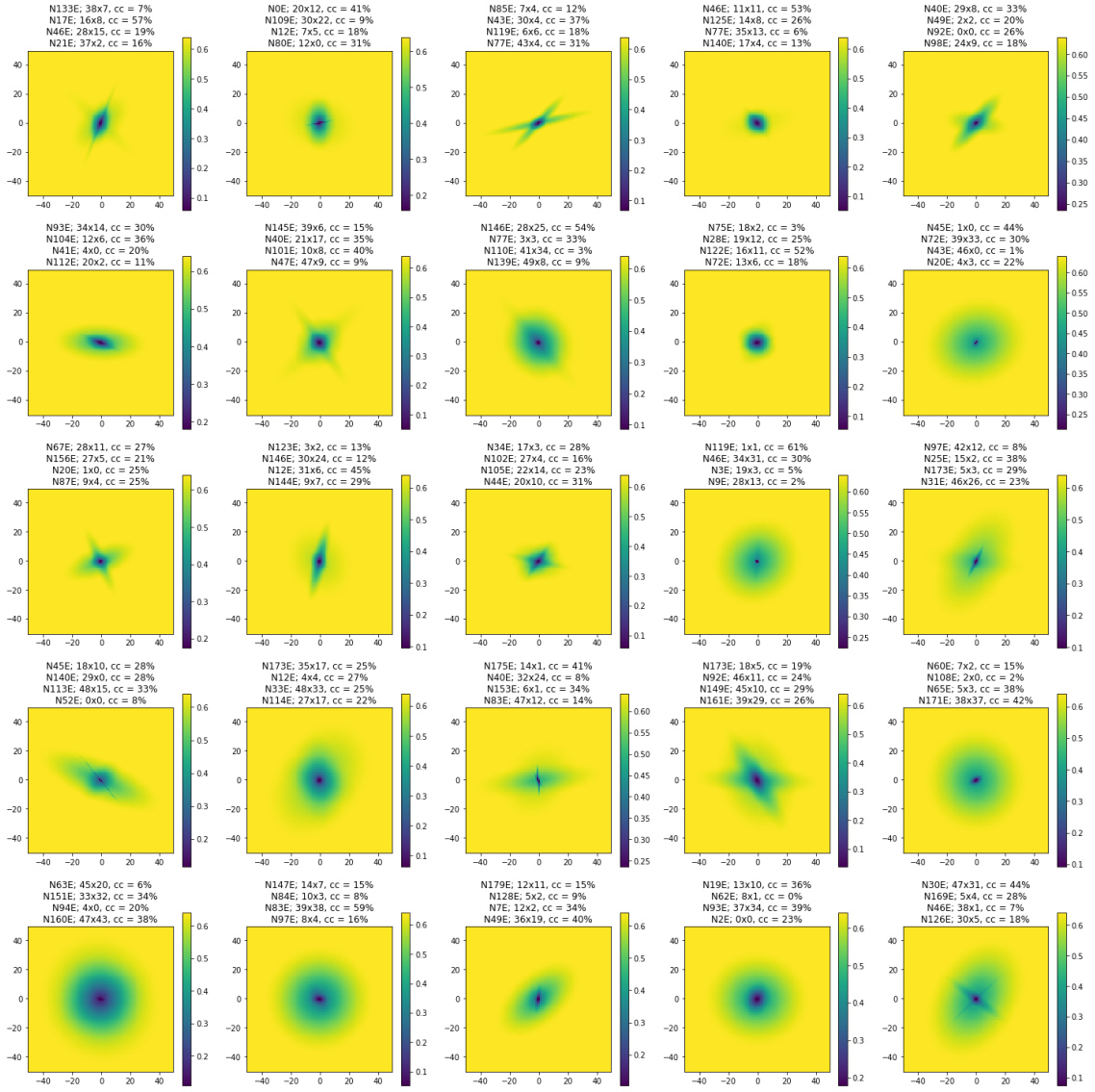


Figura 4-2: 25 modelos variográficos para treinamento da CNN. Os azimutes, semieixos e contribuições de cada uma das quatro estruturas imbricadas geradas aleatoriamente estão nos títulos de cada imagem.

A totalidade de variogramas foi dividida em dois conjuntos: 75% para treinamento em si e 25% para o conjunto de validação. Não foi usado um conjunto de teste.

Esses mapas foram utilizados para treinar (em 20 épocas) uma CNN com a arquitetura 1 (Seção 4.1). A intenção é de que a rede aprenda a relação entre as formas presentes nos variogramas e os parâmetros variográficos. A seguir, o relatório de treinamento da rede gerado pelo programa:

```
Train on 120000 samples, validate on 40000 samples
Epoch 1/20
120000/120000 [==============================] - 97s 810us/step - loss: 0.0522 - val_loss: 0.0508
Epoch 2/20
120000/120000 [==============================] - 98s 817us/step - loss: 0.0505 - val_loss: 0.0501
Epoch 3/20
120000/120000 [==============================] - 99s 823us/step - loss: 0.0501 - val_loss: 0.0502
Epoch 4/20
120000/120000 [==============================] - 123s 1ms/step - loss: 0.0499 - val_loss: 0.0497
```

```
Epoch 5/20
120000/120000 [==============================] - 128s 1ms/step - loss: 0.0497 - val_loss: 0.0495
Epoch 6/20
120000/120000 [==============================] - 115s 962us/step - loss: 0.0495 - val_loss: 0.0494
Epoch 7/20
120000/120000 [==============================] - 114s 946us/step - loss: 0.0495 - val_loss: 0.0493
Epoch 8/20
120000/120000 [==============================] - 129s 1ms/step - loss: 0.0494 - val_loss: 0.0493
Epoch 9/20
120000/120000 [==============================] - 129s 1ms/step - loss: 0.0493 - val_loss: 0.0492
Epoch 10/20
120000/120000 [==============================] - 127s 1ms/step - loss: 0.0492 - val_loss: 0.0491
Epoch 11/20
120000/120000 [==============================] - 129s 1ms/step - loss: 0.0491 - val_loss: 0.0491
Epoch 12/20
120000/120000 [==============================] - 128s 1ms/step - loss: 0.0491 - val_loss: 0.0492
Epoch 13/20
120000/120000 [==============================] - 123s 1ms/step - loss: 0.0491 - val_loss: 0.0490
Epoch 14/20
120000/120000 [==============================] - 125s 1ms/step - loss: 0.0490 - val_loss: 0.0492
Epoch 15/20
120000/120000 [==============================] - 119s 995us/step - loss: 0.0490 - val_loss: 0.0491
Epoch 16/20
120000/120000 [==============================] - 125s 1ms/step - loss: 0.0489 - val_loss: 0.0489
Epoch 17/20
120000/120000 [==============================] - 127s 1ms/step - loss: 0.0489 - val_loss: 0.0490
Epoch 18/20
120000/120000 [==============================] - 129s 1ms/step - loss: 0.0489 - val_loss: 0.0488
Epoch 19/20
120000/120000 [==============================] - 129s 1ms/step - loss: 0.0488 - val_loss: 0.0489
Epoch 20/20
120000/120000 [==============================] - 129s 1ms/step - loss: 0.0488 - val_loss: 0.0488
```

O valor *loss* corresponde ao valor da função-objetivo avaliada ao final de cada época. Nota-se que estabiliza rapidamente em um patamar de 0,0490, de forma que um número maior de épocas não melhoraria significativamente a convergência.

Após o treinamento, que leva cerca de 45 minutos em um sistema com 12 núcleos lógicos de processamento (Intel Xeon Hexacore 3GHz), foi apresentado o mapa variográfico experimental (varmap – Figura 4-3) do dado original para predição dos parâmetros variográficos. Este teste pode ser executado com o arquivo (Python Notebook) `Arch1_Varmap.ipynb` (ver Seção 1.5 para o caminho completo).

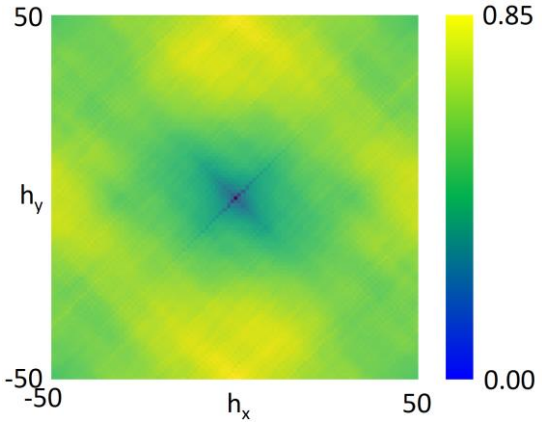


Figura 4-3: Mapa variográfico do dado.

Os resultados não foram aceitáveis (ver exemplo na Figura 4-4) e variaram muito a cada execução, implicando uma sensibilidade forte à parte estocástica (caminho aleatório) do algoritmo e comprometendo a consistência do método.

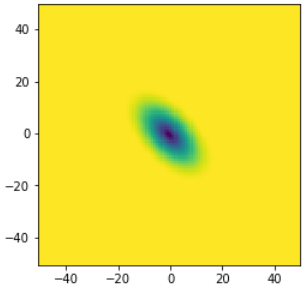


Figura 4-4: Modelo variográfico predito pela rede com a Arquitetura 1 treinada com superfícies variográficas para o caso desta tese.

4.1.2 Treinamento com mapas sintetizados com FIM

Nesta abordagem foram sintetizados 160.000 mapas com FIM (Seção 2.4) a partir de modelos variográficos aleatórios de quatro estruturas e do mesmo mapa de fases do dado original (Figura 2-13). Trata-se de um tipo de *data augmentation* (ver Seção 2.5.7). A Figura 4-5 mostra 25 desses mapas sintéticos com seus respectivos parâmetros variográficos. Assim como no treinamento com variogramas, esse número de mapas requer 26GB de RAM livres, tanto para os mapas em si, quanto para os tensores gerados no treinamento. Mais uma vez, dependendo do sistema, o número de mapas deve ser ajustado a menor. A síntese de 160.000 mapas leva cerca de 20 minutos (Intel Xeon Hexacore 3GHz) com a compilação JIT (*just in time*).

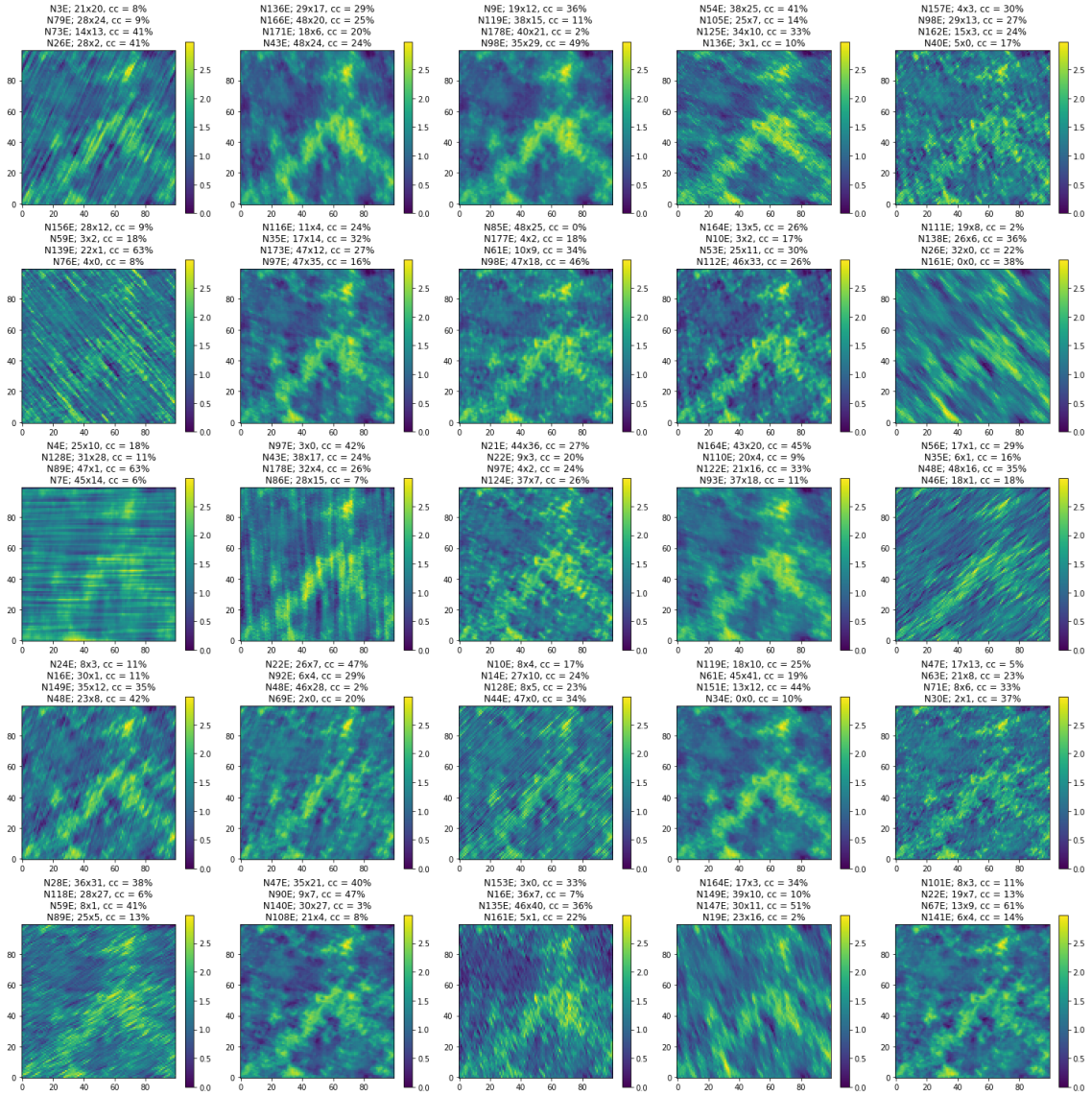


Figura 4-5: 25 mapas sintetizados com FIM para treinamento da CNN. Os azimutes, semieixos e contribuições de cada uma das quatro estruturas imbricadas geradas aleatoriamente estão nos títulos de cada mapa.

A totalidade de mapas sintéticos foi dividida em dois conjuntos: 75% para treinamento em si e 25% para o conjunto de validação. Não foi usado um conjunto de teste.

Esses mapas foram utilizados para treinar (em 20 épocas) uma CNN com a arquitetura 1 (Seção 4.1). A intenção é de que a rede aprenda a relação entre as formas presentes nos mapas e os parâmetros variográficos. A seguir, o relatório de treinamento da rede gerado pelo programa:

```
Train on 120000 samples, validate on 40000 samples
Epoch 1/20
120000/120000 [==============================] - 100s 830us/step - loss: 0.0519 - val_loss: 0.0509
Epoch 2/20
120000/120000 [==============================] - 100s 832us/step - loss: 0.0508 - val_loss: 0.0506
Epoch 3/20
120000/120000 [==============================] - 127s 1ms/step - loss: 0.0505 - val_loss: 0.0502
```

```
Epoch 4/20
120000/120000 [==============================] - 136s 1ms/step - loss: 0.0504 - val_loss: 0.0504
Epoch 5/20
120000/120000 [==============================] - 129s 1ms/step - loss: 0.0502 - val_loss: 0.0504
Epoch 6/20
120000/120000 [==============================] - 131s 1ms/step - loss: 0.0501 - val_loss: 0.0500
Epoch 7/20
120000/120000 [==============================] - 122s 1ms/step - loss: 0.0501 - val_loss: 0.0499
Epoch 8/20
120000/120000 [==============================] - 114s 949us/step - loss: 0.0500 - val_loss: 0.0500
Epoch 9/20
120000/120000 [==============================] - 126s 1ms/step - loss: 0.0500 - val_loss: 0.0501
Epoch 10/20
120000/120000 [==============================] - 125s 1ms/step - loss: 0.0499 - val_loss: 0.0499
Epoch 11/20
120000/120000 [==============================] - 110s 917us/step - loss: 0.0499 - val_loss: 0.0498
Epoch 12/20
120000/120000 [==============================] - 104s 867us/step - loss: 0.0498 - val_loss: 0.0501
Epoch 13/20
120000/120000 [==============================] - 111s 924us/step - loss: 0.0498 - val_loss: 0.0499
Epoch 14/20
120000/120000 [==============================] - 108s 899us/step - loss: 0.0498 - val_loss: 0.0497
Epoch 15/20
120000/120000 [==============================] - 128s 1ms/step - loss: 0.0498 - val_loss: 0.0498
Epoch 16/20
120000/120000 [==============================] - 117s 976us/step - loss: 0.0497 - val_loss: 0.0498
Epoch 17/20
120000/120000 [==============================] - 129s 1ms/step - loss: 0.0497 - val_loss: 0.0498
Epoch 18/20
120000/120000 [==============================] - 110s 915us/step - loss: 0.0497 - val_loss: 0.0500
Epoch 19/20
120000/120000 [==============================] - 118s 985us/step - loss: 0.0497 - val_loss: 0.0497
Epoch 20/20
120000/120000 [==============================] - 114s 947us/step - loss: 0.0496 - val_loss: 0.0498
```

O valor *loss* (função-objetivo) estabiliza rapidamente em um patamar de 0,05, de forma que um número maior de épocas não melhoraria significativamente a convergência.

Após o treinamento, que leva cerca de 40 minutos em um sistema com 12 núcleos lógicos de processamento (Intel Xeon Hexacore 3GHz), foi apresentado o mapa original para predição dos parâmetros variográficos. Este teste pode ser executado com o arquivo (Python Notebook) `Arch1_FIM.ipynb` (ver Seção 1.5 para o caminho completo).

Mais uma vez os resultados não foram aceitáveis (ver exemplo na Figura 4-6), embora por vezes com sucesso parcial (predizendo corretamente uma ou duas estruturas). Entretanto, também, os resultados variaram muito a cada execução, implicando uma sensibilidade forte à parte estocástica do algoritmo, o que compromete a consistência do método.

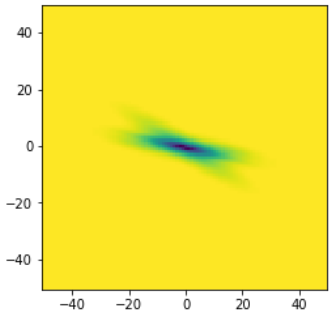


Figura 4-6: Modelo variográfico predito pela rede com a Arquitetura 1 treinada com mapas sintetizados com FIM para o caso desta tese.

4.2 Arquitetura 2

Tendo em vista os resultados insatisfatórios obtidos com a rede da Seção 4.1, faz-se necessário um método para avaliar a capacidade da rede de aprender a parametrização das imagens. Para isso, foi definida uma arquitetura de rede autocodificadora (Seção 2.5.5) a partir da parte convolutiva da Arquitetura 1.

A Figura 4-7 mostra a arquitetura usada nesta Seção. `UPSAMPLING` são as "camadas" que aumentam a definição da entrada (*upscaling*) por algum método como vizinho mais próximo, interpolação bilinear, etc. `ZERO PAD` são "camadas" que acrescentam linhas e/ou colunas à entrada e as preenchem com zeros. As demais camadas e elementos presentes na definição da arquitetura estão explicados na Seção 4.1.

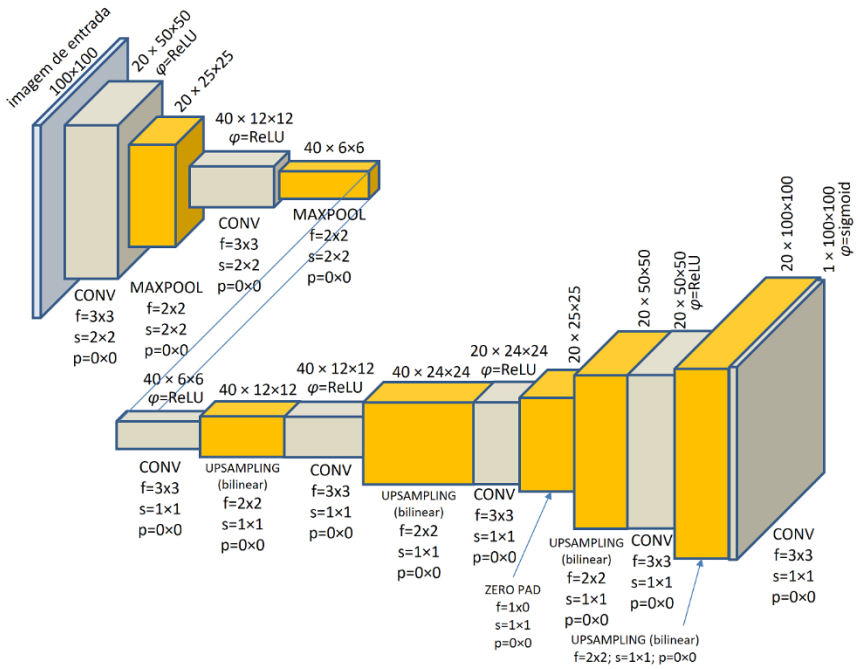


Figura 4-7: Arquitetura de rede 2. Ver texto.

4.2.1 Treinamento com superfícies variográficas

Nesta abordagem foram gerados 6.800 modelos variográficos aleatórios de quatro estruturas. A redução em 24 vezes no número de imagens para treinamento em relação à Seção 4.1.1 se deve ao tempo de treinamento que é 24 vezes maior por época: de 1ms para 24ms por iteração segundo os relatórios de treinamento. Essa redução foi necessária para viabilizar o experimento. A Figura 4-8 mostra dez resultados para dez entradas com a rede treinada em 20 épocas. A seguir, o relatório de treinamento mostrando a convergência em um patamar de 0,658 para função-objetivo:

```
Train on 5100 samples, validate on 1700 samples
Epoch 1/20
5100/5100 [==============================] - 119s 23ms/step - loss: 0.6685 - val_loss: 0.6638
Epoch 2/20
5100/5100 [==============================] - 133s 26ms/step - loss: 0.6631 - val_loss: 0.6622
Epoch 3/20
5100/5100 [==============================] - 138s 27ms/step - loss: 0.6619 - val_loss: 0.6616
Epoch 4/20
5100/5100 [==============================] - 135s 26ms/step - loss: 0.6614 - val_loss: 0.6610
Epoch 5/20
5100/5100 [==============================] - 133s 26ms/step - loss: 0.6609 - val_loss: 0.6608
Epoch 6/20
5100/5100 [==============================] - 133s 26ms/step - loss: 0.6607 - val_loss: 0.6606
Epoch 7/20
5100/5100 [==============================] - 133s 26ms/step - loss: 0.6605 - val_loss: 0.6606
Epoch 8/20
5100/5100 [==============================] - 132s 26ms/step - loss: 0.6602 - val_loss: 0.6601
Epoch 9/20
5100/5100 [==============================] - 142s 28ms/step - loss: 0.6601 - val_loss: 0.6598
Epoch 10/20
5100/5100 [==============================] - 134s 26ms/step - loss: 0.6598 - val_loss: 0.6601
Epoch 11/20
5100/5100 [==============================] - 133s 26ms/step - loss: 0.6597 - val_loss: 0.6599
Epoch 12/20
5100/5100 [==============================] - 133s 26ms/step - loss: 0.6595 - val_loss: 0.6596
Epoch 13/20
5100/5100 [==============================] - 133s 26ms/step - loss: 0.6592 - val_loss: 0.6590
Epoch 14/20
5100/5100 [==============================] - 133s 26ms/step - loss: 0.6589 - val_loss: 0.6588
Epoch 15/20
5100/5100 [==============================] - 125s 24ms/step - loss: 0.6585 - val_loss: 0.6582
Epoch 16/20
5100/5100 [==============================] - 127s 25ms/step - loss: 0.6583 - val_loss: 0.6581
Epoch 17/20
5100/5100 [==============================] - 123s 24ms/step - loss: 0.6580 - val_loss: 0.6578
Epoch 18/20
5100/5100 [==============================] - 127s 25ms/step - loss: 0.6579 - val_loss: 0.6578
Epoch 19/20
5100/5100 [==============================] - 125s 25ms/step - loss: 0.6578 - val_loss: 0.6577
Epoch 20/20
5100/5100 [==============================] - 129s 25ms/step - loss: 0.6577 - val_loss: 0.6577
```

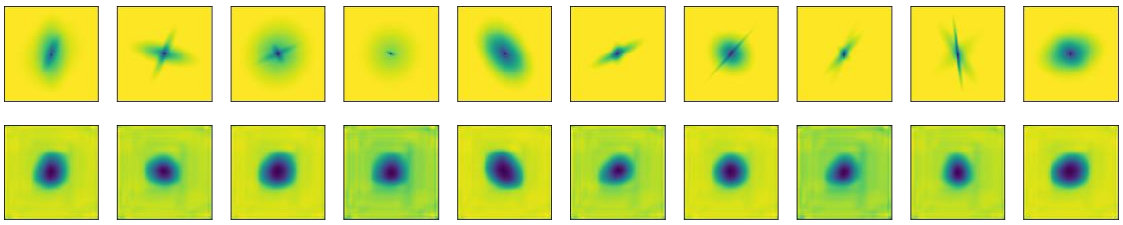


Figura 4-8: Emprego de uma rede autocodificadora. A fileira superior contém dez imagens de entrada (modelos variográficos) e a inferior, dez tentativas de reconstruí-las pela rede.

A julgar pelos resultados, não surpreende a predição ruim dos parâmetros variográficos para o dado de teste feita na Seção 4.1.1. Esse resultado parece sugerir que detalhes de alta frequência estão sendo perdidos. De fato, o número de núcleos na primeira camada convolutiva é pequeno (20), o que pode contribuir para tal efeito adverso. A primeira camada, conforme apresentado na Seção 2.5.4, captura os elementos de menor escala na imagem. Contudo, a perda de detalhes parece ser intrínseca ao emprego de redes autocodificadoras (Theis *et al.* 2017), não significando necessariamente que o treinamento da parte convolutiva esteja ruim.

O experimento pode ser realizado com o arquivo (Python Notebook) `Arch2_Varmap.ipynb` (ver Seção 1.5 para o caminho completo). Com esses resultados ruins, não foi feita a conexão da parte convolutiva a uma MLP para predição de parâmetros variográficos.

4.2.2 Treinamento com mapas sintetizados com FIM

Nesta abordagem foram sintetizados 6.800 mapas com FIM (Seção 2.4) a partir de modelos variográficos aleatórios de quatro estruturas e do mesmo mapa de fases do dado original. A Figura 4-9 mostra dez resultados para dez entradas com a rede treinada em 20 épocas. A seguir, o relatório de treinamento mostrando a convergência para um patamar de 0,653 para a função-objetivo:

```
Train on 5100 samples, validate on 1700 samples
Epoch 1/20
5100/5100 [==============================] - 122s 24ms/step - loss: 0.6831 - val_loss: 0.6772
Epoch 2/20
5100/5100 [==============================] - 122s 24ms/step - loss: 0.6766 - val_loss: 0.6713
Epoch 3/20
5100/5100 [==============================] - 126s 25ms/step - loss: 0.6722 - val_loss: 0.6712
Epoch 4/20
5100/5100 [==============================] - 137s 27ms/step - loss: 0.6687 - val_loss: 0.6696
Epoch 5/20
5100/5100 [==============================] - 122s 24ms/step - loss: 0.6658 - val_loss: 0.6651
Epoch 6/20
5100/5100 [==============================] - 121s 24ms/step - loss: 0.6635 - val_loss: 0.6620
Epoch 7/20
5100/5100 [==============================] - 129s 25ms/step - loss: 0.6614 - val_loss: 0.6645
Epoch 8/20
5100/5100 [==============================] - 129s 25ms/step - loss: 0.6603 - val_loss: 0.6620
Epoch 9/20
5100/5100 [==============================] - 139s 27ms/step - loss: 0.6587 - val_loss: 0.6581
Epoch 10/20
5100/5100 [==============================] - 136s 27ms/step - loss: 0.6576 - val_loss: 0.6589
Epoch 11/20
5100/5100 [==============================] - 130s 26ms/step - loss: 0.6567 - val_loss: 0.6551
Epoch 12/20
5100/5100 [==============================] - 122s 24ms/step - loss: 0.6553 - val_loss: 0.6557
Epoch 13/20
5100/5100 [==============================] - 129s 25ms/step - loss: 0.6550 - val_loss: 0.6532
Epoch 14/20
5100/5100 [==============================] - 120s 24ms/step - loss: 0.6541 - val_loss: 0.6555
Epoch 15/20
5100/5100 [==============================] - 125s 24ms/step - loss: 0.6534 - val_loss: 0.6514
Epoch 16/20
5100/5100 [==============================] - 126s 25ms/step - loss: 0.6528 - val_loss: 0.6511
Epoch 17/20
5100/5100 [==============================] - 120s 24ms/step - loss: 0.6525 - val_loss: 0.6522
Epoch 18/20
```

```
5100/5100 [==============================] - 120s 23ms/step - loss: 0.6519 - val_loss: 0.6513
Epoch 19/20
5100/5100 [==============================] - 120s 24ms/step - loss: 0.6516 - val_loss: 0.6522
Epoch 20/20
5100/5100 [==============================] - 120s 23ms/step - loss: 0.6515 - val_loss: 0.6508
```

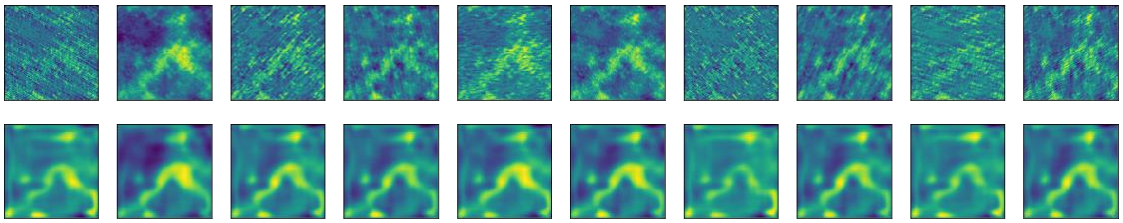


Figura 4-9: Emprego de uma rede autocodificadora. A fileira superior contém dez imagens de entrada (mapas sintetizados com FIM) e a inferior, dez tentativas de reconstruí-las pela rede.

Segundo esses resultados, a rede só conseguiu aprender a parametrização das feições mais importantes encontradas nos mapas, o que confirma a suspeita de que a arquitetura de rede proposta na Seção 4.1 não está bem dimensionada.

O experimento pode ser realizado com o arquivo (Python Notebook) `Arch2_FIM.ipynb` (ver Seção 1.5 para o caminho completo). Mais uma vez, com esses resultados insatisfatórios, não foi feita a conexão da parte convolutiva a uma MLP para predição de parâmetros variográficos.

4.3 Arquitetura 3

Considerando os resultados insatisfatórios obtidos com a rede da Seção 4.2, a parte codificadora foi expandida de forma que a redução da informação espacial só possa ser feita pelas etapas convolutivas (que aprendem) em cada camada e não pelas etapas do tipo `MAXPOOL` (que não aprendem). O mesmo vale para o aumento de feições (*features* – ou número de imagens) em cada camada. E a parte decodificadora é simplesmente simétrica em relação à sua contraparte.

A Figura 4-10 mostra a arquitetura usada nesta Seção. `CROP` é uma "camada" que remove linhas e/ou colunas da entrada. As demais camadas e elementos presentes na definição da arquitetura estão explicados nas Seções 4.1 e 4.2.

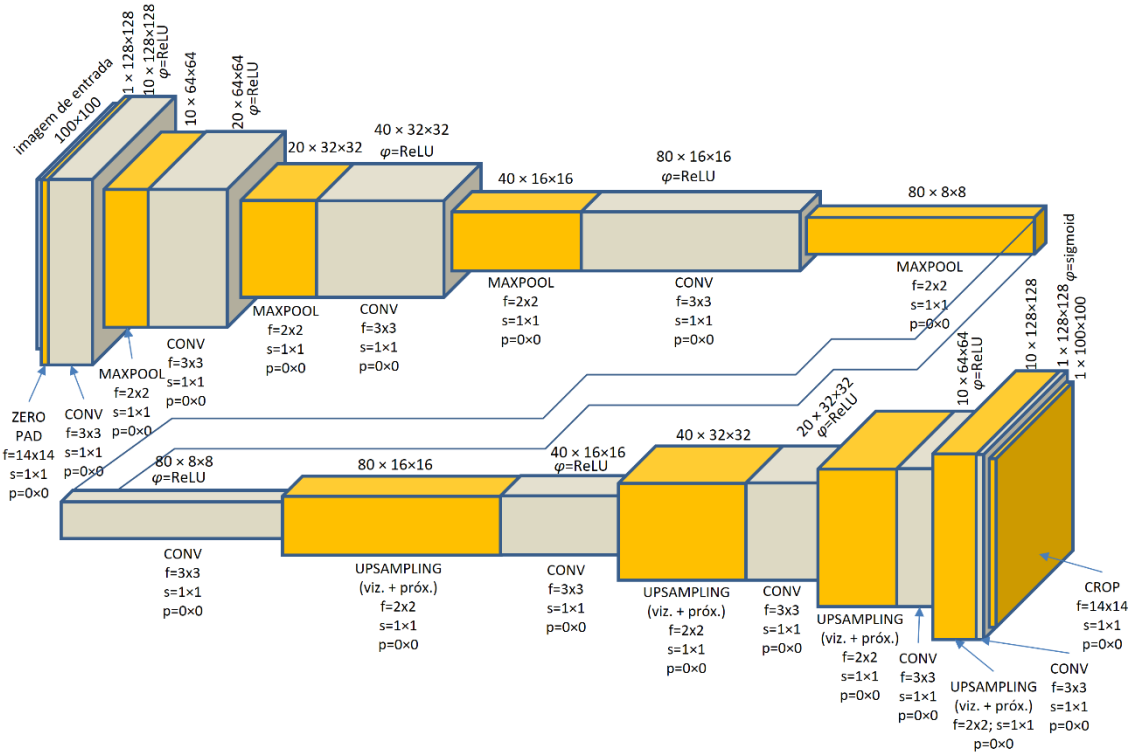


Figura 4-10: Arquitetura de rede 3. Ver texto.

4.3.1 Treinamento com superfícies variográficas

Nesta abordagem foram gerados 8.000 modelos variográficos aleatórios de quatro estruturas. A redução em 20 vezes no número de imagens para treinamento em relação à Seção 4.1.1 se deve a aumento equivalente de tempo de treinamento para viabilizar o experimento. A Figura 4-8 mostra dez resultados para dez entradas com a rede treinada em 20 épocas. A seguir, o relatório de treinamento mostrando a convergência em um patamar de 0,658 para função-objetivo:

```
Train on 6000 samples, validate on 2000 samples
Epoch 1/20
6000/6000 [==============================] - 122s 20ms/step - loss: 0.6643 - val_loss: 0.6615
Epoch 2/20
6000/6000 [==============================] - 149s 25ms/step - loss: 0.6617 - val_loss: 0.6615
Epoch 3/20
6000/6000 [==============================] - 153s 25ms/step - loss: 0.6613 - val_loss: 0.6609
Epoch 4/20
6000/6000 [==============================] - 152s 25ms/step - loss: 0.6610 - val_loss: 0.6610
Epoch 5/20
6000/6000 [==============================] - 152s 25ms/step - loss: 0.6607 - val_loss: 0.6604
Epoch 6/20
6000/6000 [==============================] - 151s 25ms/step - loss: 0.6603 - val_loss: 0.6600
Epoch 7/20
6000/6000 [==============================] - 152s 25ms/step - loss: 0.6599 - val_loss: 0.6595
Epoch 8/20
6000/6000 [==============================] - 155s 26ms/step - loss: 0.6594 - val_loss: 0.6591
Epoch 9/20
6000/6000 [==============================] - 156s 26ms/step - loss: 0.6590 - val_loss: 0.6586
Epoch 10/20
6000/6000 [==============================] - 156s 26ms/step - loss: 0.6586 - val_loss: 0.6582
Epoch 11/20
6000/6000 [==============================] - 154s 26ms/step - loss: 0.6583 - val_loss: 0.6583
Epoch 12/20
```

```
6000/6000 [==============================] - 154s 26ms/step - loss: 0.6580 - val_loss: 0.6578
Epoch 13/20
6000/6000 [==============================] - 153s 26ms/step - loss: 0.6578 - val_loss: 0.6580
Epoch 14/20
6000/6000 [==============================] - 153s 26ms/step - loss: 0.6577 - val_loss: 0.6577
Epoch 15/20
6000/6000 [==============================] - 152s 25ms/step - loss: 0.6576 - val_loss: 0.6576
Epoch 16/20
6000/6000 [==============================] - 152s 25ms/step - loss: 0.6576 - val_loss: 0.6575
Epoch 17/20
6000/6000 [==============================] - 154s 26ms/step - loss: 0.6576 - val_loss: 0.6576
Epoch 18/20
6000/6000 [==============================] - 156s 26ms/step - loss: 0.6575 - val_loss: 0.6574
Epoch 19/20
6000/6000 [==============================] - 156s 26ms/step - loss: 0.6575 - val_loss: 0.6575
Epoch 20/20
6000/6000 [==============================] - 154s 26ms/step - loss: 0.6574 - val_loss: 0.6576
```

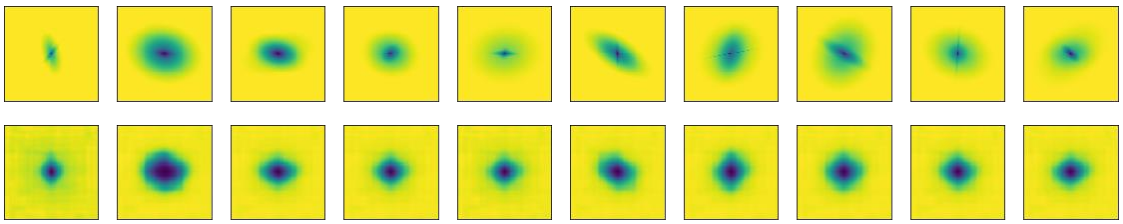


Figura 4-11: Emprego de uma rede autocodificadora. A fileira superior contém dez imagens de entrada (modelos variográficos) e a inferior, dez tentativas de reconstruí-las pela rede.

Observa-se uma melhora na reconstrução em relação ao observado na Figura 4-8. Não obstante, esse resultado parece confirmar que detalhes de alta frequência estão sendo perdidos. O experimento pode ser realizado com o arquivo (Python Notebook) `Arch3_Varmap.ipynb` (ver Seção 1.5 para o caminho completo). Com esses resultados ainda insatisfatórios, não foi feita a conexão da parte convolutiva a uma MLP para predição de parâmetros variográficos.

4.3.2 Treinamento com mapas sintetizados com FIM

Nesta abordagem foram sintetizados 8.000 mapas com FIM (Seção 2.4) a partir de modelos variográficos aleatórios de quatro estruturas e do mesmo mapa de fases do dado original. A Figura 4-12 mostra dez resultados para dez entradas com a rede treinada em 20 épocas. A seguir, o relatório de treinamento mostrando a convergência para um patamar de 0,646 para a função-objetivo:

```
Train on 6000 samples, validate on 2000 samples
Epoch 1/20
6000/6000 [==============================] - 140s 23ms/step - loss: 0.6837 - val_loss: 0.6829
Epoch 2/20
6000/6000 [==============================] - 154s 26ms/step - loss: 0.6733 - val_loss: 0.6664
Epoch 3/20
6000/6000 [==============================] - 154s 26ms/step - loss: 0.6636 - val_loss: 0.6606
Epoch 4/20
6000/6000 [==============================] - 154s 26ms/step - loss: 0.6586 - val_loss: 0.6561
Epoch 5/20
6000/6000 [==============================] - 153s 25ms/step - loss: 0.6550 - val_loss: 0.6525
Epoch 6/20
6000/6000 [==============================] - 156s 26ms/step - loss: 0.6529 - val_loss: 0.6514
Epoch 7/20
```

```
6000/6000 [==============================] - 157s 26ms/step - loss: 0.6516 - val_loss: 0.6505
Epoch 8/20
6000/6000 [==============================] - 156s 26ms/step - loss: 0.6504 - val_loss: 0.6491
Epoch 9/20
6000/6000 [==============================] - 155s 26ms/step - loss: 0.6495 - val_loss: 0.6495
Epoch 10/20
6000/6000 [==============================] - 154s 26ms/step - loss: 0.6488 - val_loss: 0.6493
Epoch 11/20
6000/6000 [==============================] - 154s 26ms/step - loss: 0.6484 - val_loss: 0.6478
Epoch 12/20
6000/6000 [==============================] - 153s 25ms/step - loss: 0.6478 - val_loss: 0.6469
Epoch 13/20
6000/6000 [==============================] - 153s 26ms/step - loss: 0.6477 - val_loss: 0.6476
Epoch 14/20
6000/6000 [==============================] - 153s 25ms/step - loss: 0.6471 - val_loss: 0.6470
Epoch 15/20
6000/6000 [==============================] - 152s 25ms/step - loss: 0.6468 - val_loss: 0.6468
Epoch 16/20
6000/6000 [==============================] - 152s 25ms/step - loss: 0.6463 - val_loss: 0.6469
Epoch 17/20
6000/6000 [==============================] - 154s 26ms/step - loss: 0.6463 - val_loss: 0.6464
Epoch 18/20
6000/6000 [==============================] - 153s 25ms/step - loss: 0.6458 - val_loss: 0.6454
Epoch 19/20
6000/6000 [==============================] - 152s 25ms/step - loss: 0.6456 - val_loss: 0.6469
Epoch 20/20
6000/6000 [==============================] - 152s 25ms/step - loss: 0.6454 - val_loss: 0.6453
```

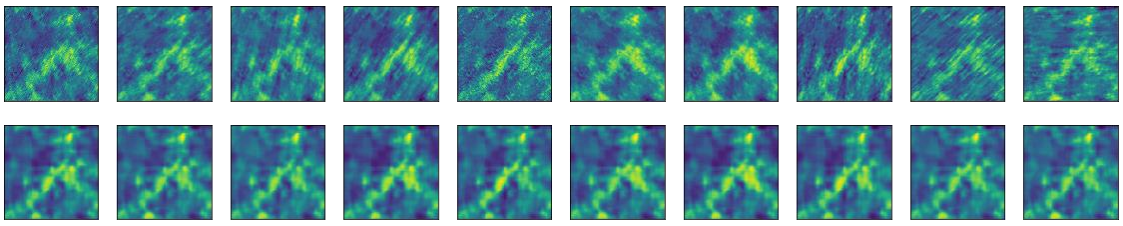


Figura 4-12: Emprego de uma rede autocodificadora. A fileira superior contém dez imagens de entrada (mapas sintetizados com FIM) e a inferior, dez tentativas de reconstruí-las pela rede.

Em relação aos resultados mostrados na Figura 4-9, houve aumento na quantidade de detalhes que a rede aprendeu a reconstruir, embora esses resultados ainda sejam insatisfatórios, pois os artefatos ainda não aparecem nas reconstruções. Se a rede não reconhece os artefatos, não se pode esperar que a mesma consiga obter os seus parâmetros variográficos em uma posterior substituição da parte decodificadora por uma rede MLP para estimação desses parâmetros. Não obstante, esses resultados podem ser úteis caso a intenção seja meramente filtrar a imagem de ruídos sem entrar no mérito de caracterizá-los.

O experimento pode ser realizado com o arquivo (Python Notebook) `Arch3_FIM.ipynb` (ver Seção 1.5 para o caminho completo). Mais uma vez, com esses resultados ainda insatisfatórios, não foi feita a conexão da parte convolutiva a uma MLP para predição de parâmetros variográficos.

4.4 Arquitetura 4

Apesar da melhoria considerável dos resultados da arquitetura da Seção 4.3, ainda há perda de feições de alta frequência. Conforme visto na Seção 2.5.6, é possível preservar a

informação de alta resolução através da rede. Para isso, foi definida uma arquitetura de rede U a partir da parte convolutiva da Arquitetura 3. A Figura 4-14 mostra a arquitetura usada nesta Seção (notar o formato em U deitado).

CONVTRAN (convolução transposta[23]) é uma camada que, tal como a UPSAMPLING, aumenta a definição da imagem de entrada (*upscaling*), porém é uma camada convolutiva, isto é, aprende a melhor forma de fazer o *upscaling* ao invés de empregar um algoritmo tradicional como Vizinho Mais Próximo ou Interpolação Bilinear. A convolução transposta é um artifício computacional para obter a convolução de uma imagem tal que resulte em uma imagem com maior contagem de pixels. Nessa operação, a imagem de entrada é expandida (*padded*) de alguma forma e procede-se com a convolução normal. O leitor deve estar atento na literatura às referências a essa operação como deconvolução, mas na verdade não é. Como a imagem de saída é maior do que a de entrada, a convolução transposta pode ser vista como uma convolução com *stride* fracionário. A Figura 4-13 ilustra o funcionamento da convolução transposta.

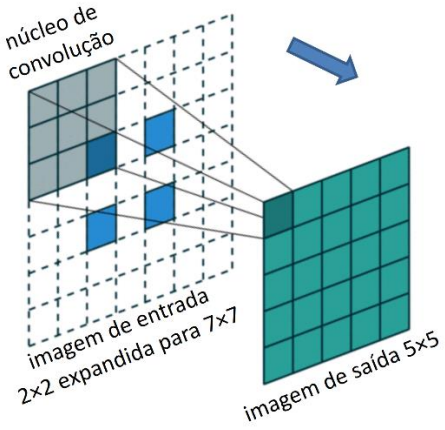


Figura 4-13: Esquema de funcionamento da convolução transposta.

CONCAT (concatenar) é uma operação que empilha dois tensores de mesmo número de linhas e de colunas. Por exemplo, se suas entradas são 8 e 10 imagens 16×16, sua saída será um único tensor com 18 imagens 16×16. Na arquitetura vê-se que as camadas CONV na parte decodificadora têm duas entradas, o que não é possível de implementar. Assim, CONCAT é usada para empilhar dois (ou mais) tensores de forma a possibilitar a rede U de ser

---

23 https://medium.com/activating-robotic-minds/up-sampling-with-transposed-convolution-9ae4f2df52d0 (acessado em 16/11/2019)

implementável em software. As demais camadas e elementos presentes na definição da arquitetura estão explicados nas Seções 4.1, 4.2 e 4.3.

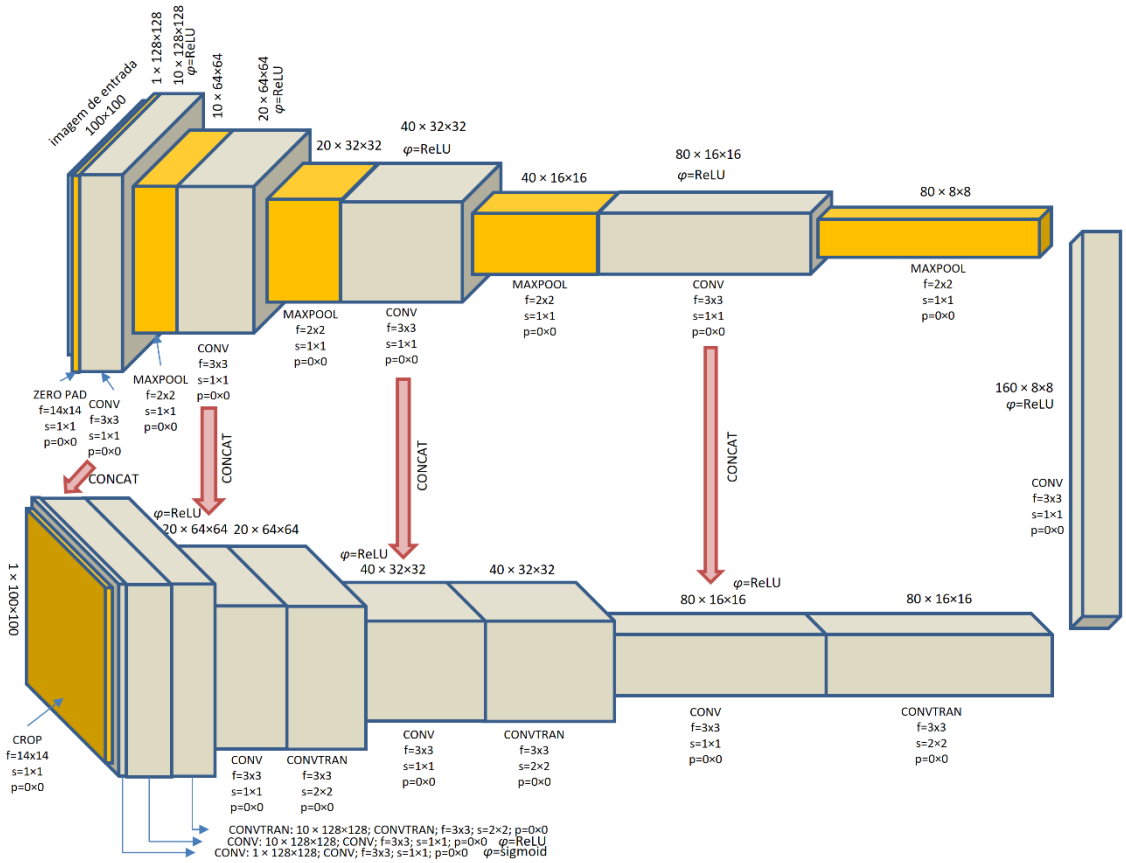


Figura 4-14: Arquitetura de rede 4. Ver texto.

4.4.1 Treinamento com superfícies variográficas

Nesta abordagem foram gerados 5.500 modelos variográficos aleatórios de quatro estruturas. Conforme já visto para as demais arquiteturas a redução no número de imagens de treinamento em relação à Seção 4.1.1 se deve a aumento de tempo de treinamento para viabilizar o experimento. A Figura 4-15 mostra dez resultados para dez entradas com a rede treinada em 20 épocas. A seguir, o relatório de treinamento mostrando a convergência em um patamar de 0,658 para função-objetivo:

```
Train on 4125 samples, validate on 1375 samples
Epoch 1/20
4125/4125 [==============================] - 148s 36ms/step - loss: 0.6653 - val_loss: 0.6608
Epoch 2/20
4125/4125 [==============================] - 176s 43ms/step - loss: 0.6607 - val_loss: 0.6607
Epoch 3/20
4125/4125 [==============================] - 172s 42ms/step - loss: 0.6606 - val_loss: 0.6605
Epoch 4/20
4125/4125 [==============================] - 174s 42ms/step - loss: 0.6604 - val_loss: 0.6603
Epoch 5/20
4125/4125 [==============================] - 171s 41ms/step - loss: 0.6602 - val_loss: 0.6606
Epoch 6/20
4125/4125 [==============================] - 170s 41ms/step - loss: 0.6603 - val_loss: 0.6604
Epoch 7/20
```

```
4125/4125 [==============================] - 170s 41ms/step - loss: 0.6605 - val_loss: 0.6601
Epoch 8/20
4125/4125 [==============================] - 170s 41ms/step - loss: 0.6604 - val_loss: 0.6606
Epoch 9/20
4125/4125 [==============================] - 169s 41ms/step - loss: 0.6602 - val_loss: 0.6608
Epoch 10/20
4125/4125 [==============================] - 169s 41ms/step - loss: 0.6602 - val_loss: 0.6599
Epoch 11/20
4125/4125 [==============================] - 171s 41ms/step - loss: 0.6595 - val_loss: 0.6591
Epoch 12/20
4125/4125 [==============================] - 169s 41ms/step - loss: 0.6593 - val_loss: 0.6596
Epoch 13/20
4125/4125 [==============================] - 169s 41ms/step - loss: 0.6593 - val_loss: 0.6592
Epoch 14/20
4125/4125 [==============================] - 169s 41ms/step - loss: 0.6591 - val_loss: 0.6588
Epoch 15/20
4125/4125 [==============================] - 171s 41ms/step - loss: 0.6589 - val_loss: 0.6584
Epoch 16/20
4125/4125 [==============================] - 169s 41ms/step - loss: 0.6586 - val_loss: 0.6586
Epoch 17/20
4125/4125 [==============================] - 169s 41ms/step - loss: 0.6585 - val_loss: 0.6579
Epoch 18/20
4125/4125 [==============================] - 169s 41ms/step - loss: 0.6582 - val_loss: 0.6584
Epoch 19/20
4125/4125 [==============================] - 169s 41ms/step - loss: 0.6582 - val_loss: 0.6586
Epoch 20/20
4125/4125 [==============================] - 176s 43ms/step - loss: 0.6582 - val_loss: 0.6585
```

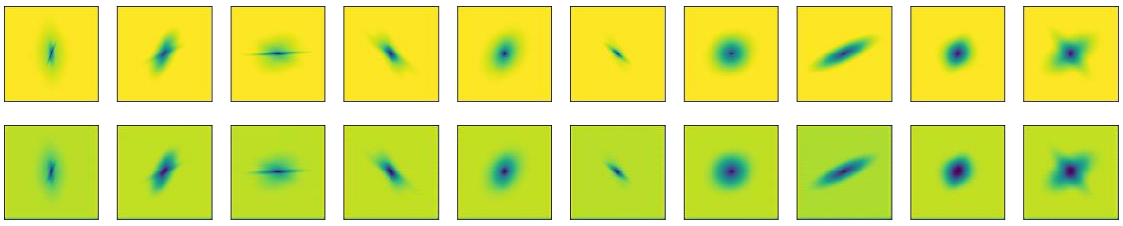


Figura 4-15: Emprego de uma rede U. A fileira superior contém dez imagens de entrada (modelos variográficos) e a inferior, dez tentativas de reconstruí-las pela rede.

É evidente que essa arquitetura aprendeu a codificar as imagens que representam modelos variográficos teóricos sem perdas perceptíveis. Ao apresentar o mapa variográfico experimental do dado, a rede treinada consegue reconstruí-lo conforme mostrado na Figura 4-16 (pode haver variações por causa da componente estocástica do algoritmo de otimização). Pode-se afirmar que a figura resultante é como a rede enxerga o mapa variográfico que lhe foi apresentado.

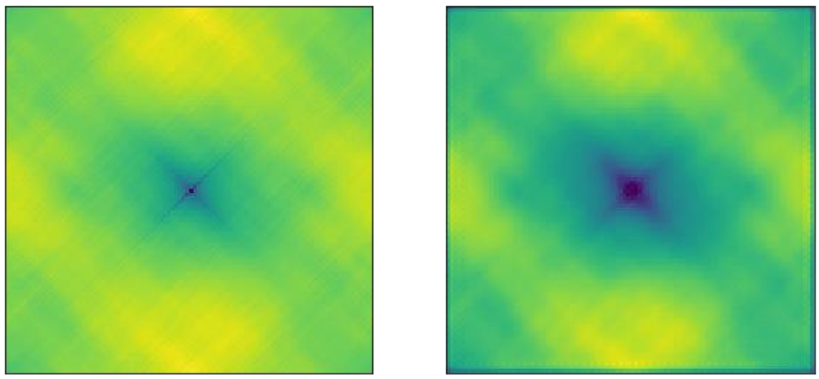


Figura 4-16: À esquerda, o mapa variográfico experimental do dado desta tese e, à direita, a sua reconstituição feita pela rede U.

Em seguida, a parte codificadora da rede U, já treinada, foi conectada a uma parte totalmente conectada (rede MLP). A parte MLP foi treinada para uma tarefa de regressão com vistas a obter os parâmetros variográficos correspondentes a um mapa variográfico apresentado na entrada. A arquitetura modificada da rede está ilustrada na Figura 4-17.

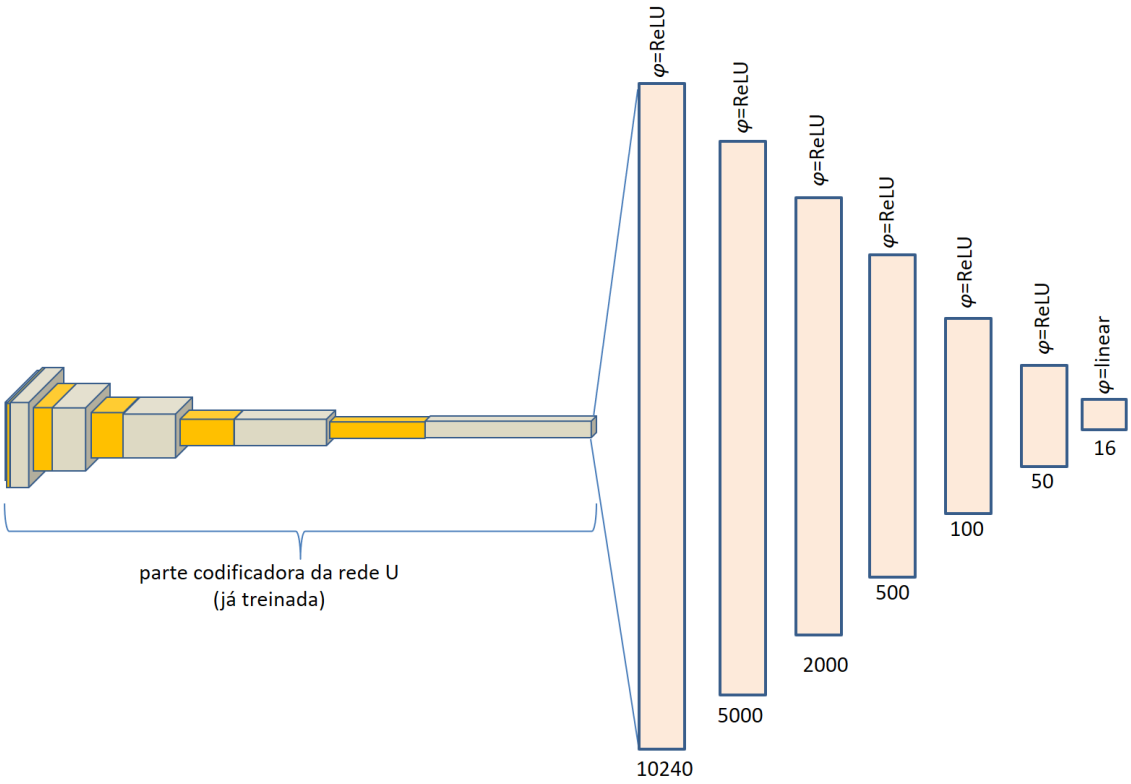


Figura 4-17: Arquitetura de rede formada pela parte codificadora da arquitetura da Figura 4-14 e uma parte MLP para regressão (predição dos 16 parâmetros variográficos a partir de um mapa variográfico de entrada).

O experimento pode ser realizado com o arquivo (Python Notebook) `Arch4_Varmap.ipynb` (ver Seção 1.5 para o caminho completo). Mesmo com a rede U, capaz de codificar uma imagem completamente, os resultados da predição de parâmetros ainda foram insatisfatórios.

Mesmo apresentando os mapas variográficos sintéticos usados no treinamento, a rede não foi capaz de predizer os parâmetros. Aliás, todas as predições variaram pouco em relação a uma média, ilustrada na Figura 4-18.

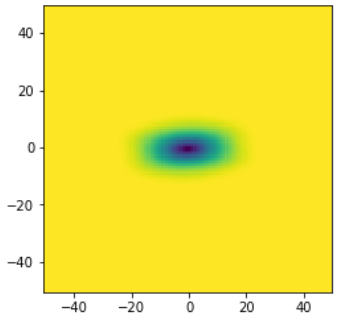


Figura 4-18: Um modelo variográfico obtido da predição da arquitetura da Figura 4-17. Para qualquer mapa variográfico de entrada, a predição era muito semelhante a este.

4.4.2 Treinamento com mapas sintetizados com FIM

Nesta abordagem foram sintetizados 5.500 mapas com FIM (Seção 2.4) a partir de modelos variográficos aleatórios de quatro estruturas e do mesmo mapa de fases do dado original. A Figura 4-12 mostra dez resultados para dez entradas com a rede treinada em 20 épocas. Conforme se observa, qualitativamente os mapas não somente foram reconstituídos com os detalhes de alta frequência, mas como apresentam maior contraste de valores. Isso sugere que a rede aprendeu uma grande quantidade de feições que podem compor satisfatoriamente todos os possíveis mapas com todos os possíveis variogramas.

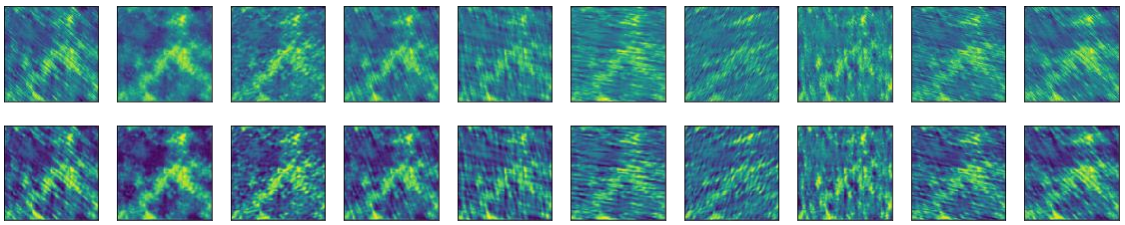


Figura 4-19: Emprego de uma rede U. A fileira superior contém dez imagens de entrada (mapas sintetizados com FIM) e a inferior, dez tentativas de reconstruí-las pela rede.

Quando o mapa original foi apresentado à rede, a sua reconstituição pela rede está mostrada na Figura 4-20. Embora esse resultado em si não seja fiel ao mapa original, as feições de alta frequência foram reproduzidas. Lembrando que o importante nesta etapa é que a rede esteja conseguindo enxergar bem o mapa de entrada, pois o objetivo é obter os parâmetros variográficos em outra etapa de treinamento.

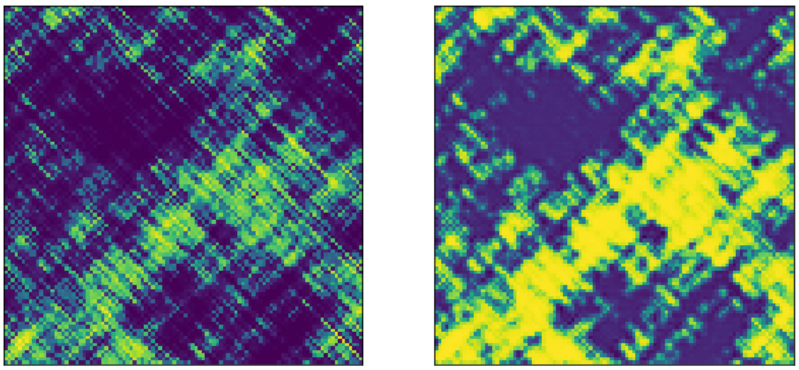


Figura 4-20: À esquerda, o mapa do dado desta tese e, à direita, a sua reconstituição feita pela rede U.

Assim como no treinamento com superfícies variográficas, a parte codificadora da arquitetura, já treinada, foi ligada a uma parte MLP, formando uma a arquitetura ilustrada na Figura 4-17. Apesar da reconstituição razoável das feições pela rede, as predições dos parâmetros variográficos foram ruins. No entanto, diferentemente do treinamento com os mapas variográficos, as predições foram variáveis, tal como mostra a Figura 4-21.

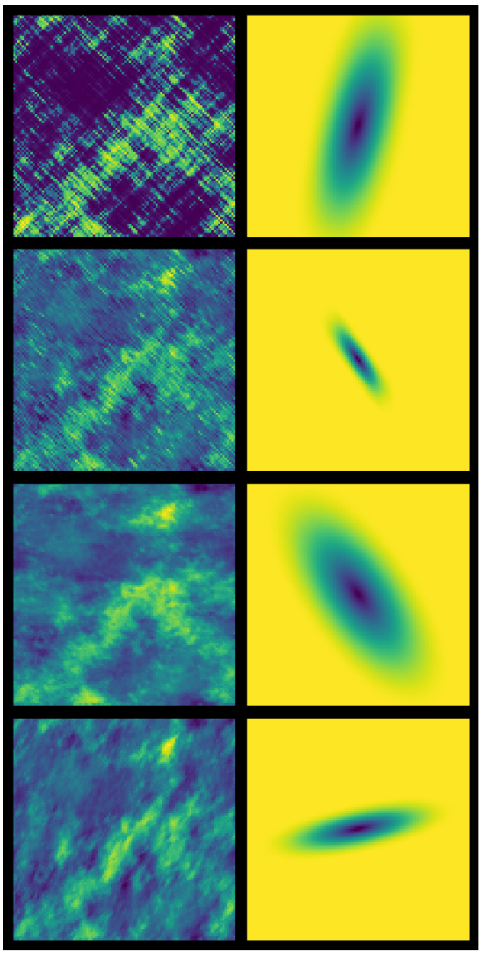


Figura 4-21: Alguns mapas e seus respectivos modelos variográficos obtidos por predição com a arquitetura da Figura 4-17. O mapa no topo é o dado de teste desta tese.

Há uma aparente contradição entre a boa reconstituição das imagens pela rede U e a predição ruim dos parâmetros variográficos. Isso pode ser explicado pelas conexões diretas entre camadas não adjacentes, o que pode resultar em perda da capacidade de uma rede autocodificadora em construir uma representação abstrata das feições das imagens. Conforme visto na Seção 2.5.3, quanto mais profunda a camada, mais abstrata é a representação da entrada. O que acontece em uma rede U é que uma parte importante da informação pode estar sendo desviada das camadas mais profundas durante o treinamento, resultando em camadas menos profundas mais treinadas do que as mais profundas. Assim, ao conectar a parte codificadora a uma MLP, o resultado é uma predição ruim dos parâmetros variográficos. Em resumo, a arquitetura U parece melhor adequada a predições do tipo imagem-imagem do que a predições imagem-escalares. O experimento pode ser realizado com o arquivo (Python Notebook) `Arch4_FIM.ipynb` (ver Seção 1.5 para o caminho completo).

4.5 Arquitetura 5

Conforme visto na Seção 4.4, a Arquitetura 4 apresentou um desvio de informação das suas camadas mais profundas de forma que prejudicou a previsão dos parâmetros variográficos. Para contornar este problema, as conexões de *bypass* da rede U foram removidas e as camadas convolutivas mais profundas, substituídas por uma MLP. A Figura 4-22 mostra a nova arquitetura.

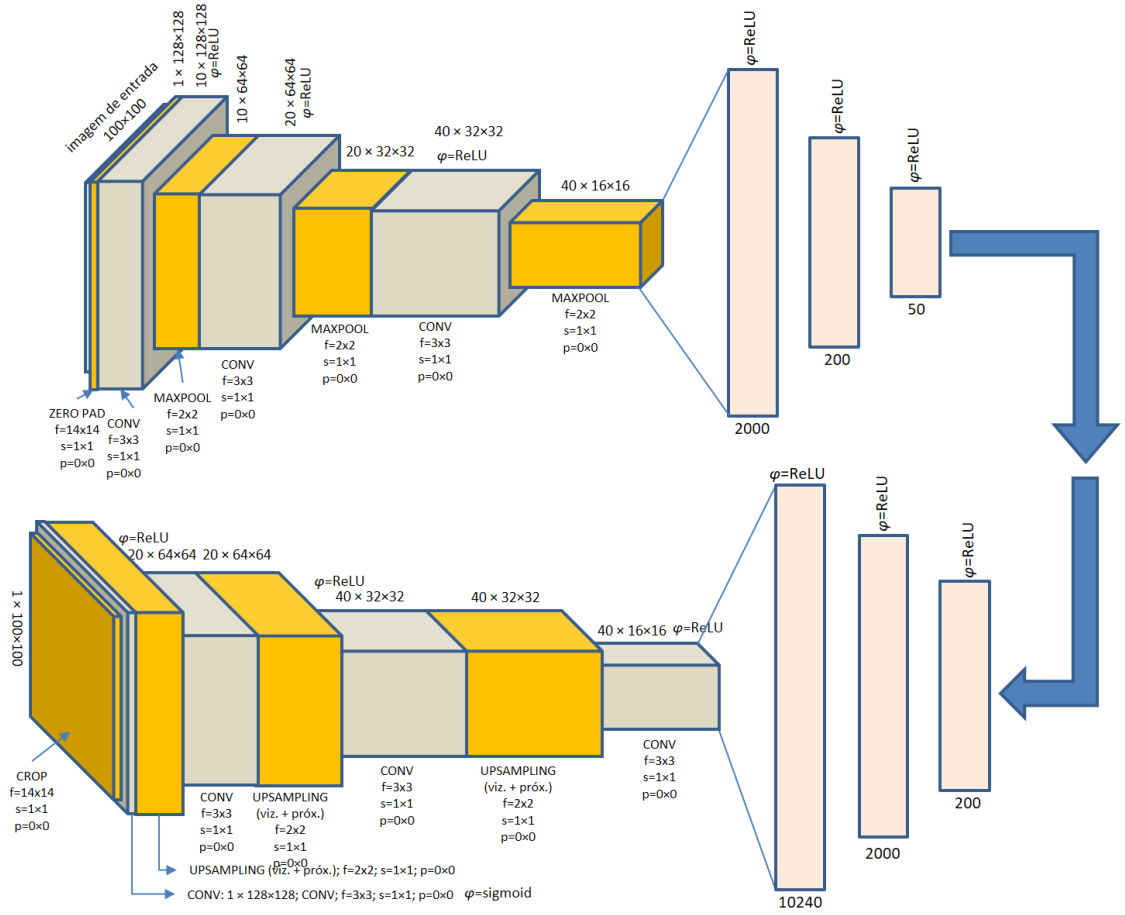


Figura 4-22: Arquitetura autocodificadora com uma rede MLP no lugar de camadas convolutivas profundas.

4.5.1 Treinamento com superfícies variográficas

Após treinada em 20 épocas com 5.500 modelos variográficos aleatórios, o resultado para dez deles está mostrado na Figura 4-23. Observa-se, mesmo sem as conexões diretas de uma rede U, que as reconstruções são de boa qualidade, no entanto chama atenção o fato de que as reconstruções são totalmente simétricas. Uma explicação para isso reside no fato de que as estruturas espaciais se perdem nas camadas não-convolutivas introduzidas no meio da arquitetura (camadas mais profundas). O experimento pode ser realizado com o arquivo (Python Notebook) `Arch5_Varmap.ipynb` (ver Seção 1.5 para o caminho completo).

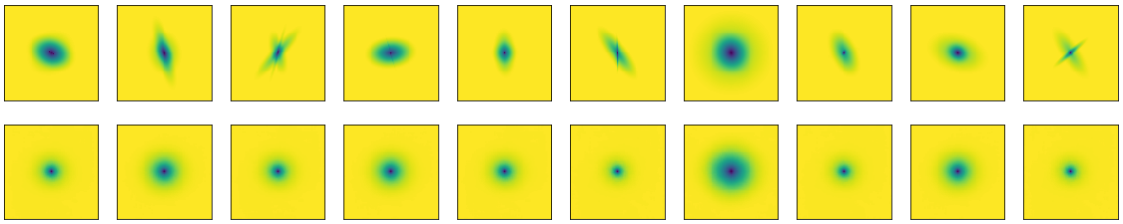


Figura 4-23: Emprego da Arquitetura 5. A fileira superior contém dez imagens de entrada (modelos variográficos) e a inferior, dez tentativas de reconstruí-las pela rede.

4.5.2 Treinamento com dados sintetizados com FIM

O treinamento com dados sintetizados com FIM (Seção 2.4) a partir de variogramas aleatórios resultou em reconstituições muito semelhantes às da Arquitetura 3 (Figura 4-12), ou seja, sem melhora. A Figura 4-24 mostra o resultado para dez mapas sintéticos. Como a reconstrução é praticamente invariável, não seria possível predizer os parâmetros variográficos a partir do mapa original. O experimento pode ser realizado com o arquivo (Python Notebook) `Arch5_FIM.ipynb` (ver Seção 1.5 para o caminho completo).

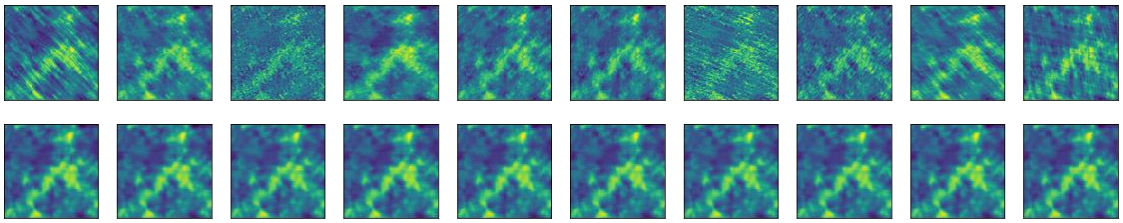


Figura 4-24: Emprego da Arquitetura 5. A fileira superior contém dez imagens de entrada (mapas sintetizados com FIM) e a inferior, dez tentativas de reconstruí-las pela rede.

4.6 Arquitetura 4 para decomposição direta

Considerando o sucesso da rede U em reconstruir imagens, a Arquitetura 4 (Seção 4.4) foi modificada para que as imagens de entrada e de saída, ao invés de serem 100×100×1, sejam volumes com 4 camadas de 100×100 células. Também, as camadas convolutivas passam a ter 10 vezes mais núcleos (ex.: de 20 para 200) para acomodar a maior dimensionalidade das imagens.

As imagens de entrada foram feitas como sendo apenas as imagens originais 100×100×1 repetidas quatro vezes para formar um volume 100×100×4. Já os volumes de saída para treinamento são uma pilha com cada uma das quatro estruturas imbricadas em separado ao invés de somadas como feito até aqui. A intenção é que a rede aprenda uma decomposição estrutural diretamente ao invés de tentar aprender uma relação entre formas e parâmetros variográficos. Esse arranjo está ilustrado na Figura 4-25.

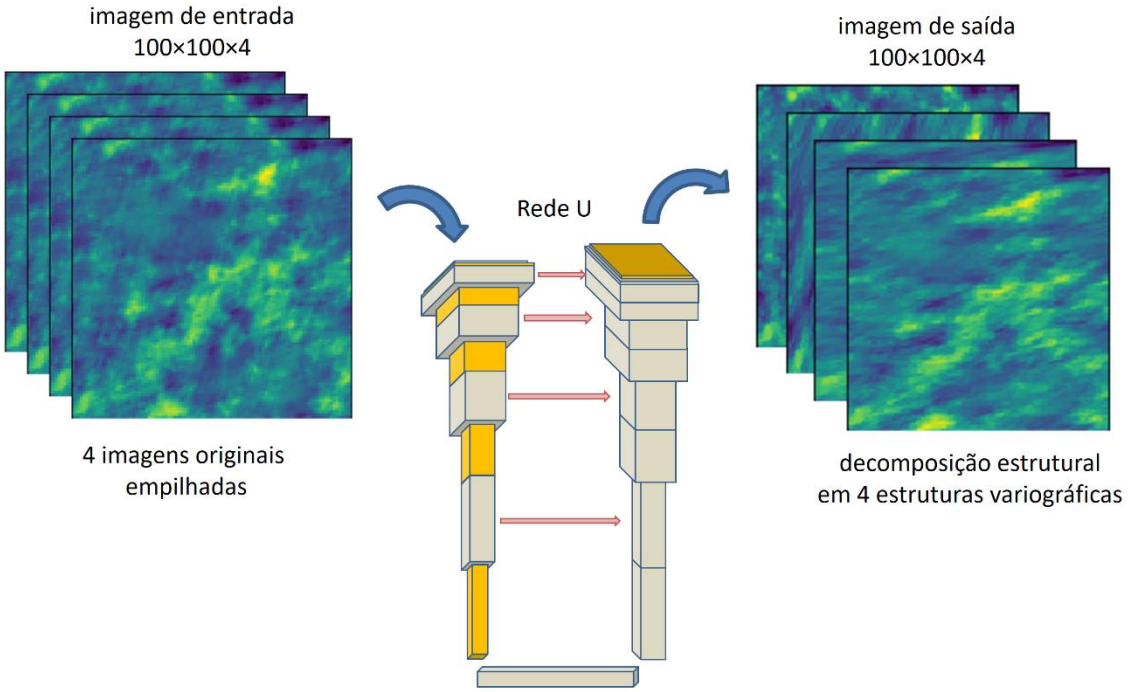


Figura 4-25: Uso de uma rede U para decompor diretamente um mapa em seus elementos estruturais, cada qual correspondendo a uma estrutura variográfica imbricada.

4.6.1 Treinamento com superfícies variográficas

Após treinada em 20 épocas com apenas 40 modelos variográficos aleatórios, o resultado para dez deles está mostrado na Figura 4-26. A razão para o pequeno número de modelos para treinamento se deve ao tempo por iteração muito maior (1 segundo). Um segundo pode não parecer muito, mas para 3.600 imagens significa perto de 1h por época, ou seja, o treinamento levaria em torno de 20h. Observa-se que a rede não conseguiu aprender a decomposição, talvez por causa do treinamento insuficiente. O experimento pode ser realizado com o arquivo (Python Notebook) `Arch4_Varmap_Decompose.ipynb` (ver Seção 1.5 para o caminho completo).

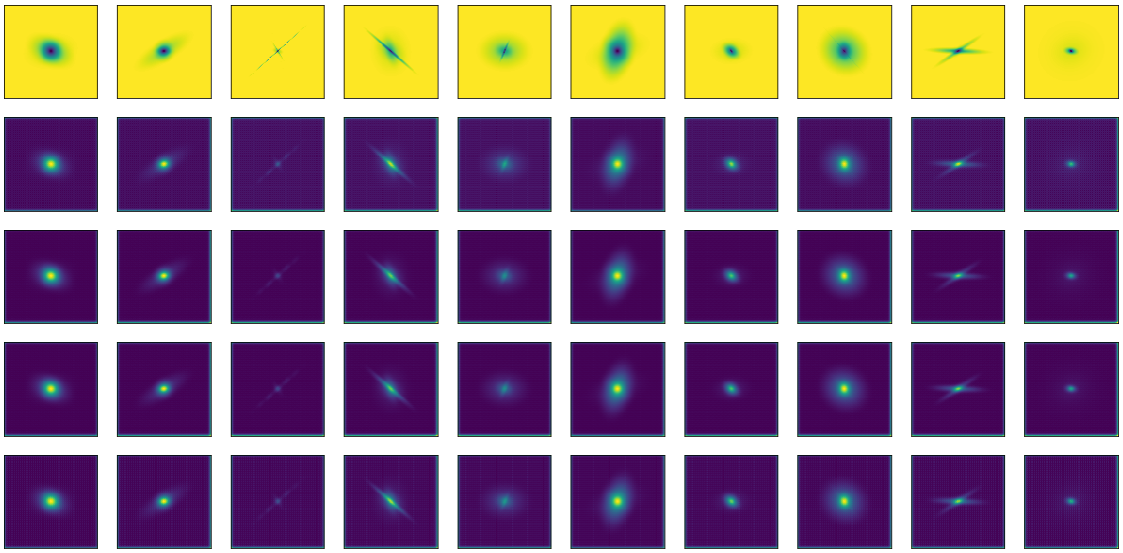


Figura 4-26: Emprego da Arquitetura 4 para decomposição direta. A fileira superior contém dez imagens de entrada (modelos variográficos) e as quatro inferiores, dez tentativas de decompô-los em suas estruturas imbricadas pela rede.

4.6.2 Treinamento com dados sintetizados com FIM

Para se chegar a um resultado conclusivo para o uso da Arquitetura 4 para decomposição direta, foi feito o treinamento em 20 épocas com 5.500 mapas sintetizados com FIM (Seção 2.4) a partir de variogramas aleatórios. Esse treinamento durou aproximadamente 27h e requereu 11GB de memória em um computador com processador Intel Xeon Hexacore (12 núcleos lógicos). A Figura 4-27 mostra o resultado para dez mapas sintéticos em que se observa que as quatro estruturas são apenas cópias da mesma reconstrução da imagem de entrada. Pode-se depreender disso que a rede U aprende a reconstruir a imagem de entrada e não aprende uma relação que transforme uma imagem em outra. O experimento pode ser realizado com o arquivo (Python Notebook) `Arch4_FIM_Decomposition.ipynb` (ver Seção 1.5 para o caminho completo).

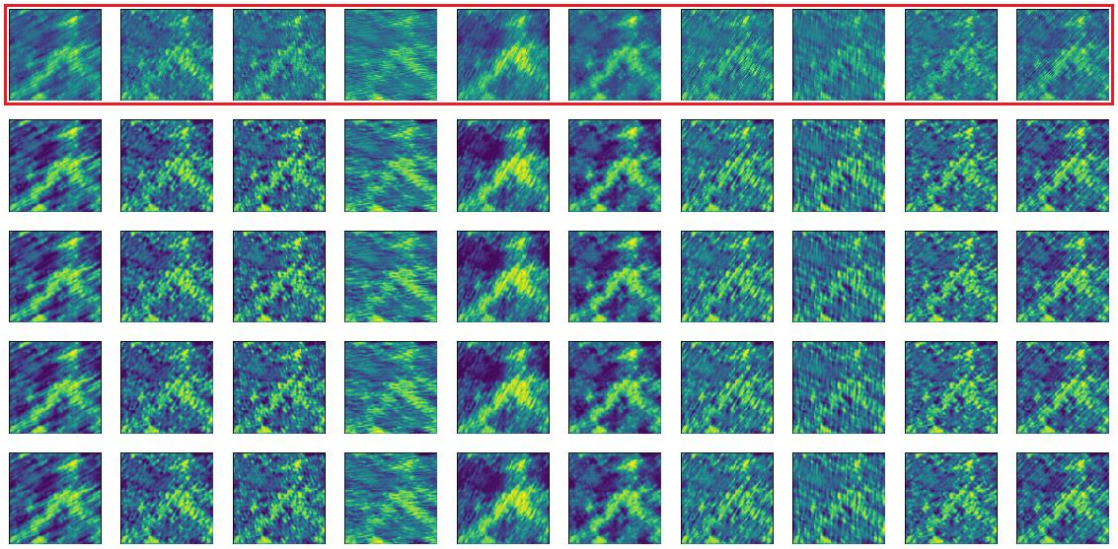


Figura 4-27: Emprego da Arquitetura 4 para decomposição direta. A fileira superior, destacada em vermelho, contém dez imagens de entrada (mapas sintetizados com FIM a partir de variogramas aleatórios) e as quatro inferiores, dez tentativas de decompô-las em componentes segundo cada estrutura variográfica imbricada pela rede.

4.7 Arquitetura 6

Devido à dificuldade em obter uma separação entre as quatro estruturas imbricadas nas Seções anteriores, uma rede autocodificadora com quatro canais em paralelo, uma para cada estrutura imbricada, foi construída. A visão geral da arquitetura está na Figura 4-28: (a) é a parte codificadora a ser treinada posteriormente; (b) são quatro instâncias de uma mesma rede que é treinada anteriormente para aprender a relação entre os quatro parâmetros variográficos (azimute, semieixos e contribuição) e o elipsoide resultante.

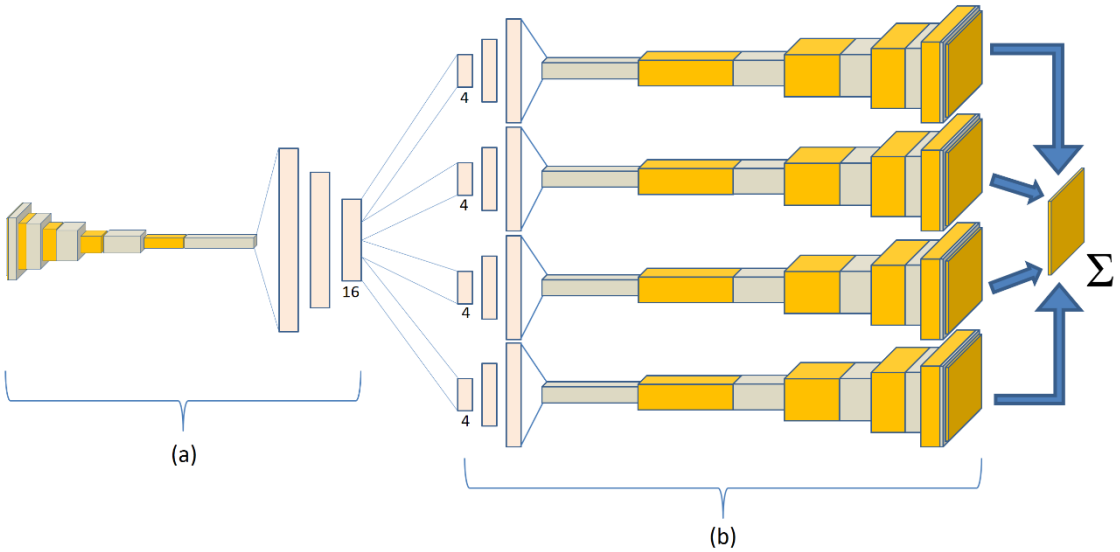


Figura 4-28: Visão geral da Arquitetura 6. (a) e (b): ver texto.

Cada parte (b) da arquitetura da Figura 4-28 tem a arquitetura mostrada na Figura 4-29. Essa sub-rede aceita como entrada os quatro parâmetros variográficos de uma estrutura variográfica e retorna como saída a imagem do elipsoide correspondente aos parâmetros. Essa rede foi então treinada com 5000 pares constituídos, cada qual, pelos quartetos de parâmetros variográficos como entrada gerados aleatoriamente e pelas imagens dos respectivos elipsoides como saída. O resultado do treinamento está na Figura 4-30 em que se constata desempenho insatisfatório. O experimento pode ser realizado com o arquivo (Python Notebook) `Arch6_subnet_Varmap.ipynb` (ver Seção 1.5 para o caminho completo).

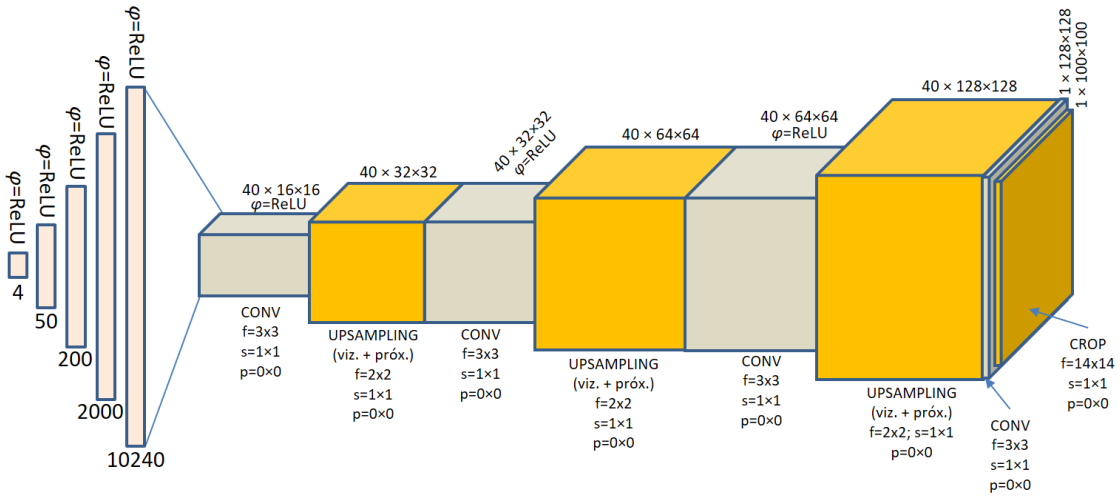


Figura 4-29: Detalhe da arquitetura das partes (b) da arquitetura geral (Figura 4-28).

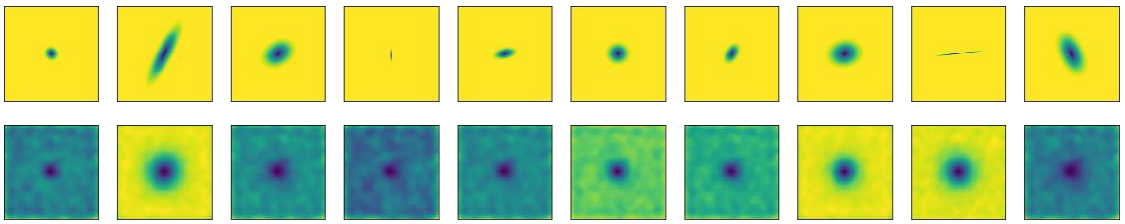


Figura 4-30: Resultado do treinamento da rede da Figura 4-29. A fileira superior são os resultados esperados (estruturas variográficas simples). A fileira inferior são as predições feitas pela rede quando lhe são apresentados azimutes, semieixos e contribuições.

Em que pese o desempenho ruim dessa sub-rede, uma sub-rede autocodificadora foi construída para o treinamento prévio da parte decodificadora, seguindo o raciocínio visto na Seção 4.3. A Figura 4-31 mostra a rede usada para o treinamento prévio. Após treinada com 5.500 estruturas variográficas aleatórias, a sub-rede autocodificadora foi capaz de enxergar bem boa parte delas (Figura 4-32), embora tenha falhado com as estruturas de menor escala.

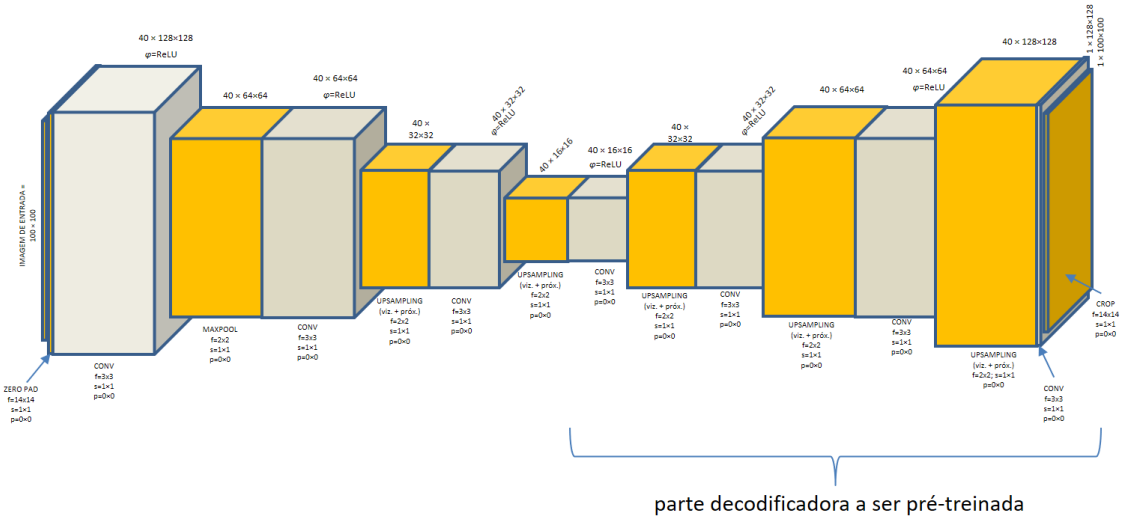


Figura 4-31: Rede autocodificadora usada para pré-treinar a parte decodificadora dos elementos (b) da arquitetura da Figura 4-28.

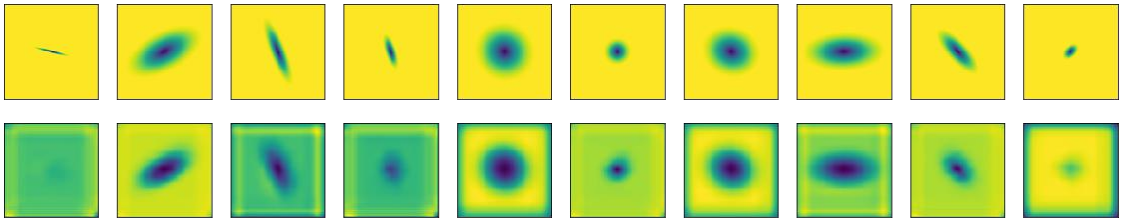


Figura 4-32: Resultado do treinamento da rede mostrada na Figura 4-31. As dez imagens da primeira linha são estruturas variográficas geradas aleatoriamente. As dez imagens na fileira de baixo são as respectivas reconstruções pela rede.

Aumentando a profundidade das camadas para 80, melhora-se um pouco as reconstruções (Figura 4-33). E mais um pouco para 120 (Figura 4-34).

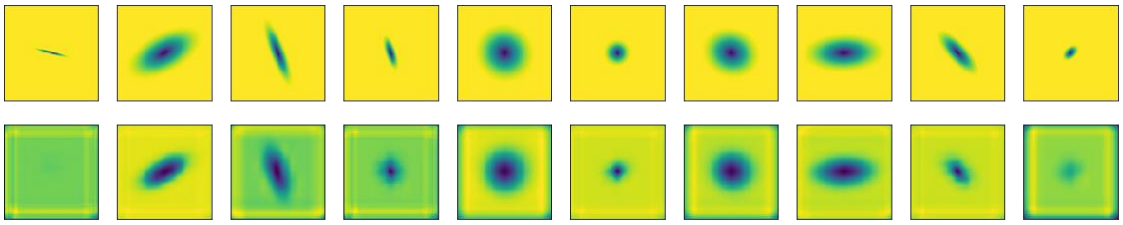


Figura 4-33: Resultado do treinamento da rede, porém com camadas com profundidade aumentada de 40 para 80.

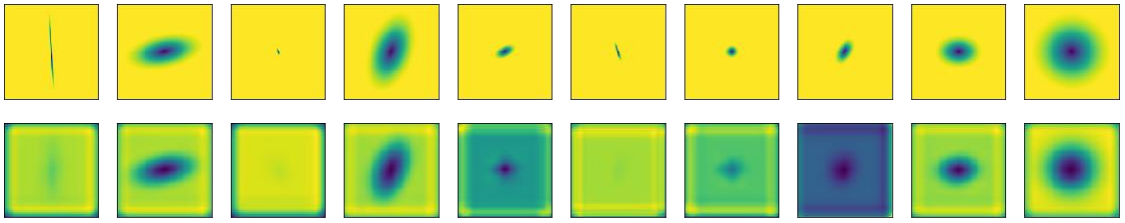


Figura 4-34: Resultado do treinamento da rede, com camadas de profundidade 120.

Deve-se ter em mente que o aumento da profundidade das camadas aumenta consideravelmente o tempo de treinamento. A pequena melhora na qualidade das reconstruções, ainda insuficiente para se considerar satisfatórias as reconstruções, não proporcionou benefício que justificasse o aumento do tempo de treinamento de 1h40, com 40 de profundidade, para 5h com 120 de profundidade por camada convolutiva. As estruturas variográficas de alta anisotropia, justamente as que normalmente correspondem a artefatos, tiveram as piores reconstruções. Aumentando a profundidade da rede como um todo (número de camadas) até o ponto máximo, em que as imagens foram codificadas de 100×100×1 para 1×1×512, piorou as reconstruções (Figura 4-35).

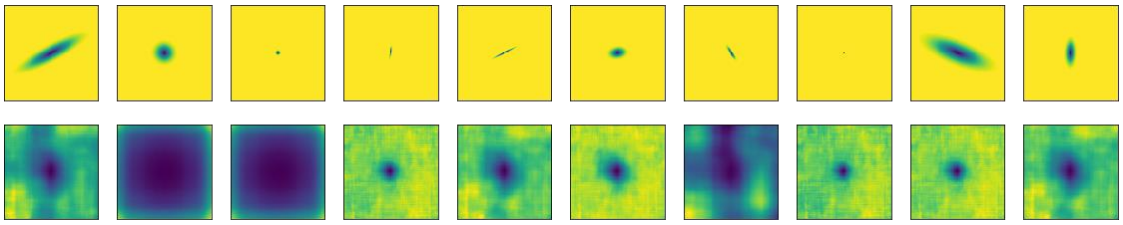


Figura 4-35: Resultado do treinamento da rede, com número de camadas aumentado de 4 para 7 nas partes codificadora e decodificadora.

O experimento pode ser realizado com o arquivo (Python Notebook) `Arch6_subnet_pretrain_Varmap.ipynb` (ver Seção 1.5 para o caminho completo).

4.8 Arquitetura 7 com núcleos de Gabor (GCN)

Sendo o treinamento com imagens sintetizadas nas Seções anteriores uma forma de *data augmentation* (Seção 2.5.7), é possível que o número de imagens de treinamento tenha sido insuficiente (*underfitting*). O treinamento insuficiente pode ser devido ao grande número de parâmetros variográficos (16) relativamente ao pequeno número de parâmetros típico de aplicações populares de CNNs (rotação, escala, cisalhamento e translação). Van Dyk e Meng (2001) argumentam que "construir esquemas de *data augmentation* que resultem em algoritmos tanto simples quanto rápidos é uma questão de arte" com o critério eficácia subentendido.

Mesmo com a redução do número de parâmetros variográficos para quatro investigada na Arquitetura 6, os resultados ainda se mostraram insatisfatórios.

Conforme discutido na Seção 2.5.8, a robustez quanto à escala e à orientação das feições com menos treinamento das redes GCN têm um custo computacional que as inviabiliza utilizando o método da modulação. Pode-se evitar essa modulação de núcleos se a rede contivesse camadas que aprendessem os parâmetros (azimute e escala) diretamente para formar núcleos de Gabor puros. Esses, então, são utilizados na operação de convolução dentro da camada.

4.8.1 Camada Gabor

Na Seção 2.5.4, viu-se que a camada convolutiva aprende os pesos do núcleos de convolução diretamente. Núcleos de Gabor podem emergir espontaneamente durante o processo estocástico de aprendizado, mas tal depende dos dados para treinamento e da configuração do processo de treinamento em si. Esta tese, portanto, propõe um novo tipo de camada convolutiva, referida por `GCONV` nas arquiteturas, capaz de aprender parâmetros de Gabor, de forma que seus núcleos de convolução sejam necessariamente núcleos de Gabor durante todo o processo de aprendizado. A nova camada está implementada como a classe `GaborLayer` no programa `Arch7_Gabor.ipynb` (ver Seção 1.5 para o caminho completo). Essa classe deriva da classe `Layer` do Keras, portanto pode ser usada com os outros tipos de camada vistos até aqui para formar arquiteturas de rede.

A Figura 4-36 mostra uma arquitetura simples, ilustrando um exemplo de saída de uma camada Gabor. A imagem de entrada (esquerda) é submetida a 64 núcleos de Gabor, cada qual com um par de parâmetros: azimute e escala. Desses 64 núcleos resulta um tensor com 64 mapas de feições (*feature maps* no jargão do aprendizado de máquina). Uma camada de somatório (também implementada no código-fonte em `Arch7_Gabor.ipynb`) integra esses 64 mapas de feições para compor a saída da rede. O processo de treinamento, então, pode encontrar um conjunto ótimo de parâmetros de filtros de Gabor de forma que o resultado seja, idealmente, igual do mapa de entrada.

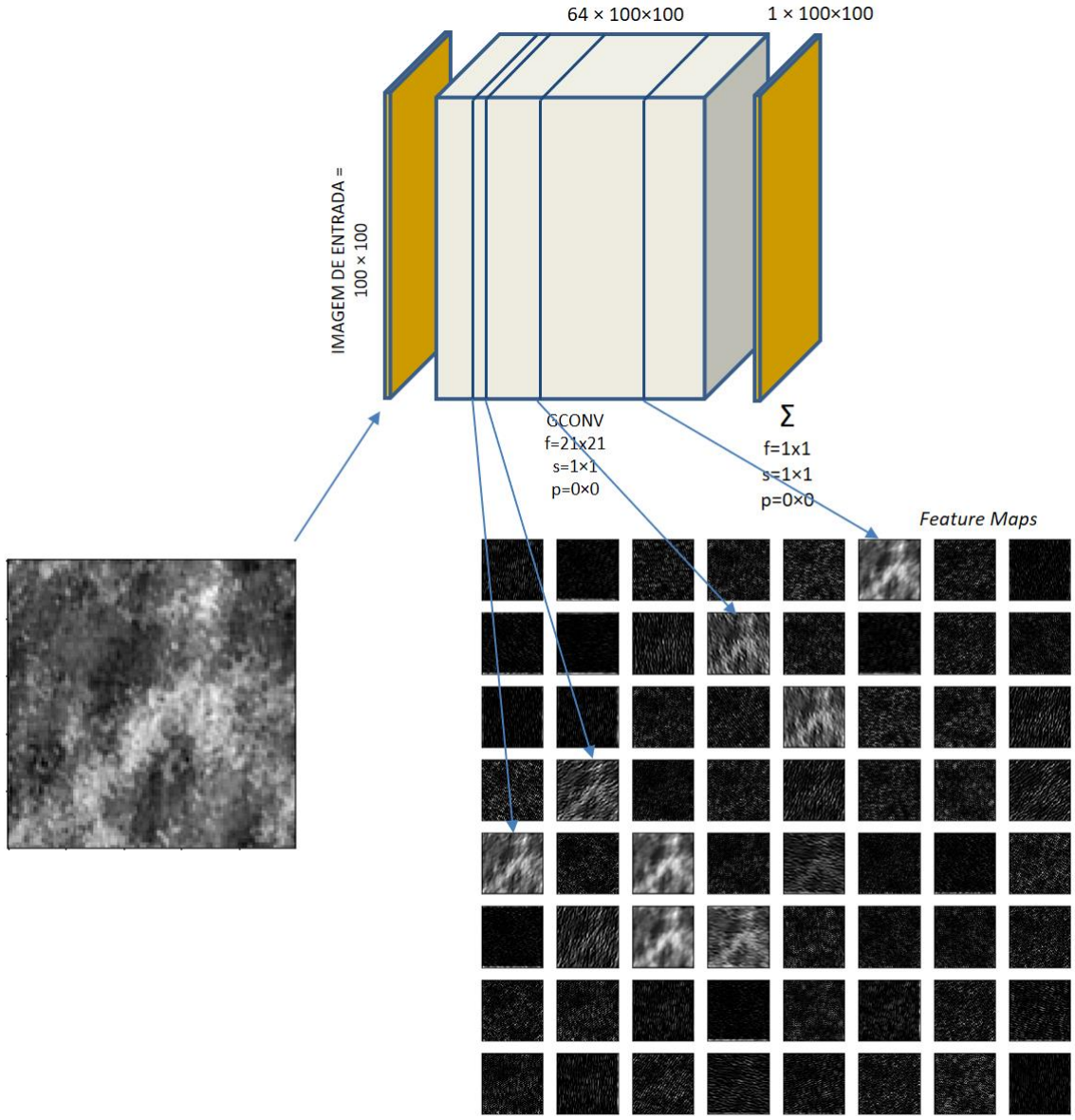


Figura 4-36: Exemplo de saída de uma camada Gabor. Ver texto.

No entanto, surge uma questão técnica: os parâmetros Gabor usados como pesos no processo de aprendizado. Obviamente, valores de azimutes e de escalas não podem ser usados na convolução. Na camada Gabor proposta, os núcleos Gabor são calculados a partir desses parâmetros os quais em seguida são utilizados na convolução. O código Python a seguir é a implementação do método `call()` da camada Gabor, que define o processamento dentro da camada:

```
def call(self, inputs):
    ###########Generate the Gabor kernels from the learned Gabor parameters###########
    gaborKernelStackShape = list( self.kernel_size )
    gaborKernelStackShape.append( self.input_dim ) #self.input_dim is the number of 2D grids in the input tensor
    gaborKernelStackShape.append( self.filters ) #self.filters is the number of 2D grids in the output tensor
    gaborKernels = np.zeros( tuple( gaborKernelStackShape ) )
    for iKernel in range( self.filters ):
        angle = gaborParameters[ iKernel ][0] #get azimuth parameter (orientation of Gabor wavelet)
        lambd = gaborParameters[ iKernel ][1] #get scale parameter (wavelenght of Gabor wavelet)
        #Make a Gabor kernel from the parameters
```

```
        gaborKernel = makeGabor( gridSize = self.kernel_size, \
                                 angle = angle, \
                                 lambd = lambd, \
                                 sigmaX = max(self.kernel_size)/10.0, \
                                 ratioSigmaXY = 1.0, \
                                 phase=0.0, \
                                 func=np.cos ) #cosine = real part of the Gabor kernel
        #Copy the Gabor kernel values to the array of values for the tensor to be
        for iInputTensor in range( self.input_dim ) :
            gaborKernels[ :, :, iInputTensor, iKernel ] = gaborKernel

    #---Create the kernels tensor for the convolution framework
    gaborKernelsTensor = K.variable(value=gaborKernels, dtype=K.dtype( self.kernel ),
name='gaborKernelsTensor')
    ##############################################################################

    #---Perform the 2D convolution between the input tensor and the kernels tensor to get the output tensor-----
    outputs = K.conv2d(
            inputs,
            gaborKernelsTensor, #<------ the Gabor kernels generated from the learned parameters.
            strides=self.strides,
            padding=self.padding,
            data_format=self.data_format,
            dilation_rate=self.dilation_rate)

    zerosShape = list( inputs.shape )
    zerosShape[0] = 1
    zerosShape[3] = 1
    zeros = np.zeros( zerosShape )
    zerosTensor = K.variable(value=zeros, dtype=K.dtype( self.kernel ),
name='zerosTensor')
    outputs += K.conv2d(
            zerosTensor,
            self.kernel, #<------ using the learned parameters to avoid the "'None' for gradients" error.
            strides=self.strides,
            padding=self.padding,
            data_format=self.data_format)

    if self.activation is not None:
        return self.activation(outputs)
    return outputs
```

Esse código matematicamente corresponde a S = E*G + 0*W, onde *S* é um mapa de saída; *E* é o de entrada; *G* é um núcleo Gabor resultante de um par de parâmetros no tensor *W*; *0* é um tensor com todos os elementos iguais a zero e * é o operador convolução. Algebricamente, o resultado é apenas S = E*G, assim cabe uma explicação dos detalhes de como a implementação funciona no arcabouço (*framework*) Keras utilizado nesta tese para a experimentação com aprendizado de máquina.

Sendo Python uma linguagem interpretada, seria computacionalmente ineficiente interpretar o *script* em `call()` para cada época durante o treinamento. O que acontece na prática é que, uma vez definidas as camadas de uma rede, chama-se o método `compile()` do objeto que implementa a rede. A compilação produz um código binário que pode ser executado nativamente na CPU, GPU ou TPU com alto desempenho. Entretanto, para que a compilação seja bem-sucedida, quaisquer variáveis que tenham sido definidas como pesos para o processo de aprendizado devem participar do cálculo realizado dentro da função `call()`, mesmo que não influenciem o resultado numérico. Assim, a solução de contorno

para isso é convolver o tensor dos pesos (os parâmetros Gabor) contra um tensor nulo e somar esse resultado ao resultado final. Disso decorre que o tensor de saída passa a ser invariante com relação ao tensor de pesos, portanto, o gradiente calculado no processo de aprendizado (ver *back-propagation* na Seção 2.5.3) é zero. Portanto, não há como a rede aprender os pesos de Gabor ótimos sem uma reformulação teórica do processo de aprendizado, o que está além do escopo deste trabalho.

4.8.2 Camada convolutiva modulada por Gabor

Não obstante os problemas dessa abordagem discutidos na discutido na Seção 2.5.8, foi tentado um aprendizado com camada Gabor por modulação tal como proposto por Luan *et al.* (2018), porém os resultados não foram satisfatórios. Além disso, pesa ainda o desempenho computacional ruim já discutido e confirmado na experimentação quando comparado a uma camada convolucional convencional. A camada convolutiva modulada por filtros Gabor está implementada no arquivo `Arch7_Gabor_Modulated.ipynb` (ver Seção 1.5 para materiais e métodos).

4.9 Discussão

Convém chamar atenção para os requisitos de memória e de tempo de processamento para emprego de aprendizado de máquina, mesmo para um mapa pequeno (100×100 células). Os métodos de otimização projetados para o fim de decomposição estrutural vistos na Seção 2.1 fazem o mesmo trabalho em tempo muito menor (alguns minutos) e requerendo bem menos memória para o mesmo mapa. Pode-se argumentar que é possível salvar os tensores de pesos como arquivos para reutilização posterior a fim de que o treinamento da rede só precise ser feito uma vez. De fato, o processo de predição executa muito rápido, e que a vantagem do aprendizado de máquina sobre os demais métodos torna-se evidente no longo prazo. Entretanto, cada mapa tem uma composição estrutural diferente, assim, é necessário um aprendizado diferente para cada mapa.

Parece ser importante que os dados passados para a rede sejam reescalados de forma que os valores variem entre 0,0 e 1,0. Ao menos isso vale para a implementação usada nesta tese (pacote Keras no Python usando TensorFlow como *backend*).

Uma rede neuronal pode, em princípio, aprender qualquer transformação. Entretanto, o custo computacional para aprendê-la é normalmente menos eficiente do que utilizar um algoritmo dedicado a determinado problema bem projetado. Contudo, convém observar o seguinte: uma explicação para a ineficácia observada pode estar no fato de que 160.000

imagens usada com a Arquitetura 1, por exemplo, não sejam suficientes para açambarcar um espaço de parâmetros com 16 parâmetros contínuos. Considerando-se apenas três valores para cada parâmetro, trata-se $3^{16} \approx 43$ milhões de combinações. Um número de imagens dessa grandeza seria necessário para um aprendizado suficiente, o que demandaria uma arquitetura de hardware com maior capacidade como, por exemplo, empregando clusters de GPU ou TPU. Entretanto, uma quantidade insuficiente de treinamento não explica por que a Arquitetura 6 falhou, uma vez que as entradas de seus elementos têm apenas 4 parâmetros. Essa arquitetura foi treinada com 5.000 imagens, o que corresponde a aproximadamente 9 valores para cada parâmetro: $9^4 = 6.561$.

Não foram investigadas as chamadas "redes famosas" que são arquiteturas mais complexas do que as expostas aqui. Trata-se de um grande número de arquiteturas concebidas pela comunidade de *Deep Learning* para resolver determinados tipos recorrentes de problemas com imagens e são conhecidas por diversos nomes: VGG, ResNet, Inception, Xception, etc. com destaque para as arquiteturas conhecidas coletivamente por redes pix2pix.

Capítulo 5

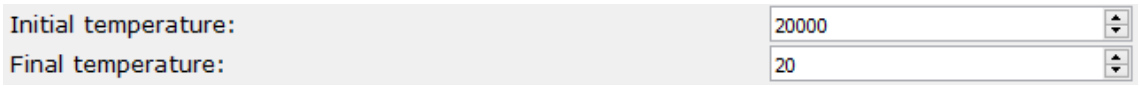
Decomposição automatizada com algoritmo dedicado

Um modelo popular de estrutura espacial empregado na modelagem geológica é o variograma, frequentemente utilizado em geoestatística para elaboração de estimativas e simulações. Em certos casos, o fenômeno modelado pode apresentar uma estrutura espacial complexa, por exemplo, com múltiplas anisotropias e múltiplas escalas. Também, há situações onde uma imagem que cobre uma grande extensão geográfica (por exemplo, nos chamados estudos regionais na fase de exploração em projetos de explotação de petróleo) provavelmente apresenta grande diversidade de estruturas (variograma local variável – Xu, 1996). Portanto, um único modelo de estrutura espacial é inadequado para representá-las. Esses são exemplos de aplicações para automatização da decomposição de uma imagem em fatores geomorfológicos que esta tese propõe. Um algoritmo projetado especificamente para este fim é apresentado e serve de contraponto para o processo de decomposição aprendido por algoritmo de aprendizado de máquina visto no Capítulo 4.

5.1 Parametrização dos métodos de otimização

Nesta seção, são explanados os parâmetros dos métodos de otimização implementados no software para execução do método de decomposição estrutural. Os métodos de otimização (Seção 2.1) são utilizados para minimizar a função-objetivo (Seção 5.3) e, com isso, encontrar os parâmetros variográficos de cada estrutura imbricada para obter os fatores geomorfológicos. Cada método de otimização pode ser acessado acionando a aba correspondente na interface (ver Figura 5-16). Depois de configurado o método de otimização, basta acionar o botão com ícone de "play", existente em cada página, para iniciar o processamento.

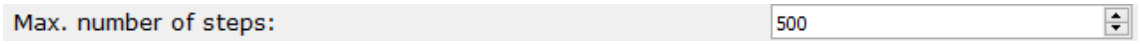
5.1.1 Parametrização para SA+GD



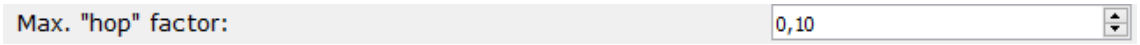
Define as temperaturas inicial e final do "processo de têmpera", a chamada *annealing schedule*. Na implementação do software GammaRay, a temperatura inicial controla o quão rápido a temperatura decai a cada passo, de modo que uma temperatura inicial de 20.000 faz com que a temperatura caia mais rapidamente do que um processo que comece com 2.000. A curva de decaimento é exponencial, assim a temperatura final controla o número de passos, de forma que uma temperatura final mais baixa resulta em um número maior de iterações. A fórmula empírica do decaimento está na Equação 5-1:

$$t_i = t_0 \cdot e^{-\frac{i}{1000} \cdot 1{,}5 \cdot \log_{10} t_0}$$

Equação 5-1: Fórmula empírica para o decaimento da temperatura durante a otimização com SA. *i* é o número do passo; t_0 é a temperatura inicial; t_i é a temperatura do passo *i*.



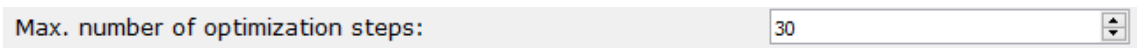
Define o número máximo de passos, lembrando que a temperatura final pode fazer SA terminar antes.



Define o tamanho máximo do "salto" em fração do tamanho do domínio. Durante o caminho aleatório do SA, é sorteado um tamanho de passo, cujo valor máximo é dado por esse valor. Um valor baixo significa uma procura mais cuidadosa, porém com maior risco de não localizar o mínimo global antes que a temperatura final seja atingida ou o número máximo de passos seja alcançado. Um valor alto significa que mais "terreno" é coberto, porém com maior risco de passar direto pelo mínimo global. Um valor entre 10% e 30% do tamanho do domínio é um bom número.

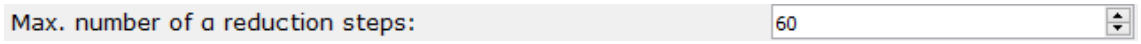
Assim que o SA termina, o algoritmo de GD (com base na Equação 2-1) continua a busca pelo mínimo da função-objetivo usando os parâmetros livres encontrados pelo SA.

Define o tamanho do passo da derivação calculada numericamente. Esse é o ε da Equação 2-3. Quanto menor o valor, maior a precisão do cálculo, porém mais demorado é para alcançar a convergência.

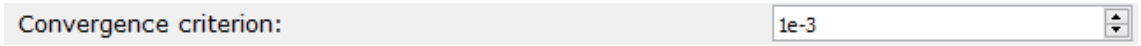


Define o número máximo de passos de GD após a execução do SA. Normalmente, o número 30 sugerido é mais do que suficiente.

Define o fator inicial de redução do gradiente no GD. É o número α da Equação 2-1. A cada passo de otimização, o programa encontra um valor adequado de α automaticamente, mas para isso precisa de um valor inicial. Um valor inicial baixo (ex.: 0,5) faz cada passo demorar menos tempo, porém pode fazer a convergência como um todo demorar mais.



Define o número máximo de passos de redução do valor α do GD. A cada passo de redução, α é reduzido à metade (1,00; 0,5; 0,25; ...). Normalmente, o sistema encontra um α adequado entre 2 e 5 passos, mas em circunstâncias especiais pode ser que isso não ocorra, nem com α muito pequeno. Assim, é necessário definir uma condição de parada para que o processamento não fique em *loop* infinito.



Define o parâmetro mais importante do GD, que é o valor ε do critério de convergência dado pela Equação 5-2. O processo de otimização ideal termina quando esse critério é satisfeito e não quando um número máximo de passos é atingido.

$$F([p]_{i-1}) \,/\, F([p]_i) \;<\; 1 \;+\; \varepsilon$$

Equação 5-2: Critério de convergência utilizado no GD implementado. *i* é o número da iteração. *F*: função-objetivo; [p]: vetor de parâmetros variográficos.

5.1.2 Parametrização para LSRS

LSRS é, dos métodos investigados, o de parametrização mais simples.

Define o tamanho do passo de derivação para calcular numericamente o gradiente da função-objetivo (ver Equação 2-3).

Define o número máximo de passos de otimização.

Define o número de pontos iniciais (e o número de linhas) gerados aleatoriamente dentro do domínio dos parâmetros variográficos.

Define o número de reinícios (reduções de domínio no entorno de um valor sub-ótimo encontrado).

5.1.3 Parametrização para PSO

A parametrização do PSO contém análogos da mecânica do movimento de partículas, o que requer entendimento de Física básica para compreender seu efeito na dinâmica do processamento.

Define o número máximo de iterações.

Define o número de partículas (conjuntos de parâmetros variográficos) que se moverão pelo domínio durante o processamento.

Define a "massa" das partículas. Como um análogo mecânico, as partículas devem ter uma propriedade equivalente à massa, caso contrário elas não se moveriam como uma partícula, expressa como uma medida de inércia. Quanto maior esse valor, menor a variação na velocidade das partículas, o que significa que uma partícula pode não ser capturada por um mínimo se a função-objetivo for muito errática.

A constante 1 define a aceleração (variação da velocidade) das partículas causada por sua melhor avaliação da função-objetivo. A constante 2 define a aceleração causada pela melhor avaliação global. Ver Equação 2-4.

5.1.4 Parametrização para GA

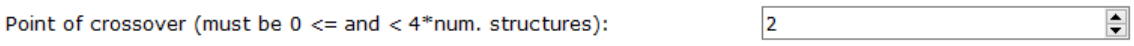
A parametrização de GA requer familiaridade com os princípios da evolução de Darwin assim como de conceitos fundamentais de genética para entender o efeito dos parâmetros no processo de busca de uma solução ótima.

Define o número de gerações (número de iterações).

Define o tamanho da população (número de vetores de parâmetros variográficos).

Define o número de indivíduos que sobrevivem de uma geração para outra. Este número deve ser menor que o da população, evidentemente.

Define a probabilidade de cruzamento (geração de novas soluções a partir de duas soluções existentes). Quanto maior o número, maior a quantidade de cruzamentos.



Define o ponto de cruzamento que é o índice do parâmetro a partir do qual a troca de valores ocorre na geração de descendentes. Por exemplo, se o usuário quiser decompor a imagem original em 3 fatores geomorfológicos, o número total de parâmetros por indivíduo é 12 (4 parâmetros variográficos por estrutura imbricada). Se o usuário definir esse parâmetro como 5, o "filho 1" herdará os parâmetros de 1 a 6 de "pai 1" e o restante será de "pai 2". O mesmo raciocínio se aplica a "filho 2". Se o usuário definir esse valor como zero ou máximo, os descendentes serão meras cópias dos pais (clonagem).

Define a taxa de mutação, que é a componente estocástica do processo. Um valor zero pode resultar em parada em um mínimo local, enquanto um valor alto resulta em uma pesquisa mais errática, explorando mais do domínio ao custo de uma convergência mais lenta.

5.2 Definição de fator geomorfológico

Tomando como referência a decomposição da imagem em fatores obtida com FK (Seção 3.1), define-se como *fator geomorfológico* a componente do mapa de entrada que corresponde a *uma estrutura imbricada* do variograma teórico. Portanto, os critérios adotados nesta tese para a definição de um fator geomorfológico são:

i. O variograma de um fator geomorfológico não tem estruturas imbricadas.

ii. Os alcances e a orientação de um fator geomorfológico devem ser diferentes dos demais. Isto serve para que não haja fatores geomorfológicos que exibam a mesma morfologia, diferindo apenas na escala de valores (contribuição).

5.3 Função-objetivo baseada no VARFIT

Uma vez estabelecidos os critérios para definição de fator geomorfológico, faz-se necessário construir uma função-objetivo *F* que é uma tentativa de expressar quantitativamente esses critérios. Uma função-objetivo *F* é uma função que retorna um valor escalar que pode ser usado nos passos iterativos dos métodos de otimização (ver Seção 2.1). O valor retornado por *F* é uma métrica que quantifica o quão distante um sistema está do ideal, objetivando a satisfação dos critérios apresentados.

A função-objetivo em questão é a extensão para 2D da função-objetivo proposta por Larrondo *et al.* (2003) para ajuste de modelos variográficos 1D na conhecida rotina `varfit` da GSLib (Deutsch e Journel, 1998) e dada pela Equação 5-3. n_{var} é o número de variáveis; n_{dir} é o número de direções (azimutes); n_{lag} é o número de *lags* (separações – valores de *h* no variograma); γ é o valor do variograma teórico para determinada separação *h*; *V* é o valor do variograma experimental para determinada separação *h*; *k* é o índice da variável *head*; *l* é o índice da variável *tail*.

$$F = \sum_{k=1}^{n_{var}} \sum_{l=k}^{n_{var}} \sum_{i=1}^{n_{dir}} \sum_{j=1}^{n_{lag}} \lambda_{j_{lag}}^{i_{dir}} \left[\gamma_{k,l}\left(h_{j_{lag}}^{i_{dir}}\right) - V_{k,l}\left(h_{j_{lag}}^{i_{dir}}\right) \right]^2$$

Equação 5-3: Função-objetivo (Larrondo *et al.* 2003) para o programa **`varfit`** da GSLib. Ver texto.

λ é um peso dado de forma a melhorar o ajuste em determinadas circunstâncias. Larrondo *et al.* (2003) propõem quatro ponderações: constante; inverso da distância; número de pares e preferência do usuário. Ainda segundo esses autores, a ponderação pelo inverso da distância é indicada para *lags* curtos. Como o mapa variográfico é calculado na resolução do mapa de entrada (ver Seção 2.3), então pode-se considerar o *lag* como curto (em verdade, é o *lag* mínimo para uma malha regular). Portanto, adotou-se a ponderação pelo inverso da distância (Equação 5-4).

$$\lambda_{j_{lag}}^{i_{dir}} = \frac{1/d(u_{idir,jlag})}{\sum_{j=1}^{n} 1/d(u_{idir,j})}$$

Equação 5-4: Cálculo do peso λ da Equação 5-3 pelo inverso da distância (Larrondo *et al.*, 2003) para uma determinada direção i_{dir} (1D). *u*: local no espaço; *d*: distância de um ponto *u* até o ponto experimental onde se está calculando as semivariâncias em todas as direções e *lags* e o qual se deseja ponderar.

O denominador da Equação 5-4 pode ser visto como um normalizador devido ao número variável de pontos em cada direção (azimute) quando se calcula o variograma experimental

para dados irregulares. Porém, quando o mapa variográfico é computado para um conjunto de dados regulares, o número de pontos experimentais é maior conforme aumenta a separação (*h*), proporcionalmente ao perímetro de um círculo de raio *h*. A Figura 5-1 ilustra a questão.

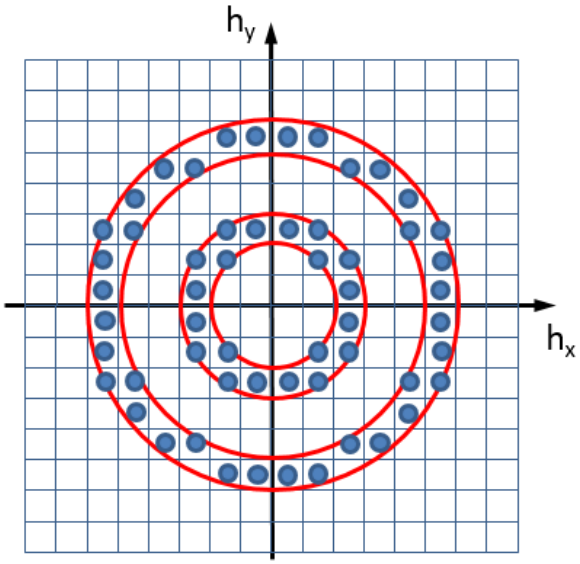


Figura 5-1: Dois *lags* de um mapa variográfico com número de pontos proporcional ao perímetro de um círculo de raio equivalente ao *lag*.

Sendo o perímetro do círculo dado por 2πr, onde *r* é o raio, o número estimado de pontos ao longo desse perímetro é 2πr/l, onde *l* é o espaçamento entre os pontos (tamanho das células). Assim, para uma malha regular 2D, o denominador normalizador da Equação 5-4 torna-se conforme a Equação 5-5.

$$\lambda_{lin,col} = \frac{{}^{1}/_{d(u_{lin,col})}}{{}^{2\pi d(u_{lin,col})}/_{l}}$$

Equação 5-5: Cálculo do peso λ pelo inverso da distância para uma determinada célula de um mapa variográfico calculado para um dado de entrada regular. *u*: local no espaço de uma célula do mapa dada sua linha e coluna; *d*: distância de uma célula *u* até o centro do mapa variográfico; *l*: espaçamento considerado entre os pontos.

Como o problema só tem uma variável (n_{var} = 1), a Equação 5-3 simplifica-se na Equação 5-6.

$$F = \sum_{i=1}^{n_{dir}} \sum_{j=1}^{n_{lag}} \lambda_{j_{lag}}^{i_{dir}} \left[\gamma \left(h_{j_{lag}}^{i_{dir}} \right) - V \left(h_{j_{lag}}^{i_{dir}} \right) \right]^2$$

Equação 5-6: versão univariada da Equação 5-3.

Para um mapa variográfico, que é uma malha regular (*grid*), a Equação 5-6 torna-se a Equação 5-7. λ é a ponderação dada pela Equação 5-5; γ é a superfície variográfica teórica avaliada na malha regular (ver Seção 5.4.1); V é o mapa variográfico obtido pelo método apresentado na Seção 2.3.

$$F = \sum_{j=1}^{J} \sum_{i=1}^{I} \lambda_{i,j} \left[\gamma_{i,j} - V_{i,j} \right]^2$$

Equação 5-7: Equação 5-6 para malhas regulares 2D. *I* e *J* são as dimensões da malha.

A superfície variográfica teórica γ é computada pela avaliação de um variograma teórico a partir de seus parâmetros (azimute, alcances, etc.) em cada célula na mesma malha do mapa variográfico. Detalhes completos de como gerar superfícies variográficas teóricas em software estão apresentados em Carvalho *et al.* (2018). Finalmente, a função-objetivo expressa em função dos parâmetros variográficos é a Equação 5-8, que é a função-objetivo implementada no software utilizado nos experimentos.

$$F(p_{11}, \dots, p_{mn}) = \sum_{j=1}^{J} \sum_{i=1}^{I} \lambda_{i,j} \left[\gamma(p_1, \dots, p_{mn})_{i,j} - V_{i,j} \right]^2$$

Equação 5-8: Função-objetivo final *F*. *I* e *J* são as dimensões da malha. *i* e *j* são os índices (linha e coluna) das células da malha. V é o mapa variográfico experimental e γ é o mapa variográfico teórico. p são os parâmetros. m e n: ver texto.

Os parâmetros p_k da função-objetivo $F(p_1, ..., p_{mn})$ são, portanto, os parâmetros do modelo variográfico, onde m é o número de estruturas imbricadas e n é o número de parâmetros por estrutura ($n = 4$). A relação entre os parâmetros variográficos e os parâmetros arranjados como um vetor $p_1, ..., p_{mn}$ para a função-objetivo está ilustrada na Figura 5-2. Os parâmetros contínuos de um modelo bidimensional são semieixo maior (*a*), razão semieixo menor/semieixo maior (b/a), azimute (θ) e contribuição (cc). O tipo da estrutura (exponencial, esférico, Gaussiano, etc.) não foi incluído como um dos parâmetros e foi fixado em "esférico" por sua versatilidade. A razão disso é que um parâmetro deste tipo (não-ordinário e descontínuo) não pode ser usado como uma coordenada Cartesiana para o

cálculo de gradientes da função-objetivo para os métodos de otimização GD+SA (Seção 2.1.1) e LSRS (Seção 2.1.2).

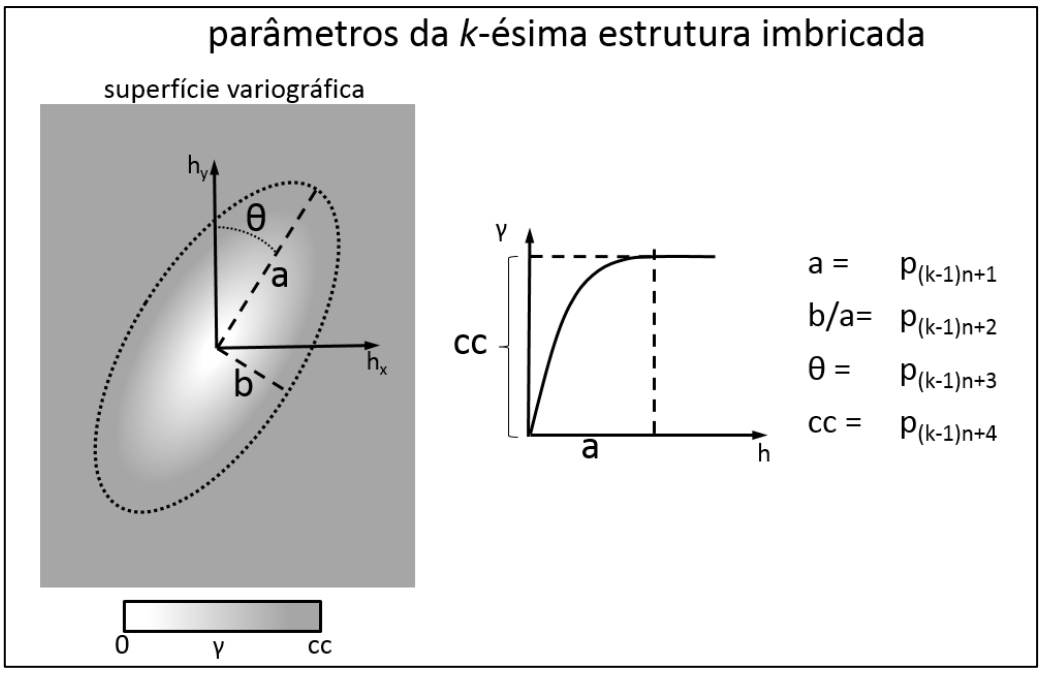


Figura 5-2: Os parâmetros contínuos de uma estrutura imbricada de um modelo variográfico. *a*: , *b*, *θ* e *cc*: ver texto; *h*: valor da separação bi-ponto no espaço variográfico; *γ*: valor da semivariância teórica; p_1,..., p_{mn}: parâmetros arranjados como vetor. *m* e *n*: ver texto.

Sendo *m* o número de fatores geomorfológicos, então *m* é, de acordo com os critérios, o número de estruturas imbricadas no modelo variográfico a ser ajustado. Então, por exemplo, se o modelador desejar decompor o mapa em 3 estruturas imbricadas (*m*=3 e *n*=4), a função-objetivo será $F(p_1, p_2, \ldots, p_{11}, p_{12})$.

5.4 Fluxo de ajuste automático de variograma baseado no VARFIT

Naturalmente, uma parte importante de um processo de decomposição estrutural automatizada passa por um passo de ajuste automático de modelo variográfico. Inicialmente, aplica-se FFT ao mapa de entrada *M*, de onde se retém o mapa de amplitudes (Figura 5-3).

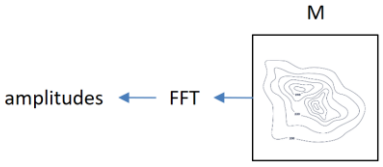


Figura 5-3: Fluxo automatizado de decomposição baseado no VARFIT: 1º passo.

Depois, eleva-se o mapa de amplitudes ao quadrado para se obter o mapa da densidade espectral. Aplica-se RFFT com mapa de fases zerado ao mapa da densidade espectral para se obter o mapa variográfico experimental (Figura 5-4).

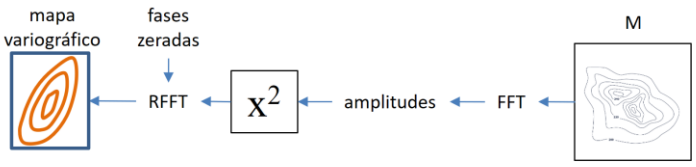


Figura 5-4: Fluxo automatizado de decomposição baseado no VARFIT: 2º passo.

Em seguida, inicializa-se os parâmetros variográficos e, a partir deles, obtém-se o mapa com a superfície variográfica teórica (Figura 5-5).

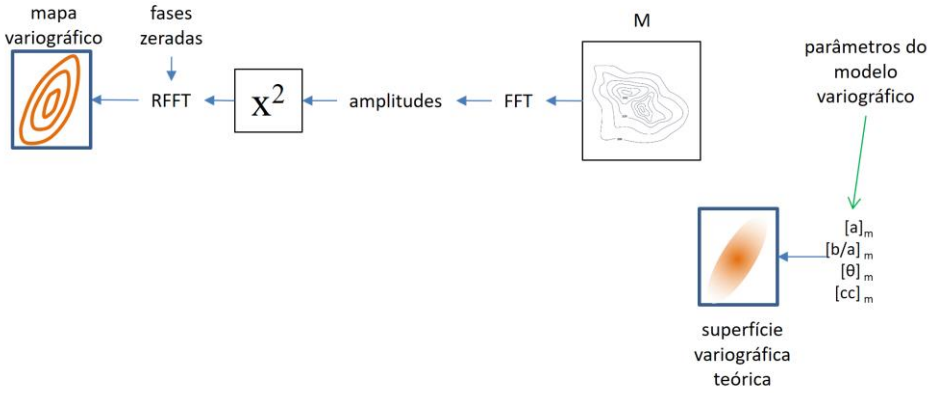


Figura 5-5: Fluxo automatizado de decomposição baseado no VARFIT: 3º passo.

Finalmente, o mapa variográfico experimental e a superfície variográfica teórica são submetidos à função-objetivo baseada no VARFIT (Equação 5-8). Os parâmetros variográficos são então atualizados por um método de otimização em função do valor-objetivo retornado para um novo ciclo de otimização (Figura 5-6). O ciclo se repete até a satisfação de um critério de convergência.

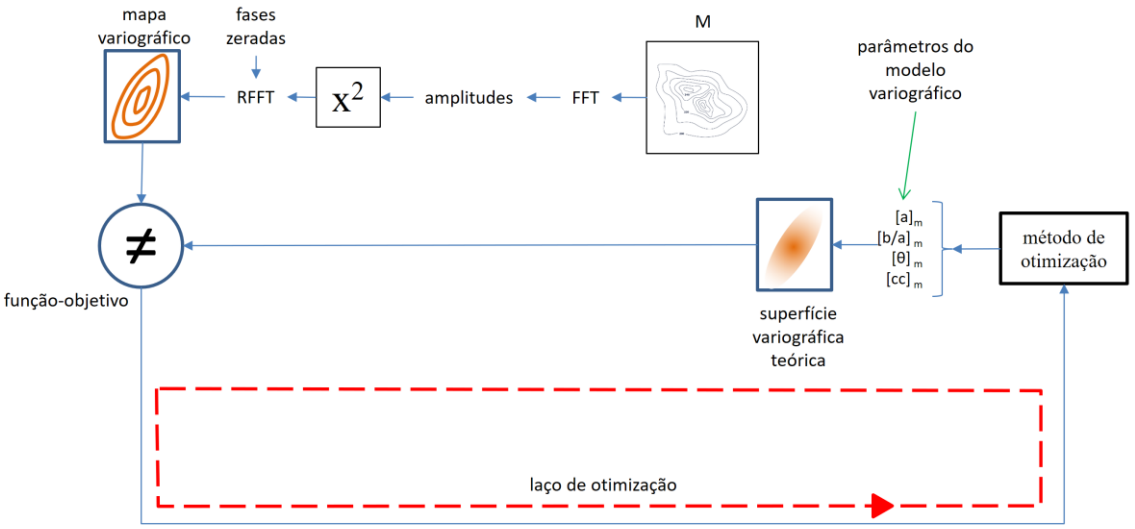


Figura 5-6: Fluxo automatizado de decomposição baseado no VARFIT: 4º passo.

Trata-se de um fluxo relativamente simples e de bom desempenho computacional.

Como todos os métodos de otimização implementados são baseados em caminhos aleatórios, deve-se executar esse fluxo um certo número de vezes variando a semente para o gerador de números aleatórios. Fazendo-se assim, espera-se que os parâmetros variográficos automaticamente ajustados formem agrupamentos bem definidos no espaço formado pelos quatro parâmetros variográficos apresentados na Figura 5-2. Esses agrupamentos podem ser avaliados preliminarmente plotando-se *cross plots* para cada par de parâmetros. Se os agrupamentos forem julgados de boa qualidade, um algoritmo de análise de agrupamento (ver Seção 2.6) pode ser usado para obter os parâmetros variográficos do modelo ajustado automaticamente.

5.4.1 Resultados preliminares

O fluxo de ajuste de variograma foi executado um certo número de vezes. Variou-se o número do gerador de números aleatórios de 131313 a 171717 em 50 passos, resultando em um total de 204 estruturas imbricadas: 4 estruturas por modelo * (1 semente inical + 50 sementes) . A sessão foi repetida quatro vezes, uma para cada método de otimização: SA+GD, LSRS, PSO e GA. Cada par de parâmetros geométricos (a, b/a e θ) foi plotado em *cross plots* para avaliação da qualidade dos agrupamentos produzidos com cada método de otimização. Para a contribuição (cc), que se espera ser de mesmo valor em todas as estruturas imbricadas, histogramas foram plotados. A Tabela 5-1 contém a parametrização tal como configurada no software usado nos experimentos. Os textos nessa tabela se referem aos nomes dos parâmetros tais como apresentados nas telas do programa para evitar confusão.

Optimization method: SA+GD	
Simulated Annealing control	
Initial temperature:	20000
Final temperature:	20
Max. number of steps:	1000
Max. "hop" factor:	0.100
Gradient Descent control	
ε:	10^{-6}
Max. number of optimization steps:	30
Initial α:	1.00
Max. number of α reduction steps:	60
Convergence criterion:	10^{-3}
Optimization method: LSRS	
ε:	10^{-6}
Max. number of optimization steps:	10
Number of starting points:	30
Number of restarts:	10
Optimization method: PSO	
Max. number of optimization steps:	50
Number of particles:	50
Inertia weight:	0.15
Acceleration constant 1:	2.90
Acceleration constant 2:	3.70
Optimization method: GA	
Number of generations:	100
Population size:	30
Natural selection size:	20
Probability of crossover:	1.00
Point of crossover:	8
Mutation rate:	0.45

Tabela 5-1: Parametrização dos métodos de otimização do software usado nos experimentos.

As tabelas a seguir (Tabela 5-2; Tabela 5-3; Tabela 5-4; Tabela 5-5) contém os *cross plots* dos pares de parâmetros geométricos e os histogramas das contribuições. A Tabela 5-2 mostra praticamente nenhum agrupamento bem definido.

SA+GD	
a × b/a	θ × b/a

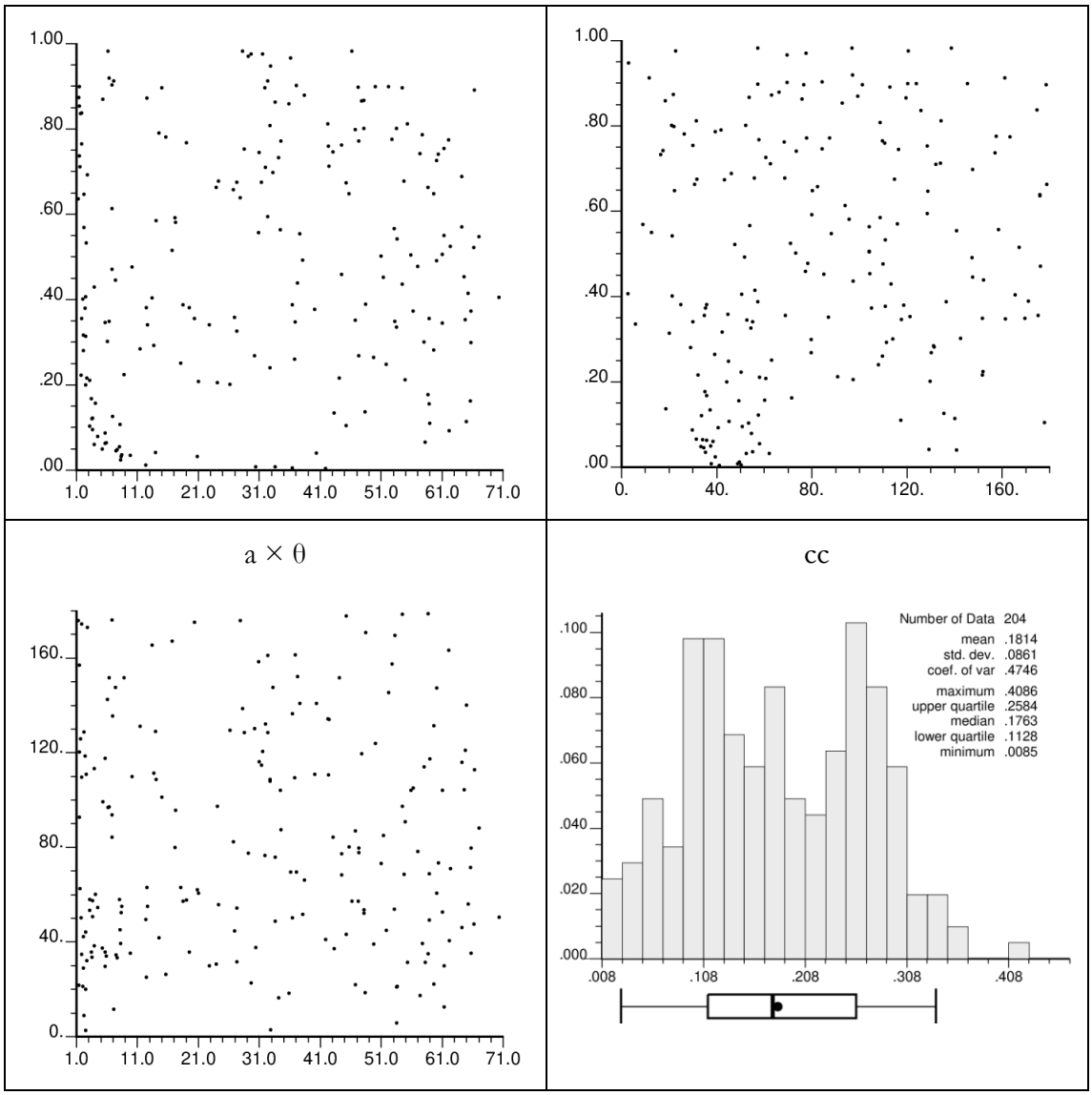


Tabela 5-2: Parâmetros do conjunto de estruturas variográficas imbricadas obtidas com o método de otimização SA+GD com a função-objetivo baseada no VARFIT.

A Tabela 5-3 não só apresenta uma ausência de agrupamentos como também se nota que o método LSRS apresenta uma tendência a ficar preso em determinados valores de parâmetros, o que se nota pelo alinhamento de pontos em linhas retas ortogonais.

LSRS	
a × b/a	θ × b/a

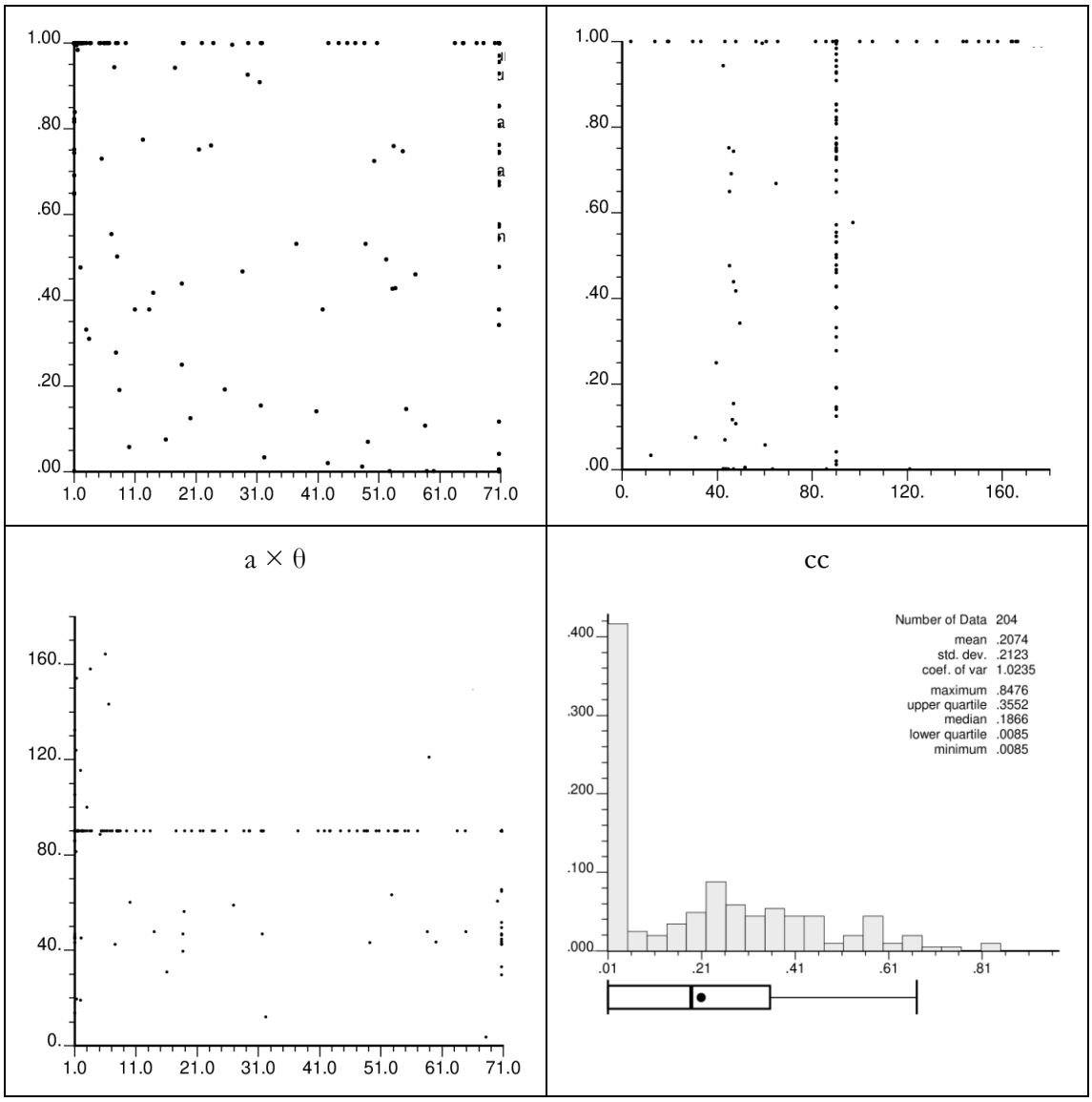


Tabela 5-3: Parâmetros do conjunto de estruturas variográficas imbricadas obtidas com o método de otimização LSRS com a função-objetivo baseada no VARFIT.

A Tabela 5-4 mostra praticamente nenhum agrupamento bem definido.

PSO	
a × b/a	θ × b/a

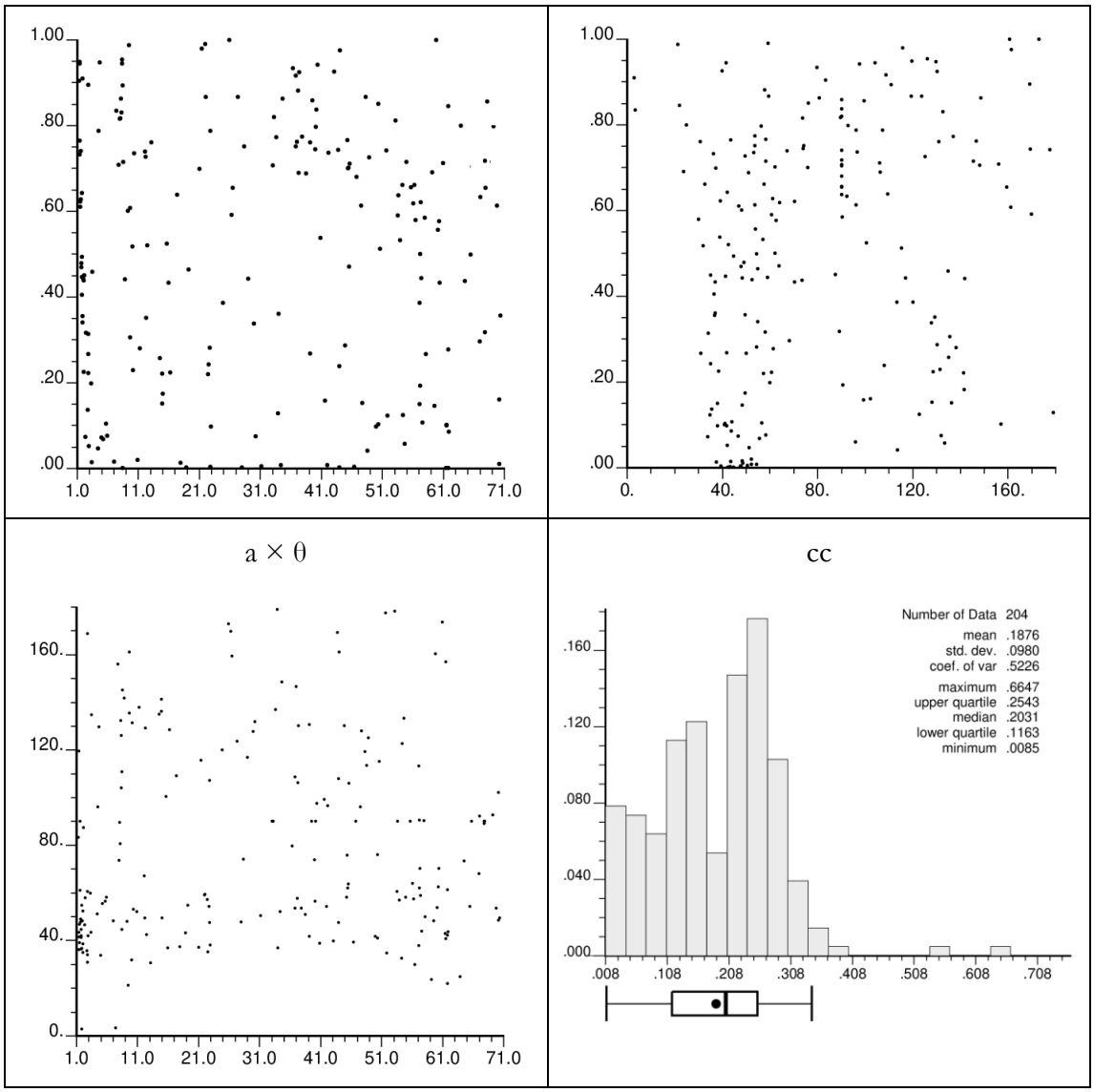


Tabela 5-4: Parâmetros do conjunto de estruturas variográficas imbricadas obtidas com o método de otimização PSO com a função-objetivo baseada no VARFIT.

A Tabela 5-5 mostra ao menos dois agrupamentos aparentemente separáveis, embora muito dispersos ainda, em torno dos azimutes N045E e N135E. Esse resultado sugere que o algoritmo de otimização GA proporciona uma maior eficácia na geração de agrupamentos separáveis por algoritmo.

GA	
a × b/a	θ × b/a

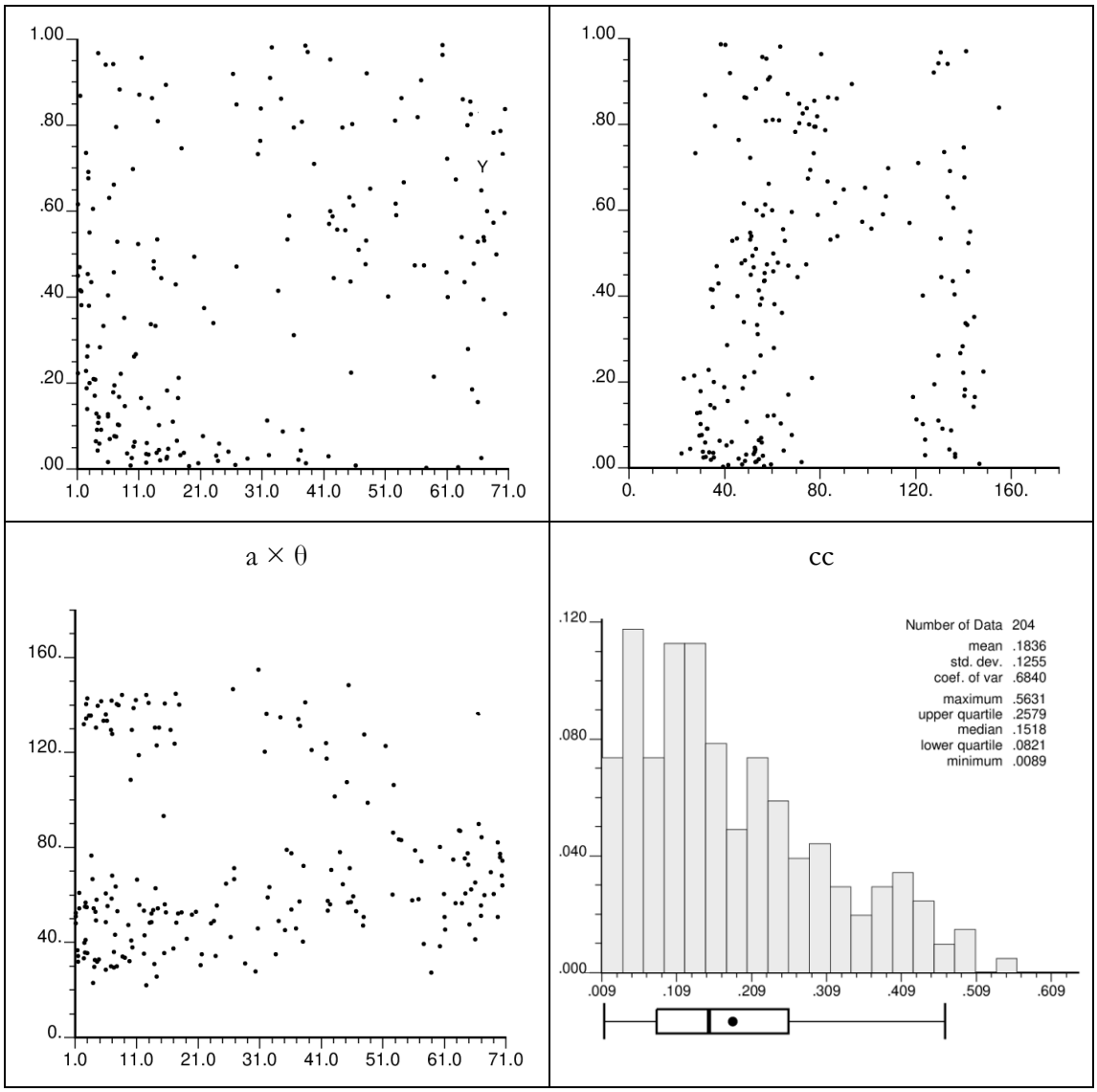


Tabela 5-5: Parâmetros do conjunto de estruturas variográficas imbricadas obtidas com o método de otimização GA com a função-objetivo baseada no VARFIT.

Não obstante o sucesso do programa `varfit` da GSLib e o resultado melhor obtido com o método de otimização GA, a função-objetivo do VARFIT estendida para o caso 2D não apresentou resultados satisfatórios. Uma explicação para esse comportamento é que a informação está concentrada nas células centrais das superfícies variográficas. Mesmo com a ponderação maior dada aos primeiros *lags* segundo a Equação 5-5, esses resultados preliminares mostraram que a função-objetivo faz com que um método de ajuste de variograma tenha baixa sensibilidade à variação dos parâmetros. Portanto, faz-se necessário propor outra função-objetivo.

5.5 Função-objetivo baseada no FIM

Tomando-se por base a soma dos quadrados das diferenças da Equação 5-7 e visando melhorar a qualidade dos fatores geomorfológicos obtidos, a função-objetivo foi modificada para a Equação 5-9. Nela, *M* é o mapa original e *M** é o mapa sintetizado a partir do variograma teórico com FIM (Seção 2.4), utilizando o processo apresentado na Figura 2-23. Ao usar o mapa no domínio original, as células do *grid* são portadoras de partes iguais de informação, portanto não é necessária uma ponderação. Com isso, teoricamente, a função-objetivo torna-se mais sensível aos parâmetros variográficos.

$$F = \sum_{j=1}^{J} \sum_{i=1}^{I} \left[M_{i,j} - M_{i,j}^{*} \right]^2$$

Equação 5-9: Função-objetivo baseada no FIM. Ver texto. *I* e *J* são as dimensões da malha.

De fato, em uma avaliação preliminar, observou-se melhora considerável na qualidade do ajuste. Por exemplo, com o algoritmo GA (parâmetros: #threads=4; #structures=4; seed=69069; varmap method=FIM-based; obj. function=FIM-based; Number of generations=100; Population size=1000; Natural selection size=500; Probability of crossover=0,7; Point of crossover=6 e Mutation rate=0,15) obtém uma separação exata das estruturas correspondentes aos artefatos, embora as outras duas estruturas não tenham correspondência exata com as duas estruturas geológicas (ver Tabela 1-1). Deve-se ter em mente, contudo, que o algoritmo de otimização conduz a um modelo variográfico que melhor se ajusta ao mapa variográfico experimental (ver Figura 5-7) e não necessariamente ao modelo variográfico usado para construir a imagem de referência com SGSIM (ver procedimentos na Seção 1.5).

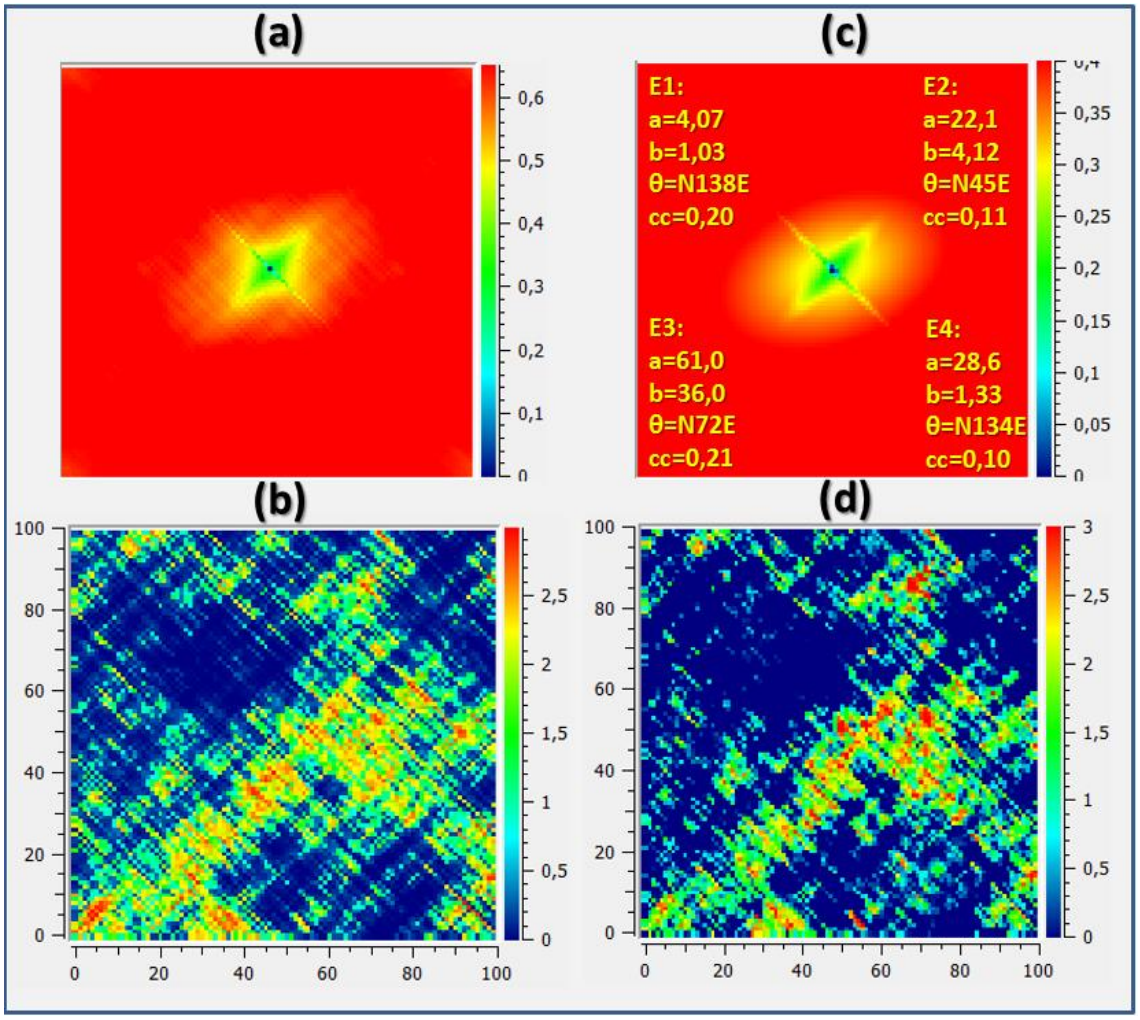


Figura 5-7: Exemplo de modelo variográfico ajustado automaticamente; (a): mapa variográfico do dado original (b); (c) modelo variográfico ajustado (E1 a E4 são os parâmetros das estruturas variográficas imbricadas); (d) mapa sintetizado com o modelo através do FIM (ver Seção 2.4).

5.6 Fluxo de ajuste automático de variograma baseado no FIM

Inicialmente, obtém-se o mapa de fases do mapa de entrada *M* com FFT (Seção 2.2.2 e Figura 5-8).

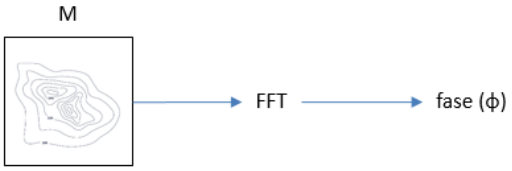


Figura 5-8: Passo 1 do fluxo automatizado de decomposição.

Em seguida, os parâmetros variográficos são iniciados no centro do domínio. O domínio para *a* vai do tamanho de uma célula (limite de Nyquist) à metade da diagonal do mapa. O domínio para b/a é fixado entre 1% e 100%. O domínio para θ naturalmente começa em

N000E e vai até N180E. O domínio para *cc* é entre 0.0 e a variância global do dado de entrada. Superfícies variográficas teóricas são geradas a partir dos parâmetros de cada estrutura imbricada (Seção 5.4.1 e Figura 5-9).

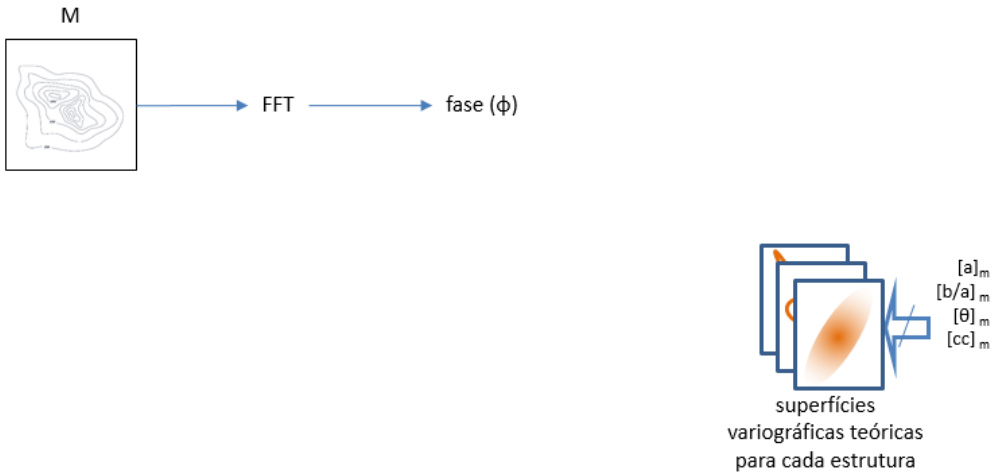


Figura 5-9: Passo 2 do fluxo automatizado de decomposição.

As superfícies teóricas são então somadas para formar o variograma teórico completo e aplica-se FIM (Seção 2.4) com o mapa de fases do mapa original e a densidade espectral obtida do modelo variográfico para sintetizar o mapa *M'* (Figura 5-10).

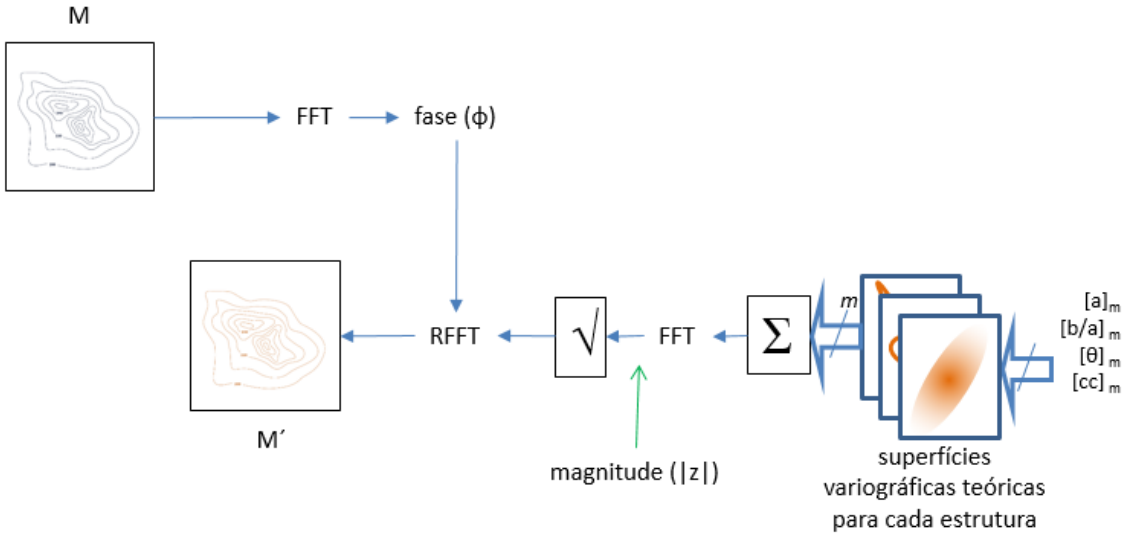


Figura 5-10: Passo 3 do fluxo automatizado de decomposição.

Finalmente, computa-se uma métrica da diferença entre *M* e *M´* (função-objetivo). Esse valor é passado para um método de otimização (Seção 2.1) que por sua vez altera os parâmetros variográficos visando reduzir esse valor na próxima iteração (Figura 5-11). As iterações terminam quando um critério de término for satisfeito, a depender do método de otimização escolhido pelo modelador.

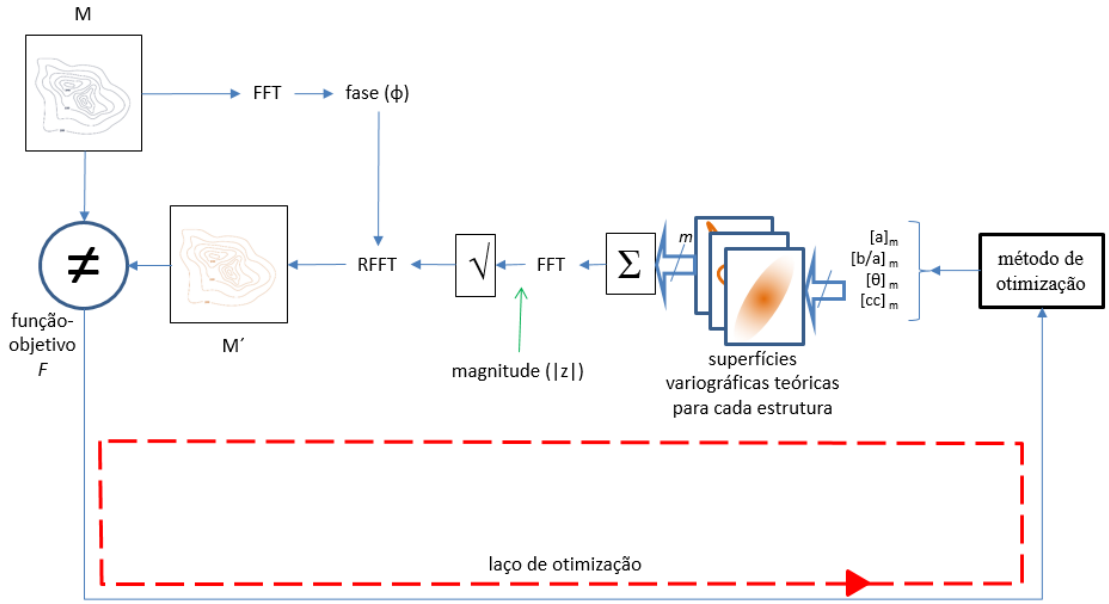


Figura 5-11: Esquema do fluxo automatizado de decomposição.

5.7 Análise de agrupamento (*cluster analysis*)

Conforme demonstrado na Seção 5.4.1, os algoritmos de otimização para alta dimensionalidade investigados são sensíveis ao caminho aleatório. Portanto, o fluxo de ajuste automático de modelo variográfico baseado no FIM (Seção 5.6) deve ser executado um certo número de vezes variando a semente para o gerador de números aleatórios.

Os parâmetros variográficos de cada estrutura imbricada encontrada existem em um espaço conhecido como espaço de feições (*feature space*) no jargão do aprendizado de máquina. O espaço de feições é um espaço Cartesiano cujos eixos coordenados correspondem aos parâmetros do modelo ao invés de coordenadas espaciais x, y e z. O espaço de feições também é conhecido por espaço de parâmetros.

Dependendo do número de execuções, espera-se que sejam formados agrupamentos (*clusters*) bem distinguíveis no espaço de feições em torno dos parâmetros da variografia "verdadeira" do fenômeno em estudo. Os agrupamentos são separados por um dos métodos de análise de agrupamento apresentados na Seção 2.6. Os centros desses agrupamentos são, por fim, os parâmetros de cada estrutura imbricada do modelo variográfico ajustado automaticamente.

5.8 Obtenção dos fatores geomorfológicos

Uma vez encontrados os parâmetros variográficos automaticamente, os fatores geomorfológicos correspondentes a cada estrutura imbricada devem ser obtidos como mapas para utilização. Uma forma de obtê-los é com o conhecido método FK (Seção 2.2.1). Outra forma, sem parametrização e de maior desempenho computacional, é com o processo baseado no FIM (Seção 2.4). Cada estrutura imbricada é avaliada em um *grid* e o mapa de fases do dado original são usados como entradas para o processo e as saídas são os mapas dos fatores geomorfológicos. A Figura 5-12 mostra o esquema desse processo.

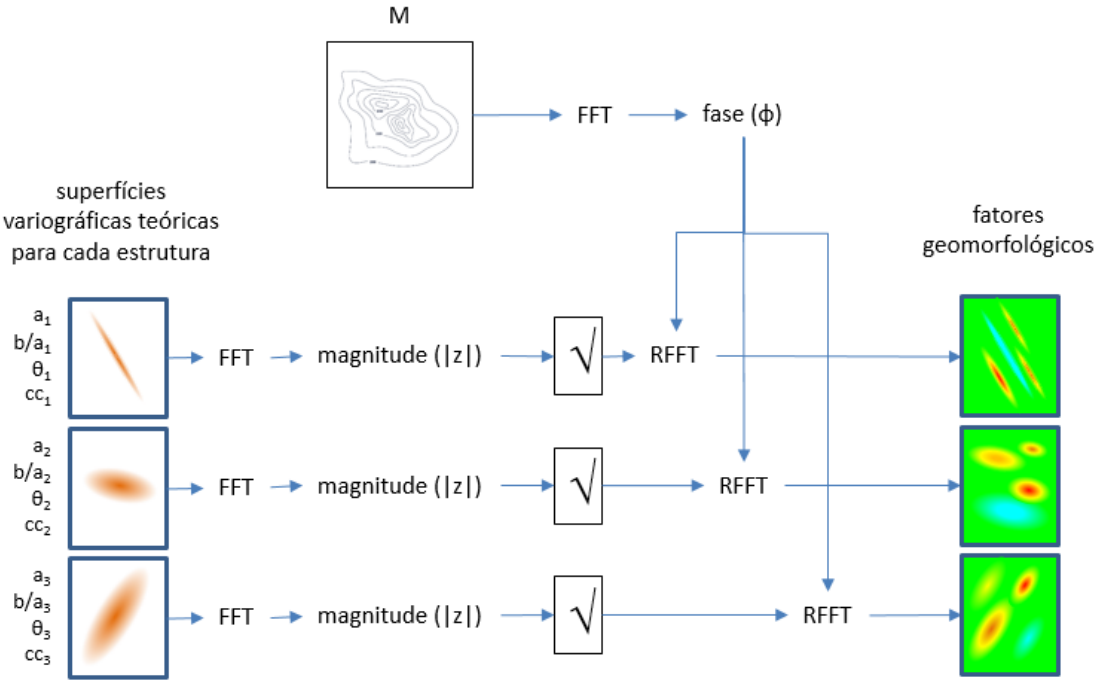


Figura 5-12: Fluxo de obtenção de fatores geomorfológicos a partir de parâmetros variográficos baseado no FIM.

5.9 Método completo de decomposição automatizada

A Figura 5-13 ilustra o macroalgoritmo completo do método proposto nesta tese. O processo de ajuste automático de variograma é o apresentado na Seção 5.6. Conforme os resultados preliminares apresentados e discutidos na Seção 5.4.1, o método de otimização usado no processo de ajuste de variograma é o GA (Seção 2.1.4). O algoritmo de análise de agrupamento será decidido no Capítulo 6 dentre os apresentados na Seção 2.6.

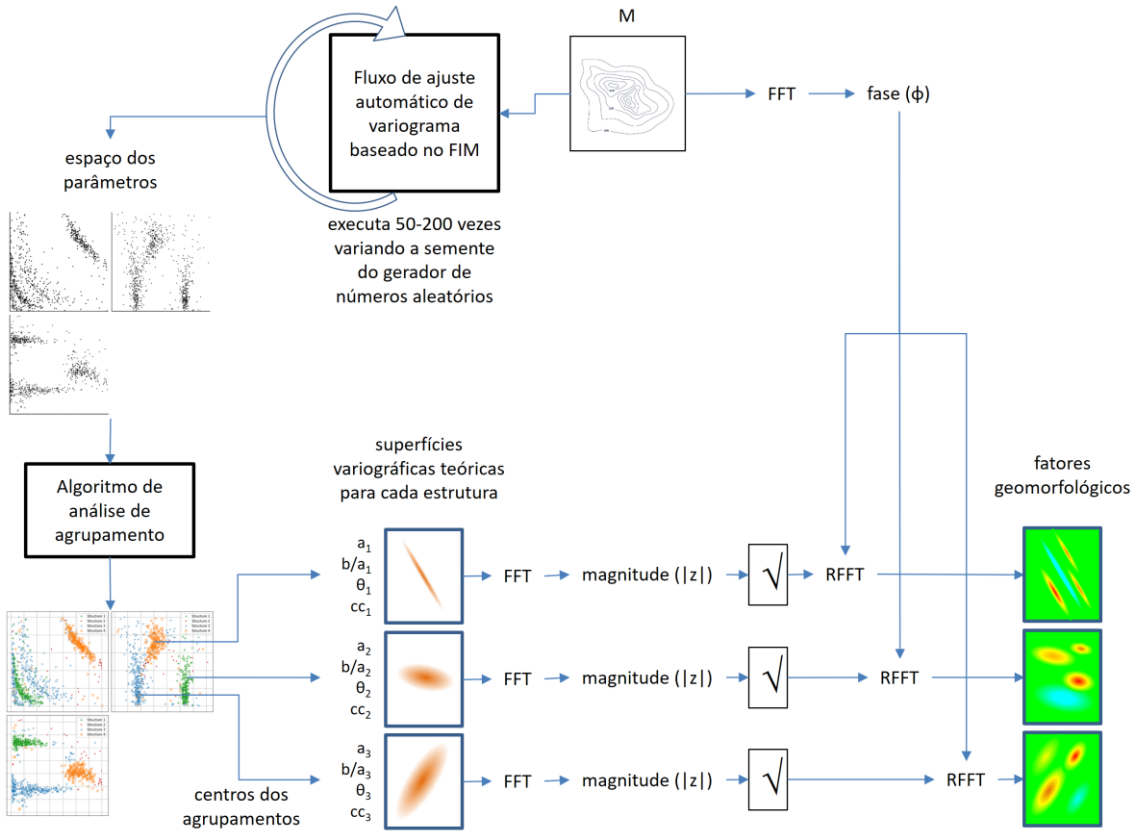


Figura 5-13: Fluxo completo de decomposição automatizada de um mapa *M* em fatores geomorfológicos.

5.10 Preparação para os experimentos

O método proposto pode ser testado no software GammaRay (ver Seção 1.5 Método). O programa pode ser obtido de [24].

Para o sistema operacional Windows, existe um arquivo .ZIP que contém, além de um executável, as bibliotecas de vínculo dinâmico (DLLs) pré-compiladas necessárias ao funcionamento do programa. Esse .ZIP deve ser descompactado em qualquer diretório e, para executar o programa, basta acionar `GammaRay.exe`.

Para outros sistemas operacionais deve-se obter os arquivos-fonte e compilar. Os arquivos-fonte estão disponíveis na mesma página de *download* do .ZIP do executável Windows. As instruções para compilação para os possíveis diferentes sistemas operacionais são deveras extensas e estão fora do escopo deste trabalho. As instruções para compilação estão no

[24] https://github.com/PauloCarvalhoRJ/gammaray/releases

manual do programa, que é o arquivo .PDF o qual se encontra também na mesma página de *download*.

Ao executar pela primeira vez, o programa encontra-se desconfigurado. Recomenda-se ler o manual do programa sobre como configurar os locais onde o programa encontrará os executáveis da GSLib e do GhostScript. Embora eles não sejam necessários para conduzir os experimentos, eles podem ser auxiliares úteis, tal como plotar um histograma.

Para os experimentos, faz-se necessário obter o dado de teste (ver Seção 1.5 Método) e adicioná-lo para uso no GammaRay. Ao abrir o programa, adicionar o dado de teste arrastando-o e soltando sobre o painel esquerdo (árvore de projeto) da janela principal. Também pode-se adicionar um dado ao estudo ao acionar o menu de contexto sobre o grupo "Data Files" (expandir a árvore de projeto no painel esquerdo) e acionando o item "Add data file...". Ao tentar adicionar, a janela da Figura 5-14 aparece (a aparência depende do sistema operacional usado) onde se deve declarar o tipo do dado sendo adicionado, no caso deve ser "GEO-EAS cartesian grid", pois o dado é uma malha regular.

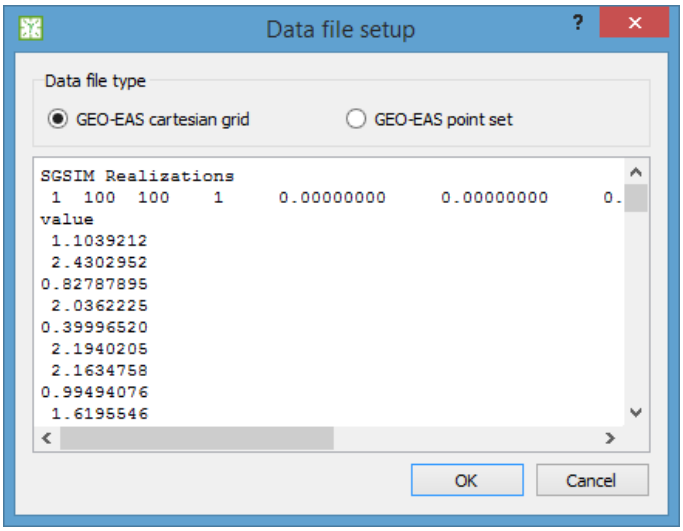


Figura 5-14: Janela para declarar o tipo do dado de teste.

Ao clicar "OK", a janela para configurar os metadados (geometria do *grid*) aparece (Figura 5-15). Preencher conforme mostrado na figura.

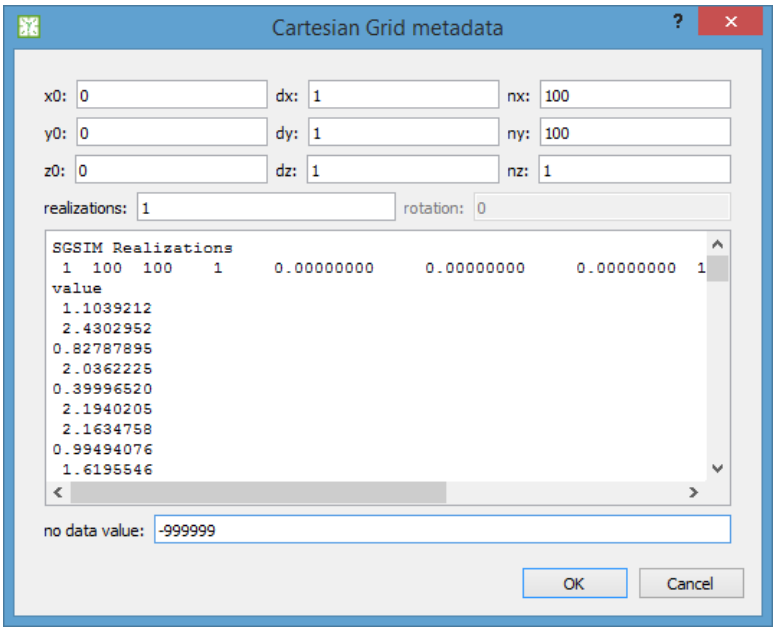


Figura 5-15: Janela para configurar os metadados do dado de teste.

Ao acionar "OK", em caso de sucesso, o dado é adicionado à árvore de projeto, que fica à esquerda da janela principal do programa. Ao expandir o item com o nome do arquivo do dado, deve-se encontrar uma única variável chamada "value". Verificar a correta configuração do dado acionando o menu de contexto sobre a variável "value" e acionando o item "Qick view". Deve aparecer um mapa semelhante ao da Figura 1-10. Verificar também se o mapa mede 100x100 unidades. A correta configuração dos parâmetros do *grid* é importante para manter a coerência com os parâmetros variográficos.

Para iniciar os experimentos, basta abrir o menu de contexto (normalmente com o botão direito do *mouse*) sobre a variável "value" e acionar o item "Automatic variogram fitting" para abrir a interface (Figura 5-16). A primeira página da janela mostra o dado de entrada e seu mapa variográfico. Primeiramente, deve-se configurar os parâmetros globais, localizados acima das abas conforme mostrado na figura.

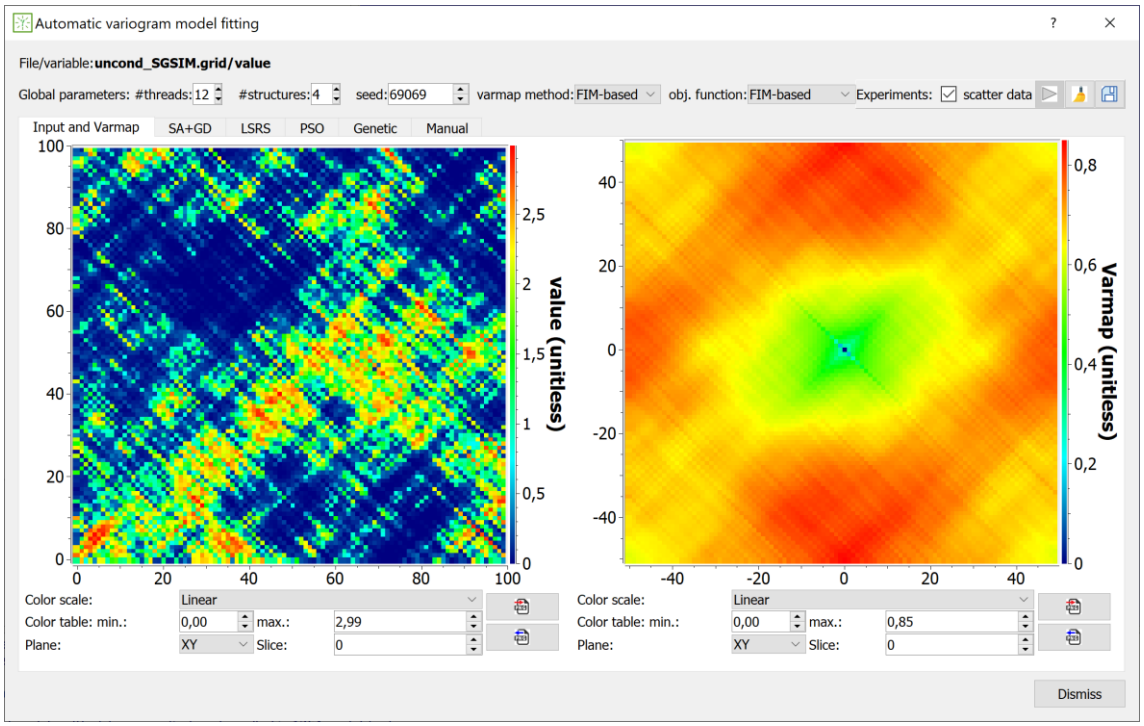


Figura 5-16: Tela de interface a ser usada nos experimentos.

i. **#threads:** é inicializado com o número de núcleos (*cores*) de processamento do sistema. Esse campo configura quantas linhas de execução (*threads*) o programa poderá disparar para reduzir o tempo de processamento. Normalmente o número de *threads* máximo para efetivo ganho de desempenho é de uma por núcleo de processamento. Um número menor pode ser especificado para deixar alguns núcleos livres para outras tarefas no computador durante um processamento mais longo.

ii. **#structures:** é o número de estruturas imbricadas (ou de fatores geomorfológicos) a serem encontrados.

iii. **seed:** é a semente para o gerador de números aleatórios. Esse valor será variado automaticamente nos experimentos.

iv. **varmap method:** é o método utilizado para computar o mapa variográfico de forma otimizada. Para reproduzir os mesmos resultados, recomenda-se usar "FIM-based", que corresponde ao fluxo apresentado na Seção 2.3. A outra opção ("Spectral") corresponde a uma forma diferente de cálculo implementada como a rotina `spectral::autocovariance()` no código-fonte do software.

v. **obj. function:** é o tipo de função-objetivo a ser minimizada. Para reproduzir os mesmos resultados, recomenda-se usar "FIM-based", que corresponde à função-

objetivo apresentada na Seção 5.5. A outra opção ("VARFIT-based") correponde à função-objetivo discutida na Seção 5.3.

Quatro das abas dessa tela correspondem a cada método de otimização (ver Seção 2.1) implementado. Marcando a caixa de seleção (*check box*) chamada "scatter data", o programa acumula na memória os parâmetros variográficos das estruturas imbricadas em todas as execuções dos experimentos. Essa memória pode ser limpa pressionando-se no botão com ícone de vassoura. O botão com ícone de disquete permite salvar essa memória como arquivo de nuvem de pontos (*point set*) no formato GEO-EAS/GSLib para análise de agrupamento (*cluster analysis*) com o Jupyter Notebook disponibilizado para esse fim (ver Seção 1.5). Neste ponto, os softwares estão prontos para realização dos experimentos apresentados no Capítulo 6.

Capítulo 6

Resultados e discussão

Neste capítulo são apresentados e discutidos os resultados obtidos com o fluxo automatizado de decomposição estrutural com algoritmo dedicado cujos passos estão detalhados nas Seções 5.5, 5.6, 5.7 e 5.8 e apresentado na Seção 5.9.

6.1 Análise de sensibilidade dos parâmetros

Nesta seção são apresentados e discutidos os perfis de convergência quando se varia um dos parâmetros relevantes para o algoritmo de otimização GA. Os motivos para seleção do algoritmo GA estão expostos na Seção 5.4.1. Convém ressaltar que somente os parâmetros centrais no comportamento do algoritmo foram testados. Parâmetros coadjuvantes como número de iterações, número de *threads* de execução ou épsilons (tolerâncias de comparação envolvendo números de ponto flutuante), não foram avaliados. A função-objetivo baseada no VARFIT não foi utilizada também pelos motivos expostos na Seção 5.4.1. Para os experimentos com o GA, configura-se como na Figura 6-1 e aciona-se o botão com ícone "play". Um pequeno diálogo se abre, onde se configura qual parâmetro variar, bem como os valores inicial, final e número de passos.

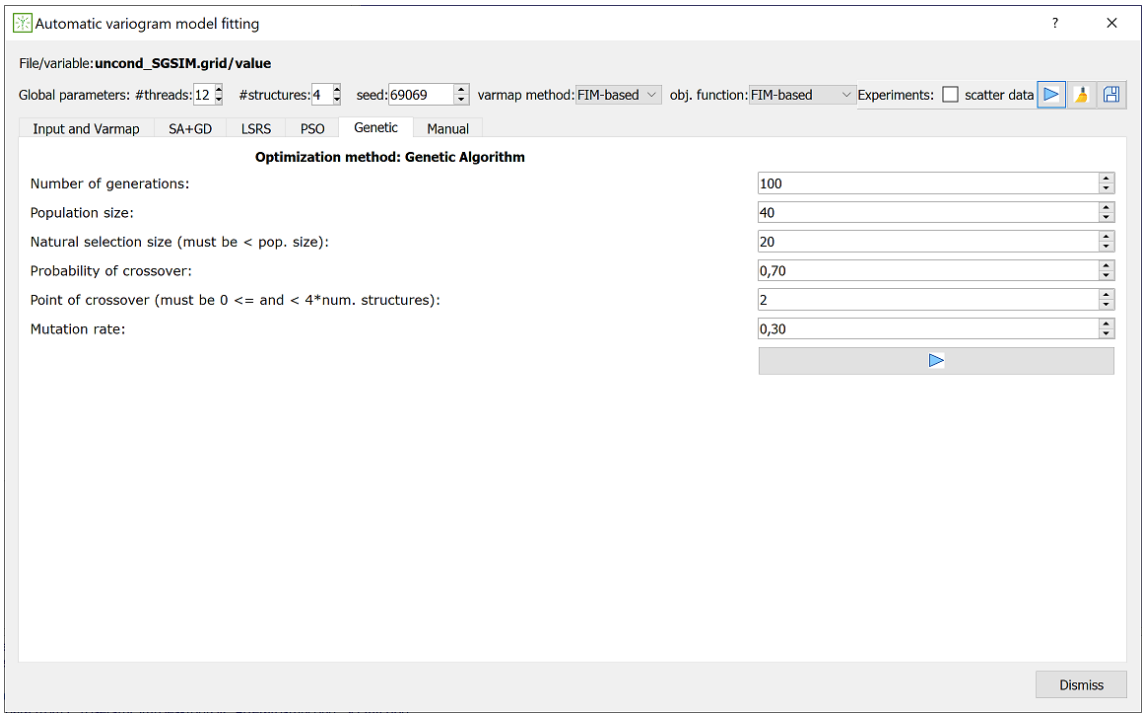


Figura 6-1: Configuração do algoritmo GA para avaliação da sensibilidade de parâmetros.

A Figura 6-2 mostra os perfis de convergência variando a semente para o gerador de números aleatórios. O algoritmo genético é mais "inteligente" por possuir regras mais sofisticadas de exploração do espaço de parâmetros do que tão somente percorrê-lo aleatoriamente. Assim, o caminho aleatório é menos impactante no valor-objetivo convergido.

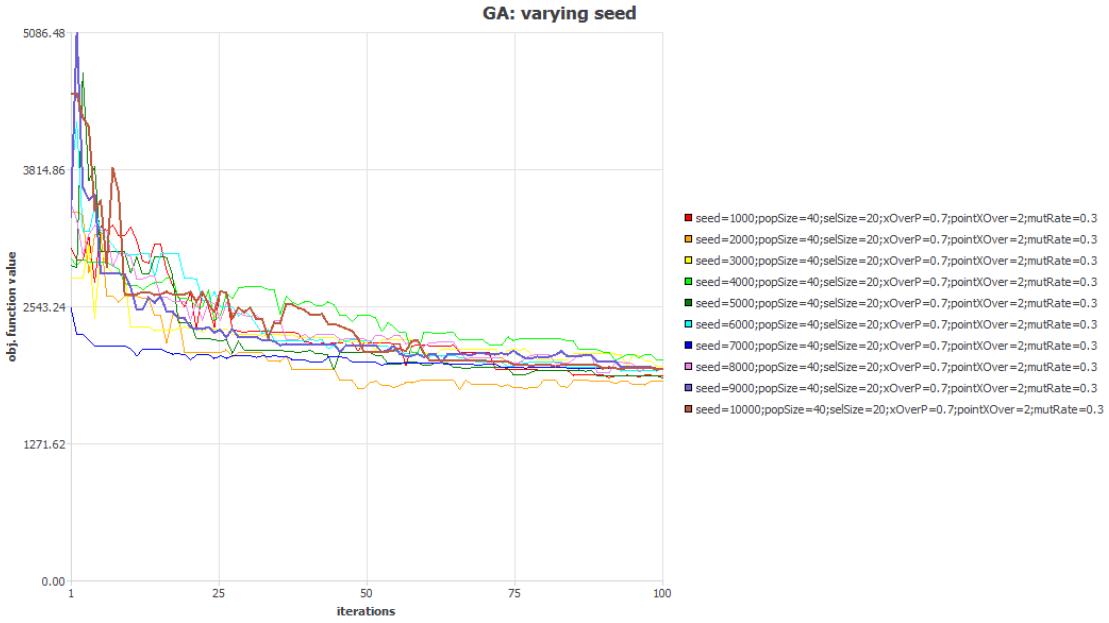


Figura 6-2: Perfis de convergência para o algoritmo genético (GA), variando a semente para o gerador de números aleatórios.

As curvas para os experimentos com o tamanho da população (Figura 6-3) mostram dificuldade para convergir conforme a população aumenta.

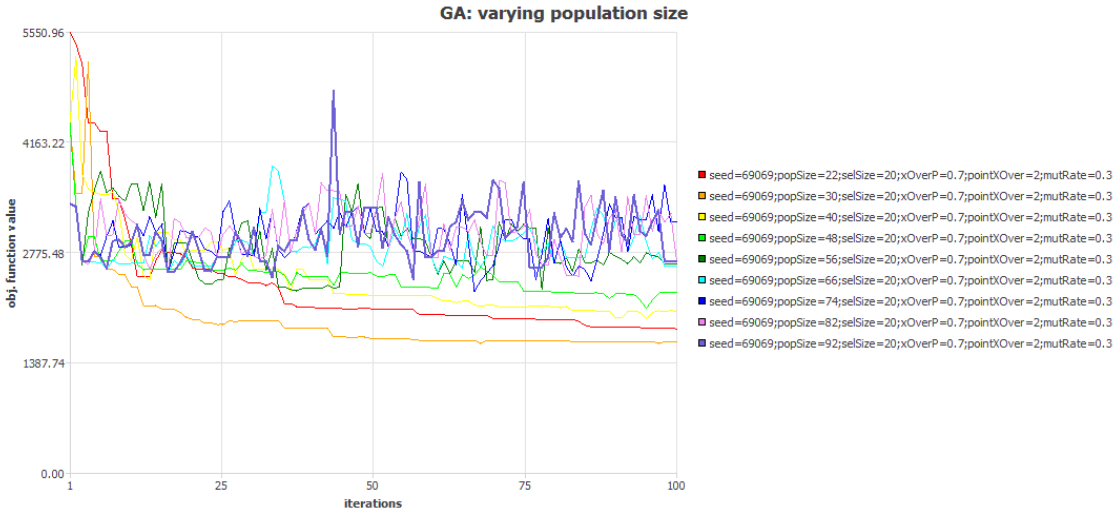


Figura 6-3: Perfis de convergência para o algoritmo genético (GA), variando o tamanho da população.

Variando-se o tamanho da seleção [dos mais aptos], os perfis de convergência (Figura 6-4) sugerem que o algoritmo converge melhor quando o *pool* de seleção é maior do que 50% do tamanho da população.

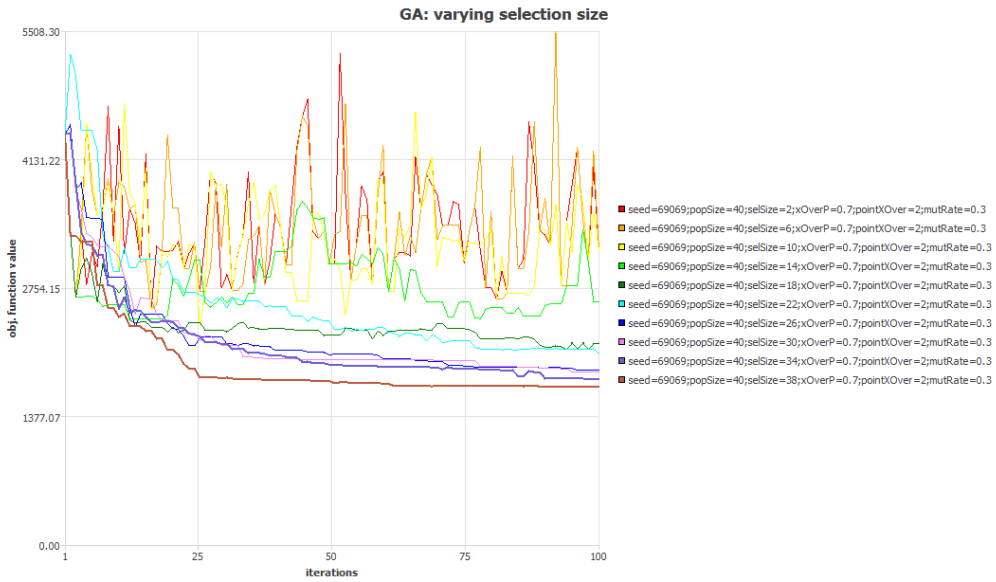


Figura 6-4: Perfis de convergência para o algoritmo genético (GA), variando o tamanho da seleção.

O experimento com a probabilidade de cruzamento (Figura 6-5) parece indicar que quanto maior a probabilidade de cruzamento, melhor é a convergência para a solução ótima. De fato, a evolução da vida na Terra acelerou depois da reprodução sexuada (troca de genes entre indivíduos), pois mutações aleatórias vantajosas que surgem em indivíduos diferentes

se combinam mais rápido nos descendentes do que se eles surgissem apenas por acaso em um mesmo indivíduo que não troca genes.

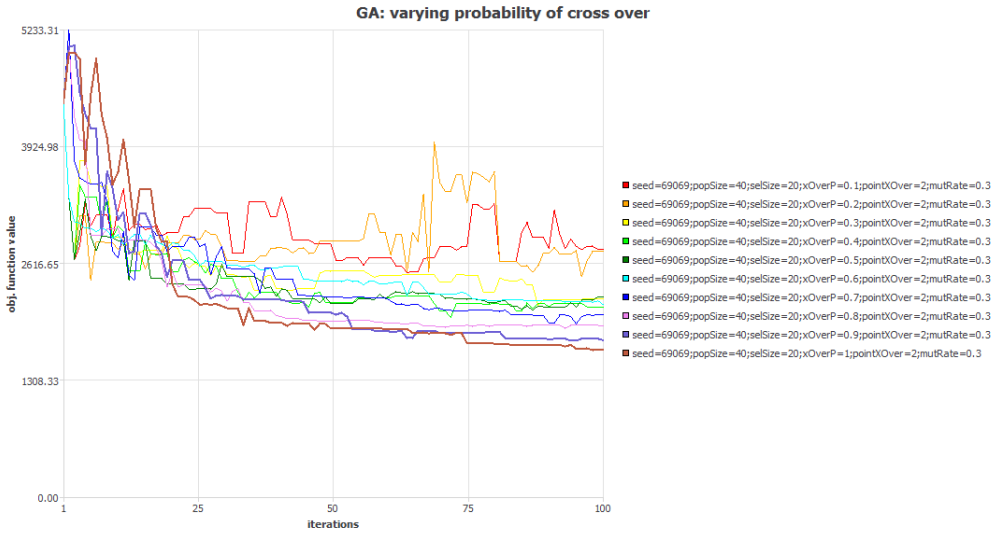


Figura 6-5: Perfis de convergência para o algoritmo genético (GA), variando a probabilidade de cruzamento.

As curvas obtidas (Figura 6-6) com a experimentação com o ponto de cruzamento (número do gene a partir do qual há troca de material genético) mostra ter pouca influência na convergência com a configuração inicial (Figura 6-1). Porém, os valores próximos do meio do genoma (8 para 16 parâmetros ou 16 genes) resultaram em uma convergência mais rápida.

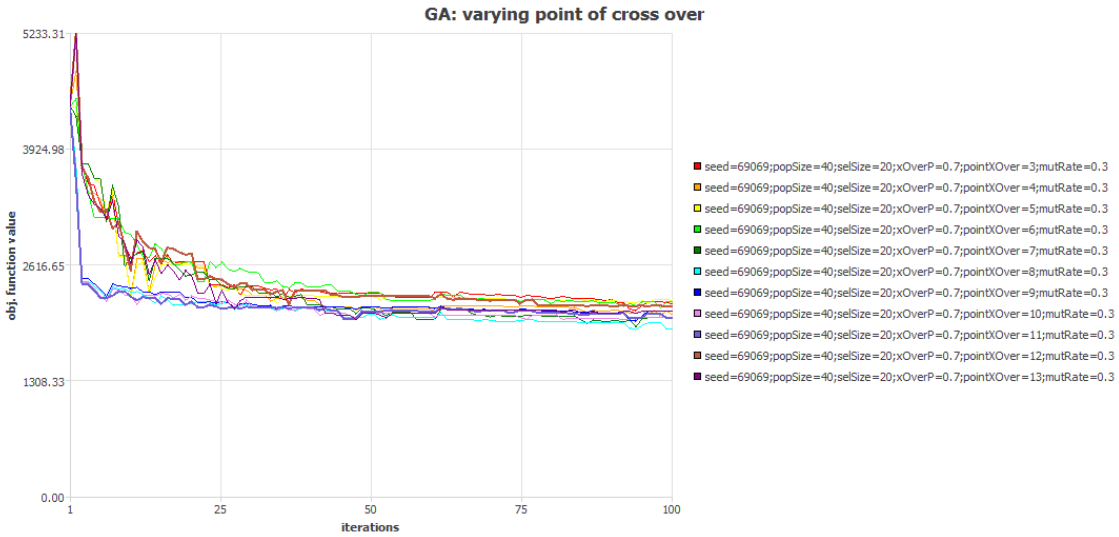


Figura 6-6: Perfis de convergência para o algoritmo genético (GA), variando o ponto de cruzamento.

Os perfis de convergência para a taxa de mutação (Figura 6-7) retratam o efeito esperado desse parâmetro: quanto maior, mais errática a convergência. Considerando a configuração inicial (Figura 6-1), os melhores valores-objetivos correspondem a taxas de mutação entre 30% e 60%.

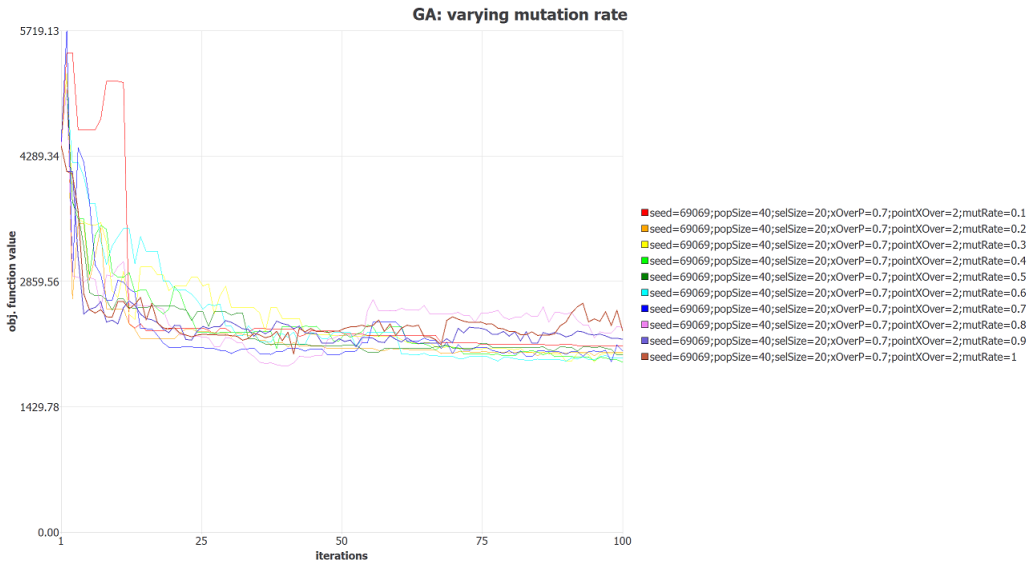


Figura 6-7: Perfis de convergência para o algoritmo genético (GA), variando a taxa de mutação.

6.2 Análise de agrupamento

Nesta seção são apresentados os quatro fatores geomorfológicos obtidos com a execução ao algoritmo dedicado de decomposição automatizada descrito na Seção 5.9. O algoritmo é executado em quatro sessões variando a semente do gerador de números aleatórios entre dois valores em 50 passos (limite do programa por sessão). A primeira sessão vai de 131313 a 171717, as demais: de 171717 a 232323, de 232323 a 272727 e de 272727 a 333333. Assim, são acumulados os parâmetros de 816 estruturas variográficas imbricadas no espaço de parâmetros: 4 sessões × 4 estruturas por modelo × (1 semente inicial + 50 passos). Com base na análise de sensibilidade dos parâmetros para o algoritmo de otimização GA feita na Seção 6.1, a parametrização desse algoritmo foi feita conforme Tabela 6-1. Por questão de simplicidade, as implementações dos algoritmos de otimização (incluindo GA) no software usam como gerador de números aleatórios a função tradicional `std::rand()`, cuja qualidade estatística da sequência gerada não está especificada e, portanto, depende da implementação na plataforma computacional alvo. Modernamente, recomenda-se o emprego da infraestrutura de geração de números aleatórios introduzida com o padrão C++11.

Number of generations:	100
Population size:	30
Natural selection size:	20
Probability of crossover:	1.00
Point of crossover:	8
Mutation rate:	0.45

Tabela 6-1: Parâmetros do algoritmo GA no software usado nos experimentos para obtenção dos resultados.

Os dados no espaço de parâmetros foram então analisados com cada um dos métodos de análise de agrupamento vistos na Seção 2.6. A funcionalidade de salvamento desses dados no software GammaRay está apresentado na Seção 5.10. A Figura 6-8 mostra os 816 resultados como pontos no espaço de parâmetros formado pelos parâmetros geométricos (*a*, *b*/*a* e *θ*). Conforme discutido anteriormente, o parâmetro *cc* não se espera variar para este caso, portanto não deve contribuir para diferenciação dos agrupamentos. Assim, *cc* foi apresentado como histograma.

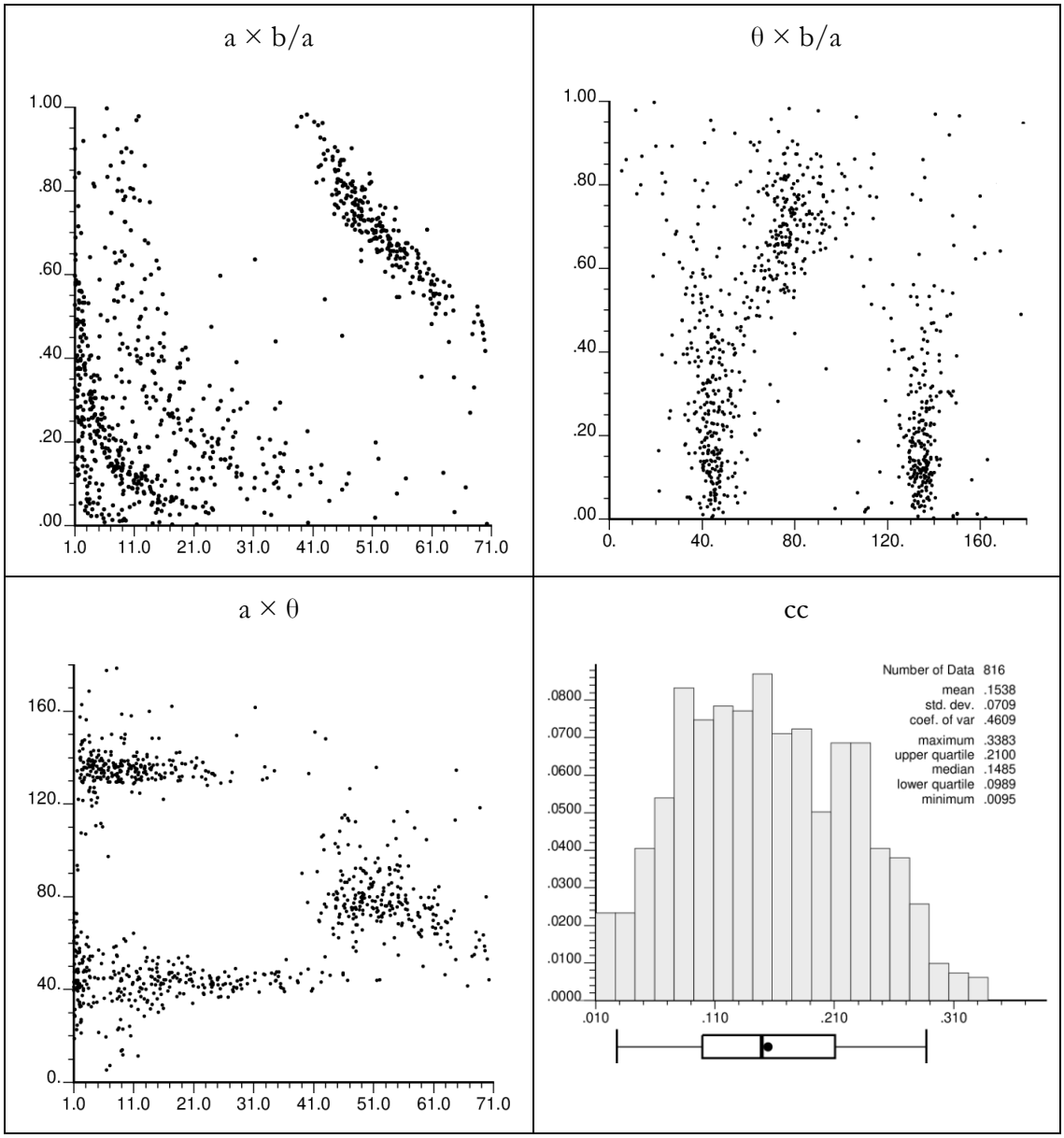
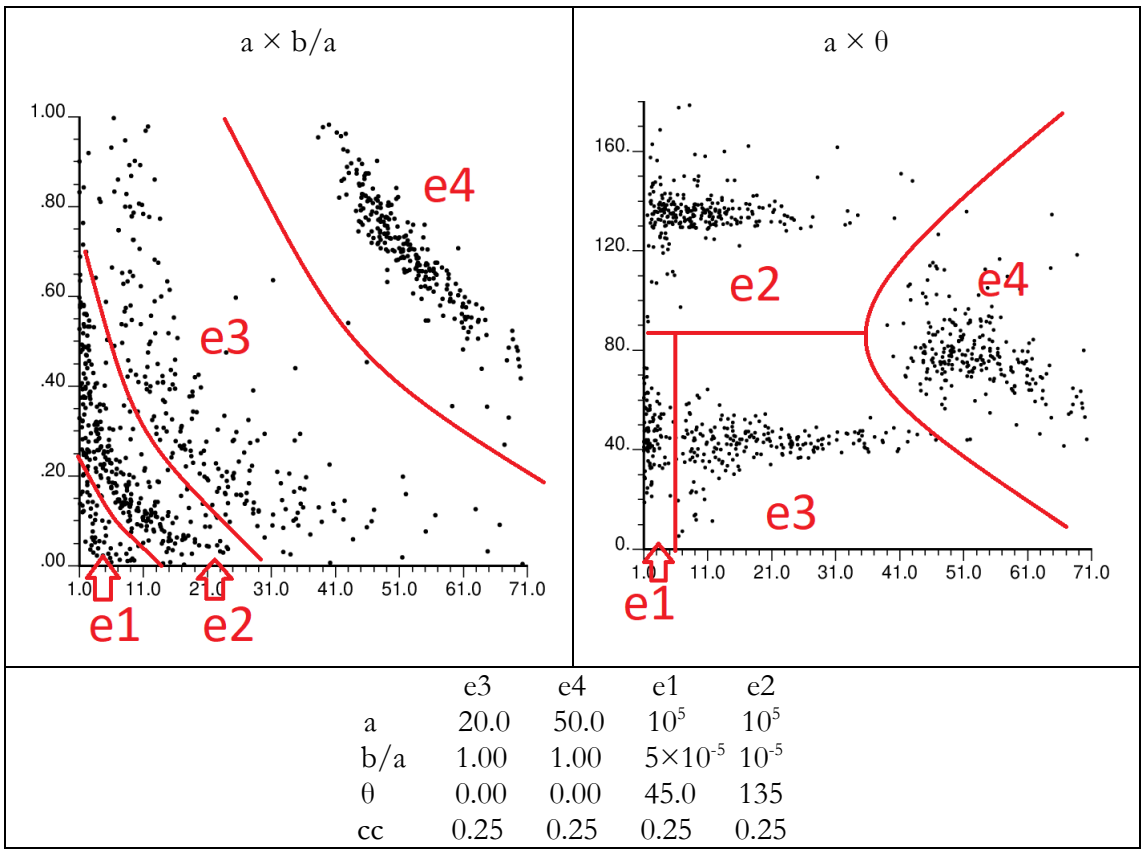


Figura 6-8: Resultados no espaço dos parâmetros variográficos.

Uma inspeção visual cuidadosa do espaço de parâmetros nos planos $a \times b/a$ e $a \times \theta$ permite divisar quatro agrupamentos tal como destacados na Figura 6-9. Nessa tabela também estão apresentadas também as quatro estruturas imbricadas do modelo variográfico usado para

gerar o dado de teste. Cada estrutura imbricada corresponde a cada um dos agrupamentos, assinalados por e1, e2, e3 e e4. Os agrupamentos e1 e e3 são mais difíceis de separar algoritmicamente.



	e3	e4	e1	e2
a	20.0	50.0	10^5	10^5
b/a	1.00	1.00	5×10^{-5}	10^{-5}
θ	0.00	0.00	45.0	135
cc	0.25	0.25	0.25	0.25

Figura 6-9: Agrupamentos perceptíveis visualmente no espaço de parâmetros (acima). Abaixo, o modelo variográfico usado na simulação do dado de teste e sua relação com os agrupamentos encontrados. Ver texto.

O passo final do processo de decomposição automatizada passa pela obtenção dos agrupamentos no espaço de parâmetros com um algoritmo de análise de agrupamento. As próximas seções apresentam o desempenho de cada um desses algoritmos (apresentados na Seção 2.6) na separação dos dados da Figura 6-8. As análises de agrupamento podem ser feitas com o Jupyter Notebook citado na Seção 1.5 e que pode ser baixado de[25]. Para os algoritmos que não aprendem o número de agrupamentos (ex.: K-means), foi usado o número 4 como número de agrupamentos, que corresponde ao número de estruturas imbricadas do modelo variográfico usado para gerar o dado para os testes. Para aplicações reais, contudo, o número de estruturas imbricadas não é conhecido *a priori*. Assim, o geomodelador deve experimentar com o número de agrupamentos nesse tipo de algoritmo.

[25] https://github.com/PauloCarvalhoRJ/pynb_paper_autovarfit2D

6.2.1 K-means

A Figura 6-10 mostra os agrupamentos obtidos com o K-means e os valores dos seus centros. Observa-se que o K-means, configurado para encontrar 4 agrupamentos, dividiu em dois o agrupamento correspondente à geologia de larga escala (50x50). Se for configurado para 3 agrupamentos, o K-means produz um resultado aceitável. O algoritmo também misturou os agrupamentos de difícil separação e1 e e3 assinalados na Figura 6-9.

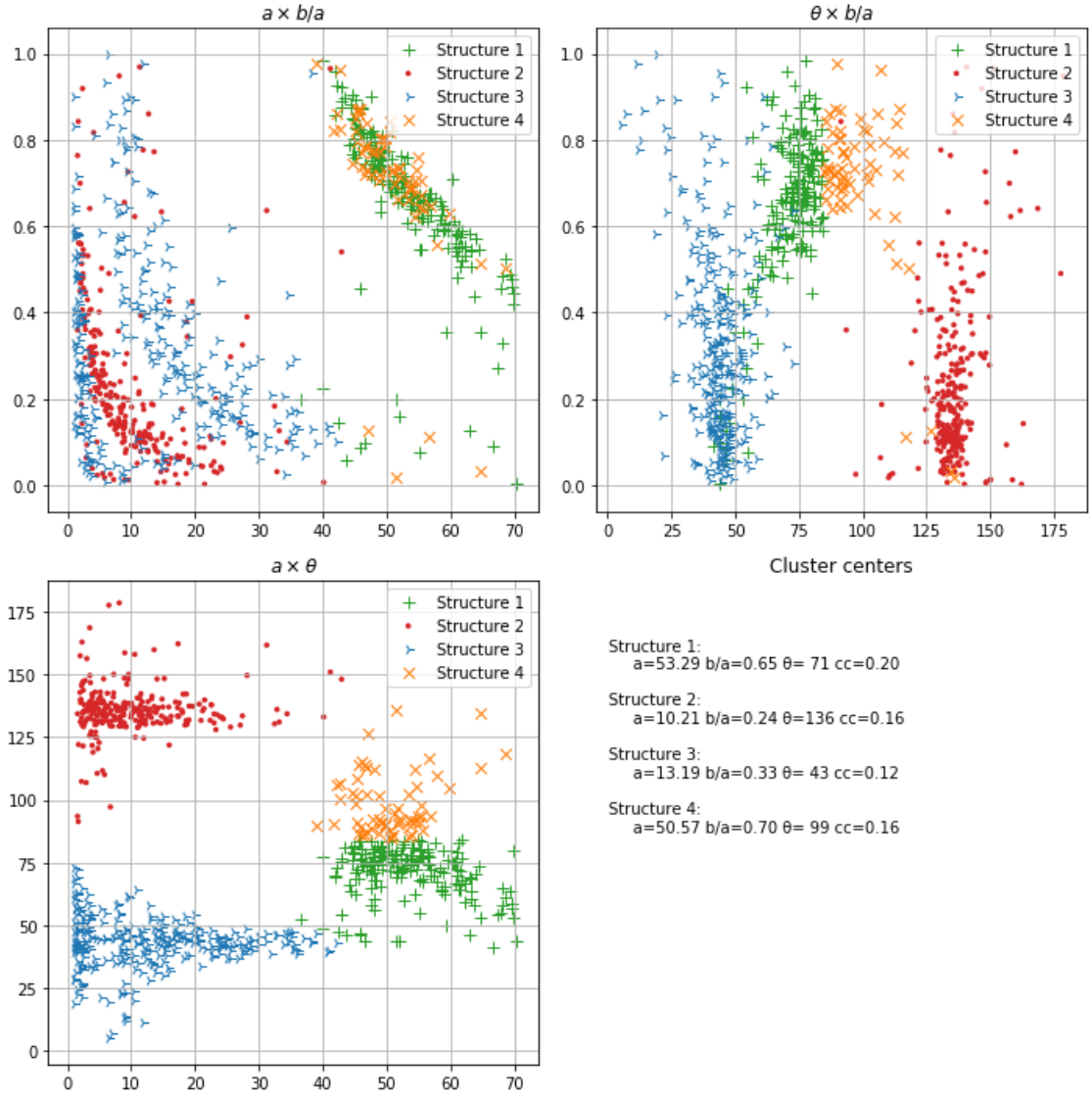


Figura 6-10: Agrupamentos identificados pelo o algoritmo K-means.

6.2.2 Agrupamento Espectral

A Figura 6-11 mostra os agrupamentos obtidos com o Agrupamento Espectral e os valores dos seus centros. Observa-se que esse algoritmo identifica corretamente o agrupamento

correspondente à geologia de larga escala (50x50). Contudo, como o K-means, também misturou os agrupamentos de difícil separação e1 e e3 assinalados na Figura 6-9.

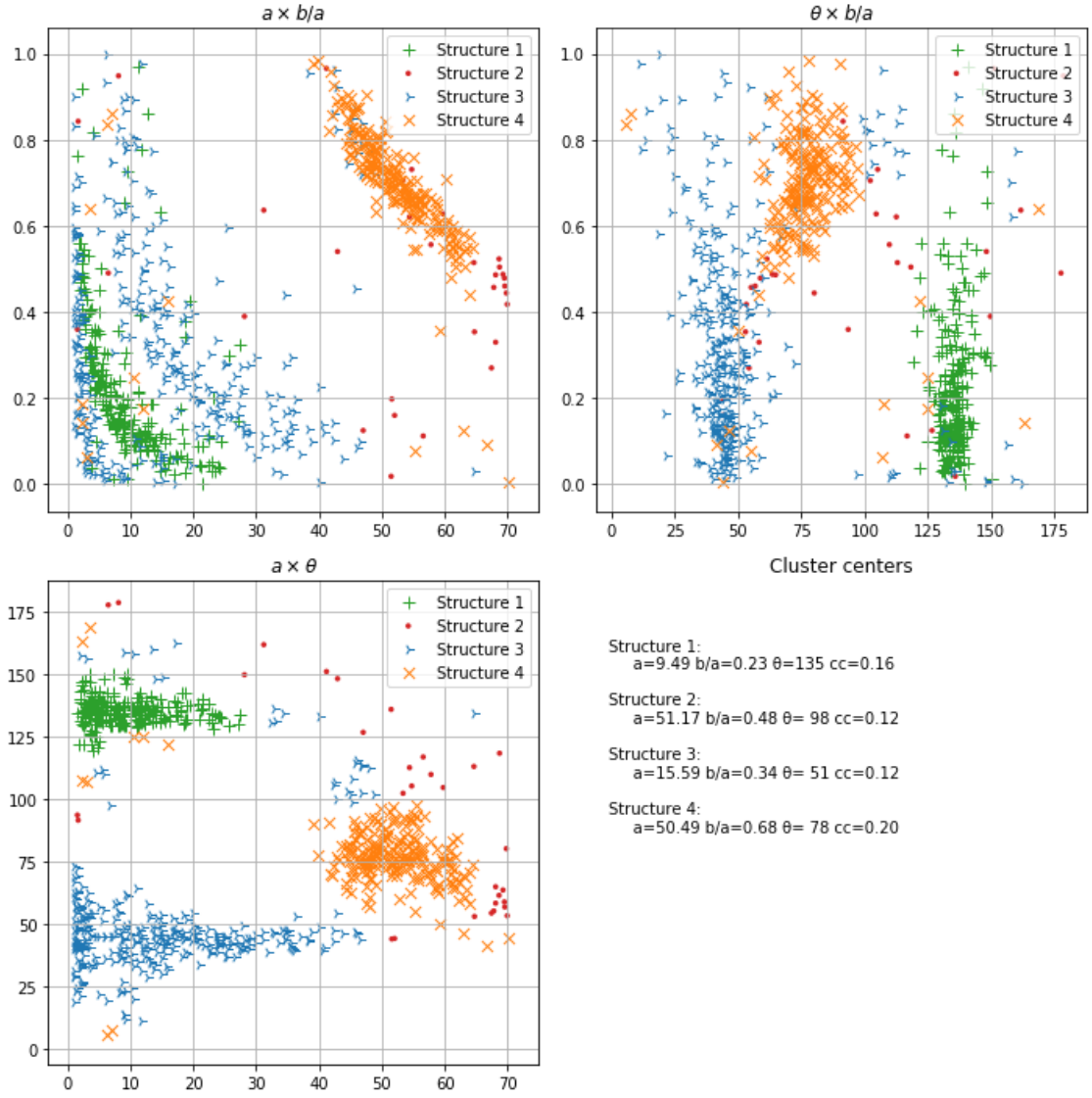


Figura 6-11: Agrupamentos identificados pelo o algoritmo Agrupamento Espectral.

6.2.3 Propagação de Afinidade

Esse algoritmo identificou automaticamente 44 agrupamentos, o que certamente é inaceitável.

6.2.4 Deslocamento da Média

A Figura 6-12 mostra os agrupamentos obtidos com o Deslocamento da Média e os valores dos seus centros. Observa-se que esse algoritmo apresentou resultado semelhante ao do Agrupamento Espectral, porém identificando os 3 agrupamentos aceitáveis automaticamente.

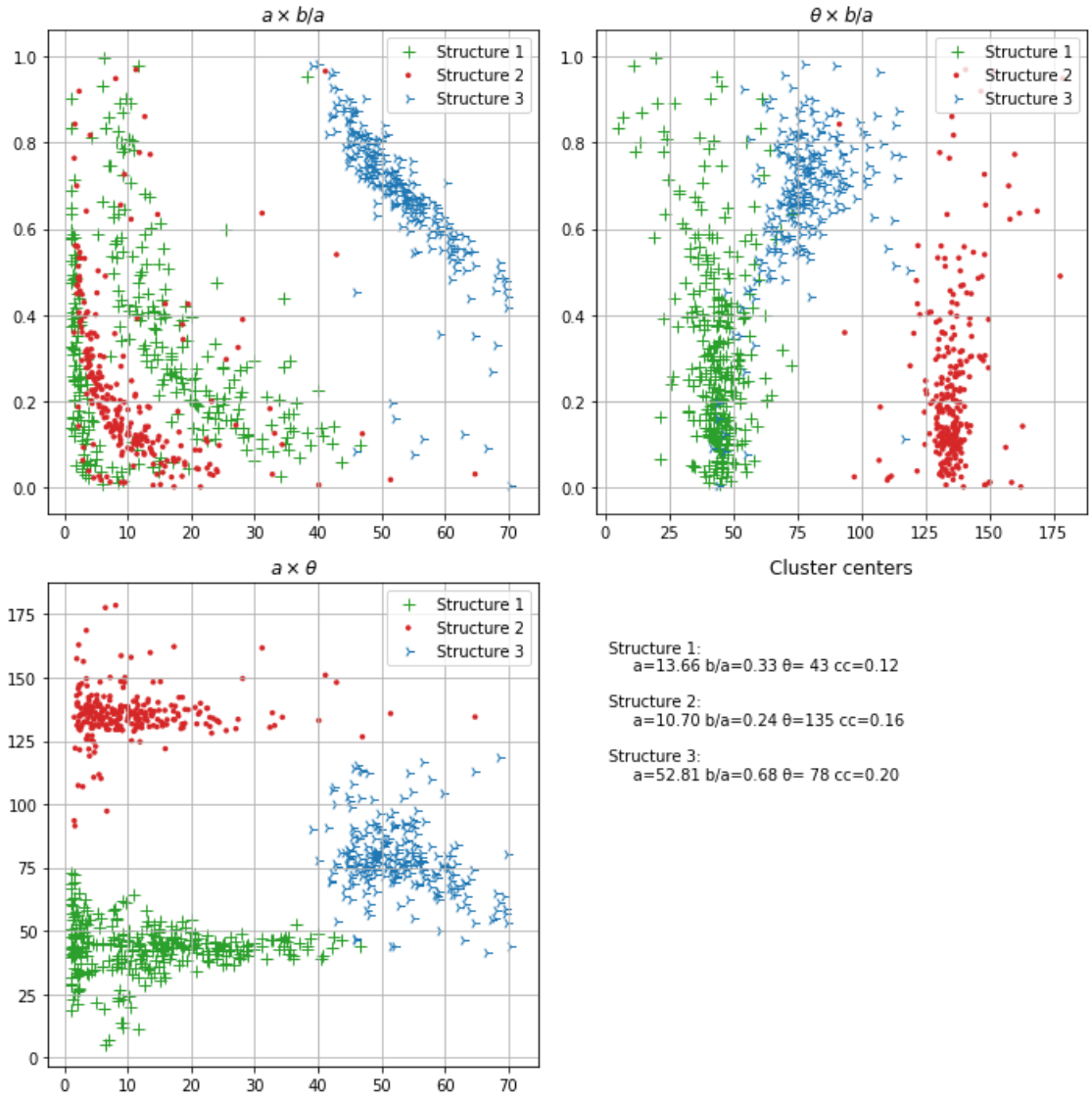


Figura 6-12: Agrupamentos identificados pelo o algoritmo Deslocamento da Média.

6.2.5 Agrupamento Aglomerativo

Esse algoritmo aceita um número inicial de agrupamentos (4), mas identificou somente dois, o que não é aceitável.

6.2.6 DBSCAN

A Figura 6-13 mostra os agrupamentos obtidos com o DBSCAN e os valores dos seus centros. O algoritmo reconheceu automaticamente as quatro estruturas conforme esperado (ver Figura 6-9) e ainda removeu pontos ruidosos (*outliers*) automaticamente que poderiam influenciar negativamente no valor dos centros. Os pontos identificados como ruído são marcados com o valor -1 na implementação. A remoção automática de pontos ruidosos também reduz a variância dos grupos, reduzindo a incerteza do ajuste.

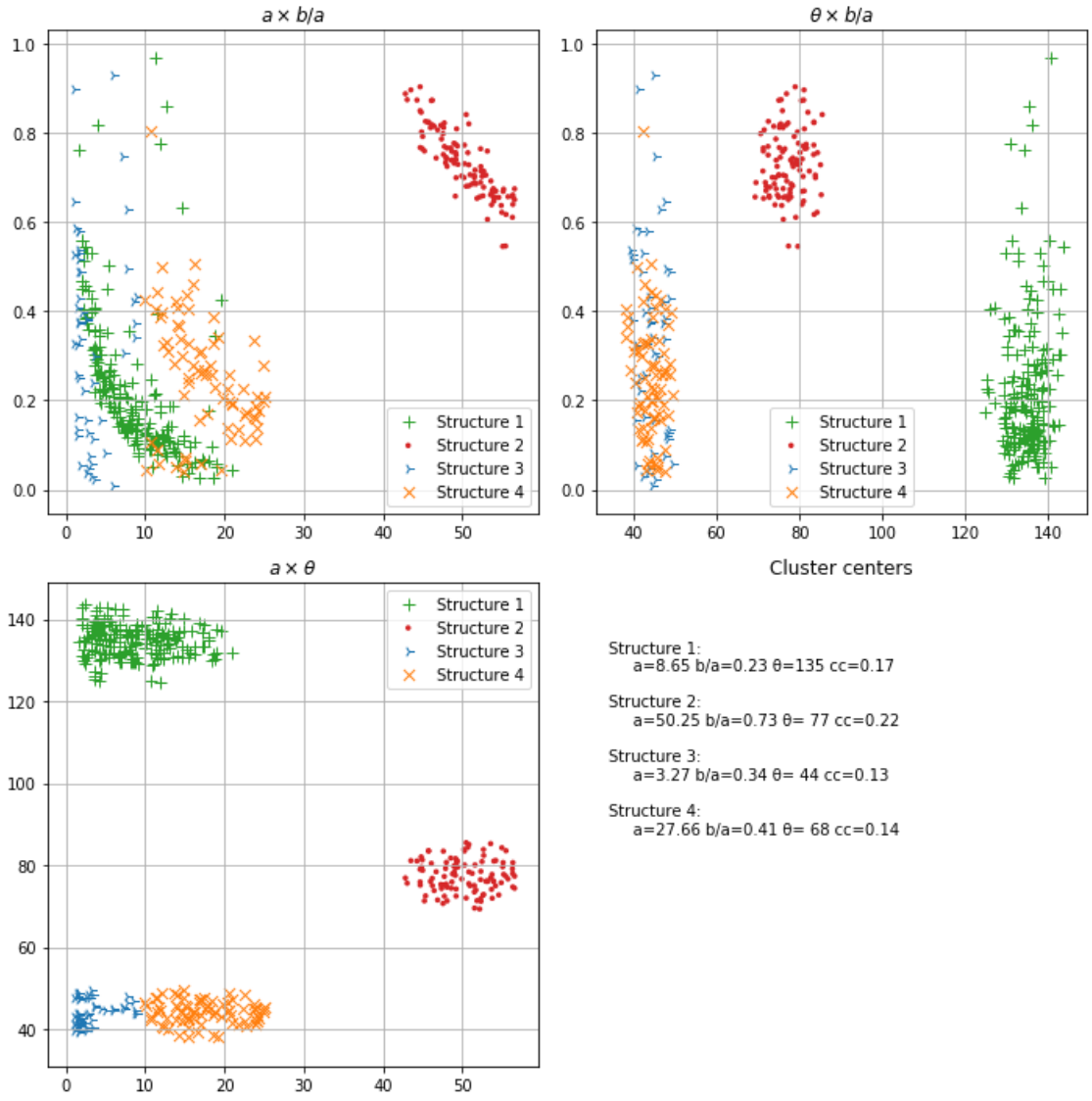


Figura 6-13: Agrupamentos identificados pelo o algoritmo DBSCAN.

6.2.7 BIRCH

A Figura 6-14 mostra os agrupamentos obtidos com o BIRCH e os valores dos seus centros. O algoritmo conseguiu apenas uma separação incompleta dos agrupamentos e1 e e3 assinalados na Figura 6-9, do contrário, os resultados seriam equivalentes aos do DBSCAN.

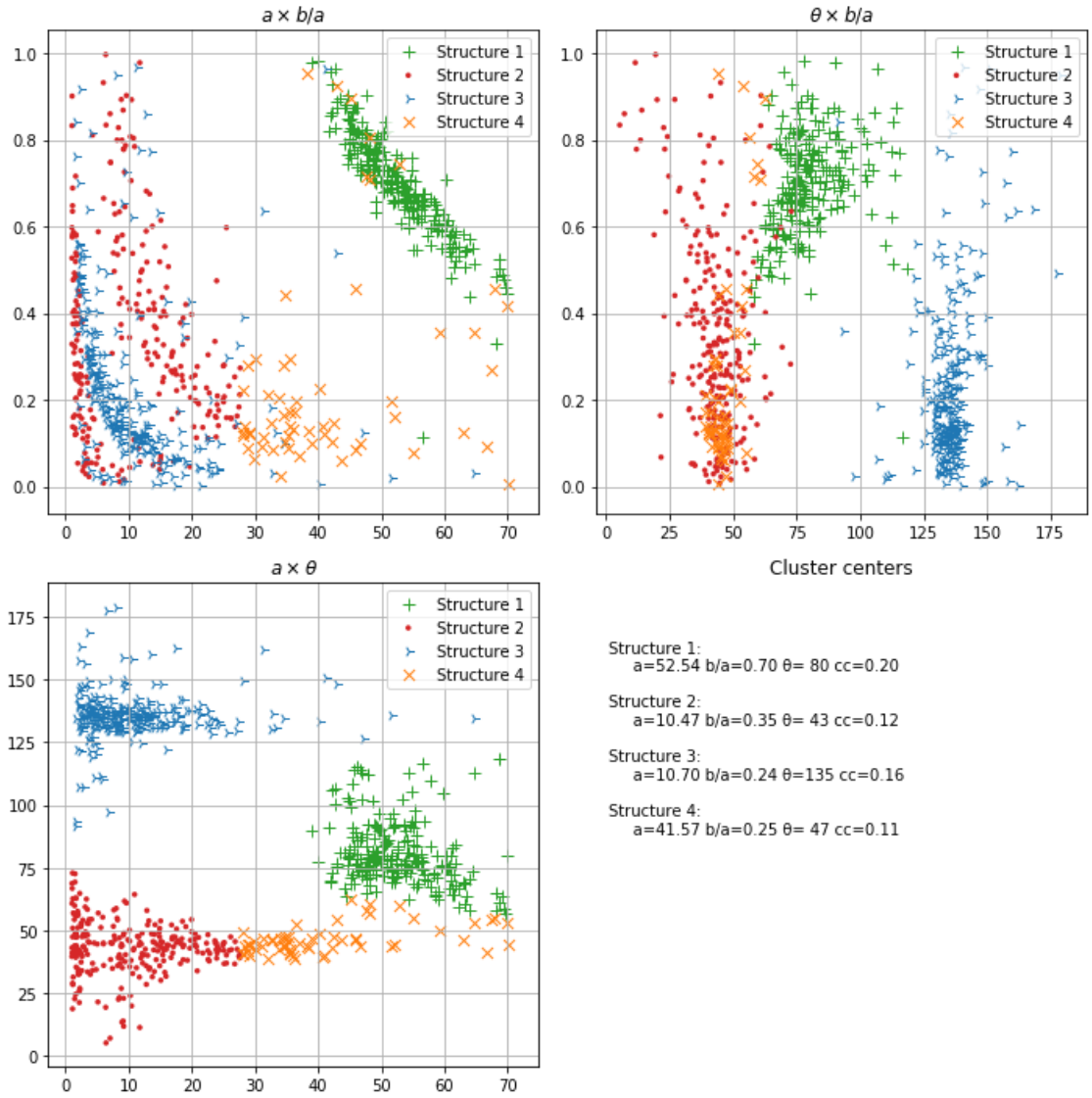


Figura 6-14: Agrupamentos identificados pelo o algoritmo BIRCH.

6.3 Aplicação a dados irregulares

O uso de FFT extensivamente por todo o fluxo proposto requer que o dado de entrada seja apresentando como uma malha regularmente espaçada. No entanto, é possível estender o método para dados irregulares se incluir um passo de regularização. Tal passo teria que ser computacionalmente barato em relação ao restante do fluxo. Um método assim pode ser a estimativa pelo Vizinho Mais Próximo perturbada com algum ruído. Pode-se pensar que tal passo adicional constitui uma alteração nos dados e, portanto, alterações na variografia. Entretanto, o modo tradicional para computar variogramas experimentais com *lags* e bandas azimutais podem ser vistos como um tipo de regularização que também muda a variografia "verdadeira". Se o geomodelador experimentar com os parâmetros como, por exemplo,

tamanho do *lag* e largura da banda, o arranjo dos pontos experimentais muda de forma também.

6.4 Extensão para 3D

A extensão para o caso tridimensional requer que cada estrutura imbricada seja representada por três parâmetros adicionais: ângulo de mergulho, ângulo de rolagem e razão 3º semieixo do elipsoide/semieixo menor. O primeiro e mais óbvio desdobramento é que o espaço de parâmetros passa a ter 7 dimensões ao invés de 4, tornando o emprego de um algoritmo de agrupamento ainda mais necessário, pois uma avaliação visual de grupos em 7 dimensões é praticamente inviável. Naturalmente, para 7 dimensões, um número significativamente maior de execuções do que 200 é requerido para obtenção de agrupamentos bem formados.

Outro desdobramento concerne os algoritmos de otimização. É necessário refazer a análise de sensibilidade (Seção 6.1) dos parâmetros dos algoritmos de otimização em face de 28 parâmetros (7 parâmetros × 4 estruturas imbricadas) contra os 16 do caso 2D. O método de Euler (Equação 2-3) usado na implementação dos métodos de otimização baseados em gradientes pode apresentar problemas numéricos com um número mais elevado de parâmetros, podendo requerer a implementação de um método mais sofisticado como o Runge-Kutta (Press *et al.*, 1992), normalmente o de 4ª ordem, conhecido como RK4 na indústria.

A síntese de mapas a partir do modelo variográfico (Seção 2.4) e cálculo de variograma (Seção 2.3) usados no algoritmo proposto se baseiam em FFT, o qual escala bem para volumes. Contudo, o uso de memória deve se tornar uma preocupação, pois a implementação de FT usada envolve ao menos três volumes em memória (uma entrada e duas saídas para FFT e duas entradas e uma saída para RFFT).

6.5 Desempenho computacional

Os experimentos foram executados em um sistema Intel Xeon Hexacore (6 núcleos de processamento físicos com *hyperthreading*) com *clock* de 2.9GHz e 32GB de RAM. A função-objetivo proposta tem evidentemente um custo computacional maior do que a versão 2D da função-objetivo baseada no VARFIT. Enquanto a última requer apenas a avaliação do modelo variográfico em uma malha regular, a primeira requer uma chamada à FFT e à RFFT para cada ciclo de otimização. Em geral, foi observado que o tempo de execução com a função-objetivo baseada no VARFIT foi de cerca de 30% do tempo com a função proposta. Todavia, FFT e sua reversa executam em O(n log n) e a geração do modelo variográfico em

grid executa em O(n), esse percentual será menor conforme o tamanho do mapa aumenta (ver notação *big*-O na Seção 2.3). Para a malha de 100×100 células, cada experimento levou entre 2 e 15 segundos para completar, a depender do método de otimização escolhido, com o a função-objetivo baseada no FIM. O requisito de memória foi insignificante para a malha em questão.

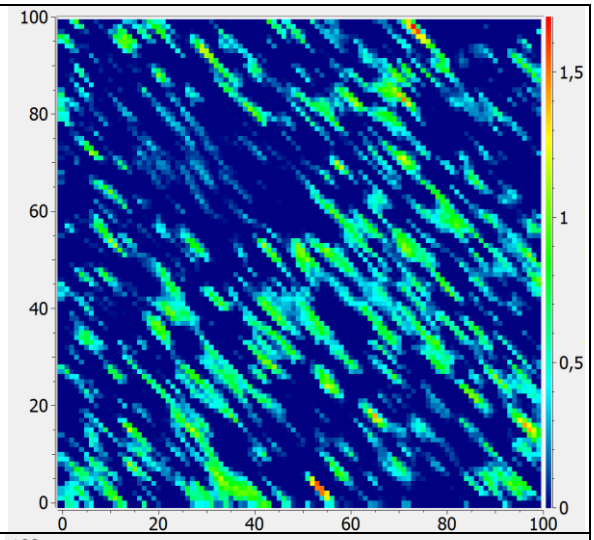
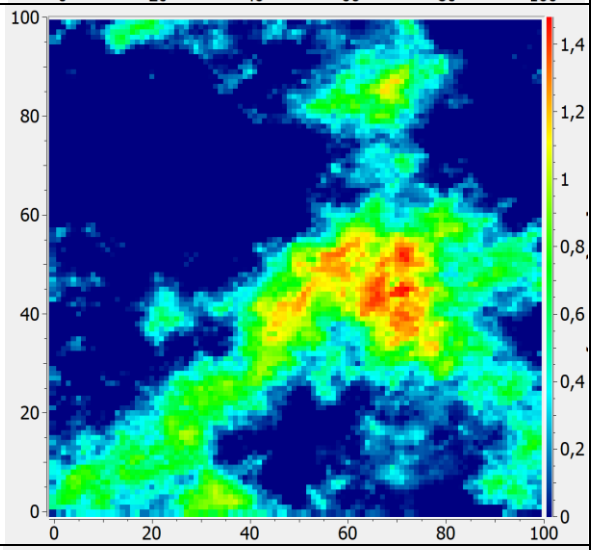
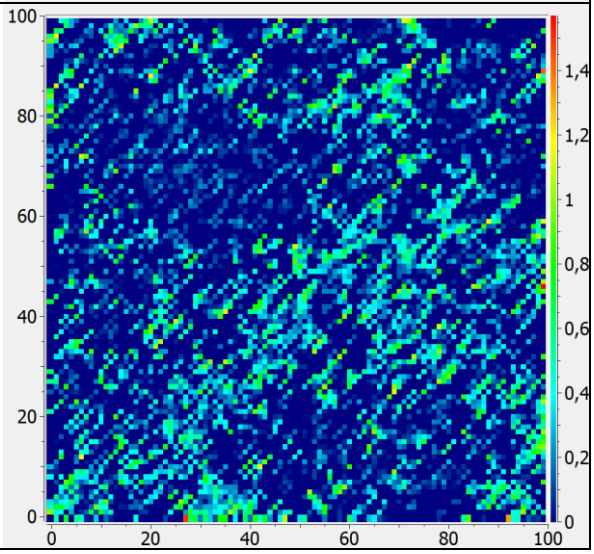
A geração dos dados em espaço de parâmetro requereu 204 repetições do processo de ajuste automático de variograma, o que levou 6 minutos para completar (usando GA como método de otimização e o método baseado no FIM como função-objetivo). Um estudo regional não estacionário medindo 1000×1000 células divididas em 100 quadrículas de 100×100 levaria cerca de 10 horas para completar. Pode não parecer muito promissor, mas o esforço e o tempo requerido a um especialista humano para ajustar 100 variogramas 2D locais, possivelmente com estruturas imbricadas, certamente justifica o método com ampla margem.

Quanto aos métodos de análise de agrupamento investigados, a implementação do Agrupamento Espectral empregada neste trabalho processou os 816 pontos no espaço paramétrico em aproximadamente 1 minuto. A Propagação da Afinidade, cerca de 5 segundos para completar a mesma tarefa. Os demais métodos, inclusive o DBSCAN que resultou nos melhores agrupamentos, terminaram a computação em menos de 1 segundo.

6.6 Fatores geomorfológicos obtidos

Indubitavelmente, o DBSCAN foi o método de análise de agrupamento que apresentou o melhor resultado (parâmetros encontrados na Figura 6-13). Assim, os valores dos centros dos agrupamentos identificados automaticamente no espaço de parâmetros foram usados para realizar a decomposição estrutural conforme fluxo ilustrado na Figura 5-13. A Tabela 6-2 mostra os parâmetros das estruturas variográficas imbricadas obtidas automaticamente pelo método proposto neste trabalho e os respectivos mapas do fatores geomorfológicos. Resultados julgados satisfatórios.

parâmetros	fator geomorfológico

a=8,65; b/a=0,23; θ=135; cc=0,17	
a=50,25; b/a=0,73; θ=77; cc=0,22	
a=3,27; b/a=0,34; θ=44; cc=0,13	

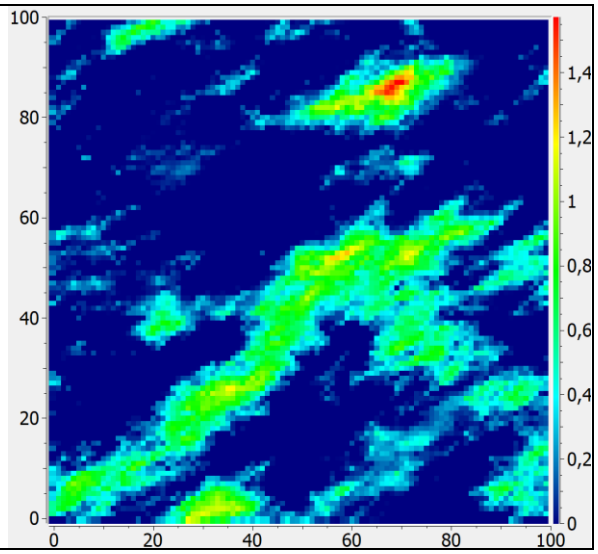
a=27,66; b/a=0,41; θ=68; cc=0,14	

Tabela 6-2: Os fatores geomorfológicos obtidos automaticamente com o método proposto na tese.

Convém chamar atenção quanto ao não determinismo dos parâmetros ajustados. Os agrupamentos possuem normalmente uma dispersão dentro da qual encontram-se outros valores que poderiam ser considerados válidos, não somente os centroides dos agrupamentos.

Para avaliar o resultado, foram realizadas duas krigagens fatoriais ordinárias (OFK) usando o mapa original de teste (Figura 1-10) como entrada. Lembrando que FK é o método de decomposição usado neste trabalho como referência. A primeira usando o variograma *a priori* cujos parâmetros estão na Tabela 1-1. A segunda com o variograma cujos parâmetros foram encontrados pelo método proposto (Tabela 6-2). Ambas OFKs foram realizadas com os mesmos parâmetros (ex. estratégia de busca), diferindo apenas nos variogramas. O software GammaRay usado nos experimentos inclui uma implementação de FK (simples e ordinária). A Figura 6-15 mostra a média de krigagem (*kriging of the mean* – 1º fator) obtida com OFK. O mapa de efeito pepita (2º fator) é zerado e não está mostrado.

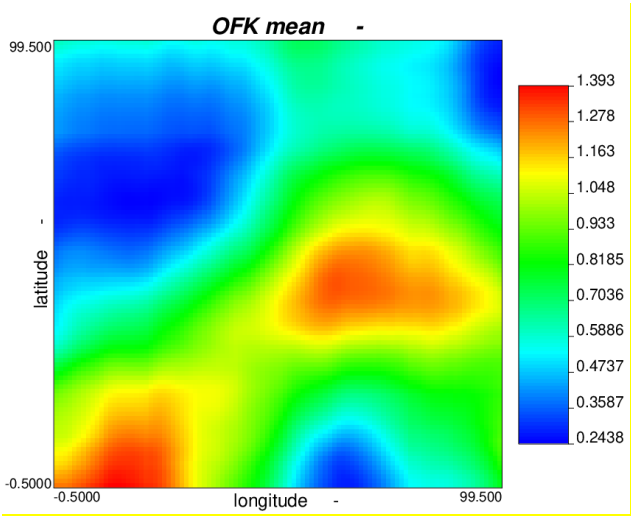


Figura 6-15: Mapa da média de krigagem das duas operações de FK ordinária usadas para avaliar o resultado.

A Figura 6-16 mostra os quatro fatores geomorfológicos obtidos com OFK para o dado de teste usando o variograma *a priori*.

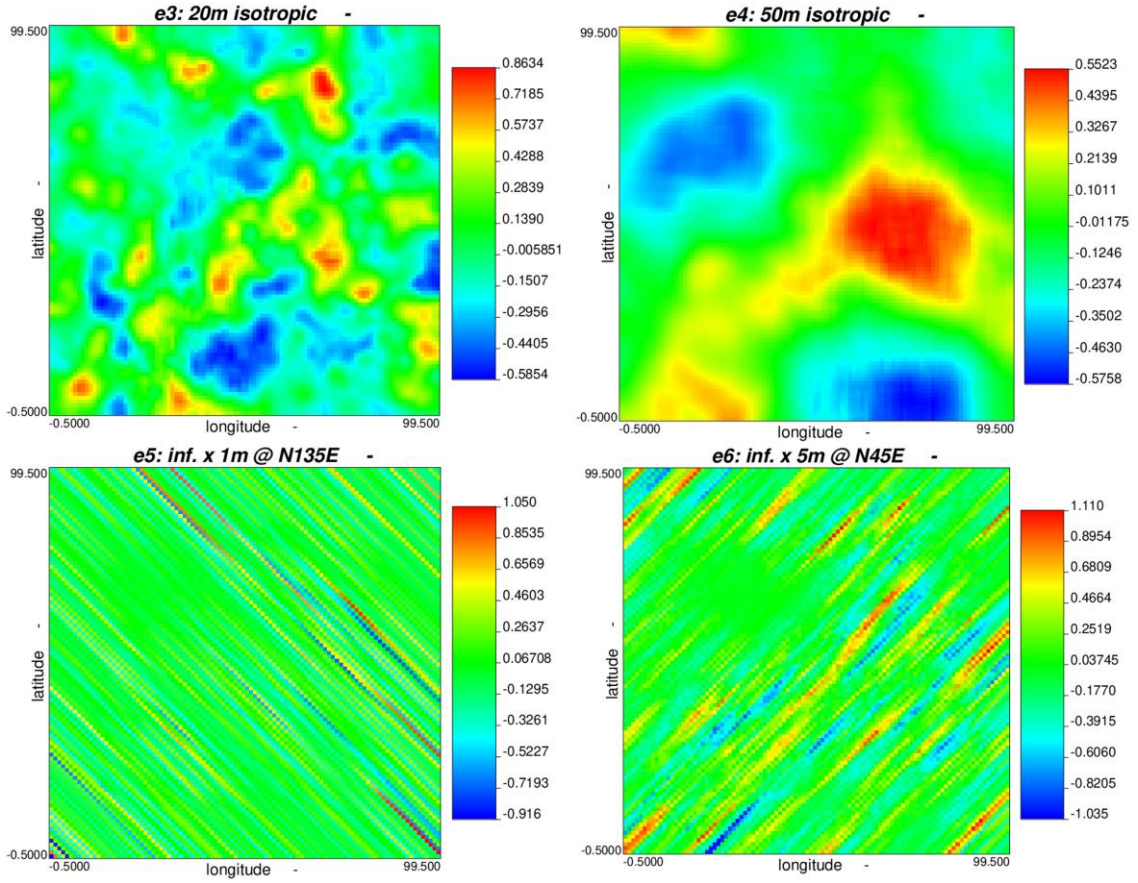


Figura 6-16: Os quatro fatores geomorfológicos obtidos com OFK a partir do variograma *a priori* usado para gerar o dado de teste com SGSIM. Todas as contribuições (cc) são 0,25.

A Figura 6-17 mostra os quatro fatores geomorfológicos obtidos com OFK para o dado de teste usando o variograma ajustado automaticamente com o método proposto.

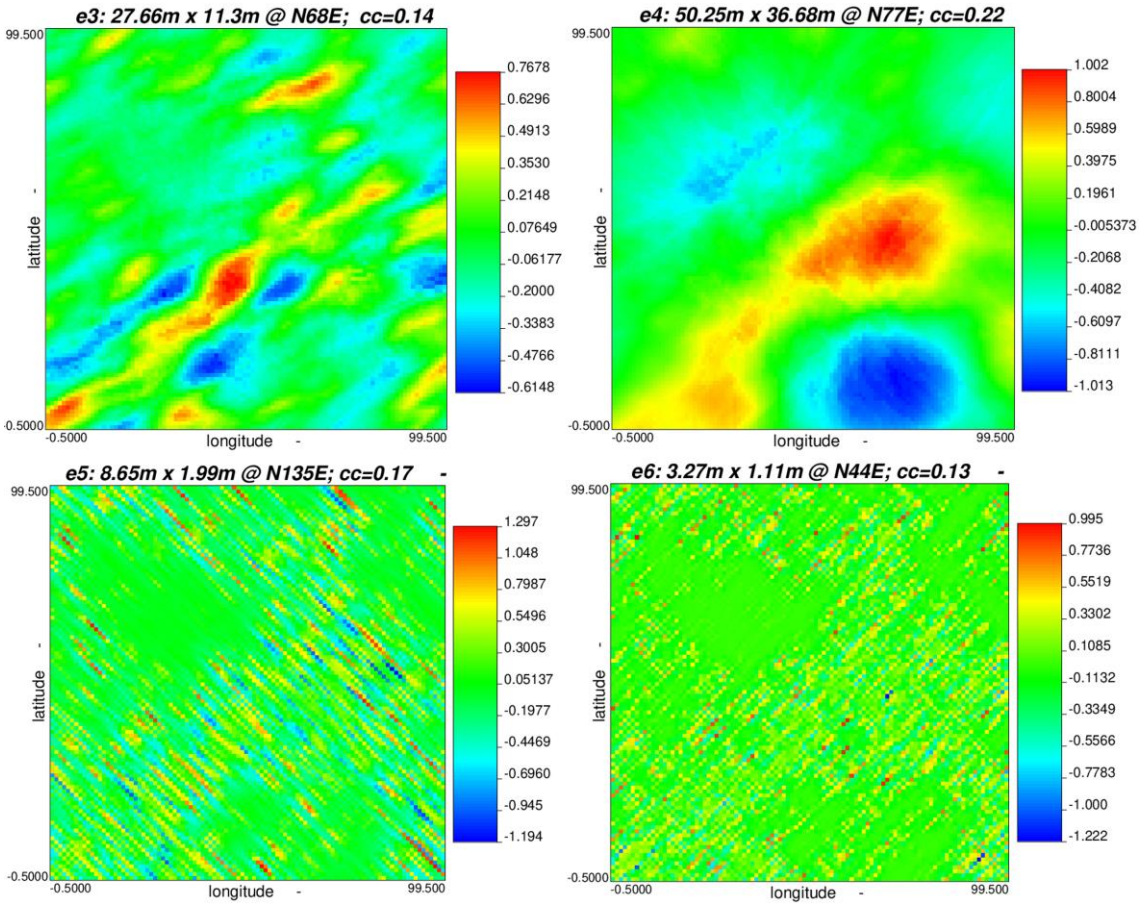


Figura 6-17: Os quatro fatores geomorfológicos obtidos com OFK a partir do variograma ajustado automaticamente com o método proposto.

A Figura 6-18 mostra, para comparação, na mesma escala de cores o mapa de entrada, a soma da krigagem da média com os fatores mostrados na Figura 6-16 (obtidos a partir do variograma *a priori*) e a respectiva soma dos fatores mostrados na Figura 6-17 (obtidos a partir do variograma ajustado pelo método proposto).

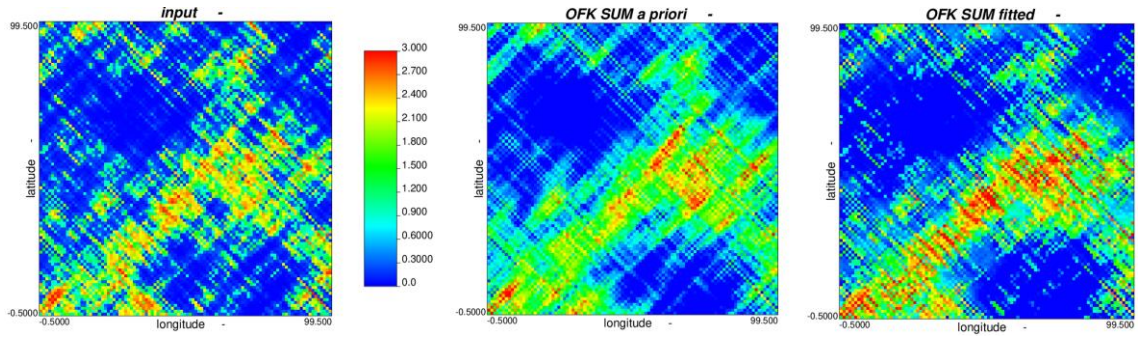


Figura 6-18: À esquerda, o mapa de entrada; no meio, a soma dos fatores obtidos com o variograma *a priori*; à direita, a soma dos fatores obtidos com o método. Todos os mapas estão na mesma escala de cores.

Embora a soma dos fatores geomorfológicos obtidos automaticamente não seja rigorosamente igual ao mapa de entrada, essa se mostrou superior em fidelidade do que a

soma dos fatores obtidos com o próprio variograma usado para sintetizar o mapa com SGSIM (variograma *a priori*). A Figura 6-19 mostra os *crossplots* entre os somatórios de fatores de ambos os casos contra o mapa de entrada. Constata-se, que o variograma ajustado pelo método explica melhor o dado de entrada porque o respectivo somatório de fatores tem menor dispersão e maior correlação em relação ao somatório obtido com o variograma *a priori* usado para sintetizar o dado de entrada com SGSIM. Os três agrupamentos vistos nos *crossplots* se devem às três modas do dado original (ver histograma da Figura 6-19).

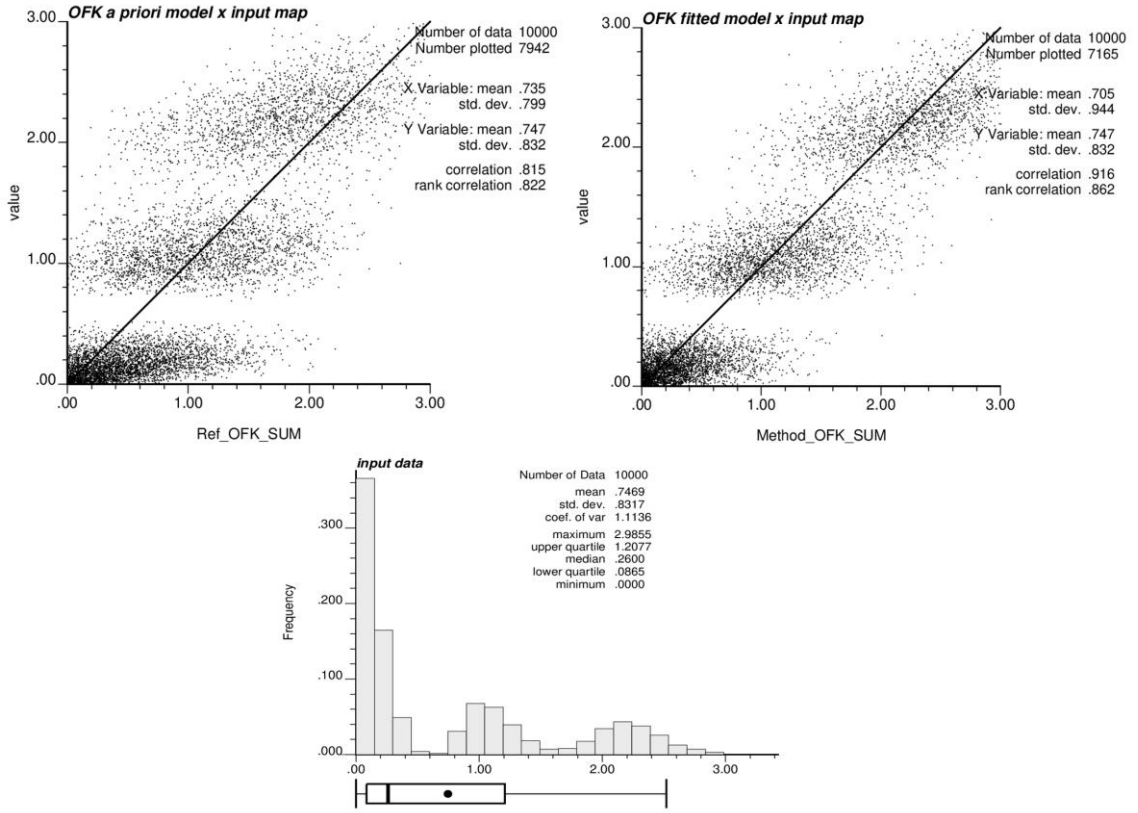


Figura 6-19: Em cima: *crossplots* entre os somatórios de fatores obtidos (eixos das abscissas): com o variograma *a priori* (esq.) e com o variograma ajustado automaticamente (dir.) contra o dado original (eixos das ordenadas). O histograma mostra as três modas do dado original.

Capítulo 7

Conclusão e trabalhos futuros

Como primeiro objetivo, foram avaliados os diversos algoritmos de decomposição estrutural manual do estado da arte em que alguns deles realizaram plenamente a tarefa colocada nesta tese, todavia requerendo esforço da parte do geomodelador. Como segundo objetivo, um método automatizado para decomposição estrutural foi proposto, foi implementado e mostrou-se eficaz e eficiente para realizar a tarefa colocada de forma automatizada. Como terceiro objetivo, os resultados do método proposto mostraram-se satisfatórios em comparação com os melhores resultados obtidos com os métodos manuais. Portanto, foi alcançada a meta colocada nesta tese, que consiste em propor um método para decomposição automatizada de uma imagem em fatores geomorfológicos de imagens com resultados equivalentes aos da krigagem fatorial.

7.1 Trabalhos futuros

Não obstante o recente progresso do aprendizado de máquina com redes neuronais e suas aplicações, seus algoritmos, aplicados com muito sucesso em problemas de classificação e de regressão, mostraram-se insatisfatórios na otimização para o problema de decomposição estrutural abordado nesta tese. Os algoritmos de aprendizado de máquina atuais baseados em redes neuronais se mostraram soluções genéricas que podem dar respostas quando não se conhece um algoritmo específico para determinado problema, mas que se mostraram ineficientes, requerendo muito tempo de processamento ou grande investimento em hardware. No entanto, uma ideia promissora a se investigar é a camada Gabor pura proposta nesta tese, atualmente não aplicável por conta do atual fundamento teórico baseado em gradientes dos pesos no processo de aprendizado de camadas convolutivas profundas.

Outra técnica de aprendizado de máquina – os algoritmos de agrupamento – revelou-se promissora na aplicação como parte do algoritmo especialmente projetado para a meta desta tese. O algoritmo proposto, baseado em algoritmo de agrupamento, mostrou-se superior tanto no que tange ao desempenho computacional quanto nos resultados quando comparado com uma abordagem baseada em aprendizado profundo. Outros algoritmos de

agrupamento que emergirem poderão ser investigados, testados e avaliados dentro do macroalgoritmo proposto.

O método proposto emprega, como método de otimização, o Algoritmo Genético devido à sua eficácia superior em relação aos demais métodos investigados dentro do escopo desta tese. Contudo, há muitos outros métodos de otimização que poderão ser investigados para emprego no fluxo proposto que podem apresentar resultados equivalentes ou resultar em melhora no desempenho computacional.

A função-objetivo proposta, baseada no Método Integral de Fourier, mostrou eficácia superior em relação à versão 2D da função-objetivo baseada no famoso programa VARFIT da GSLib, empregada com sucesso para ajustar modelos variográficos 1D. O Método Integral de Fourier mostrou ser uma ferramenta muito útil para aplicações em geoestatística, sobretudo quanto ao aspecto do desempenho computacional, e certamente merece maiores investigações para aplicações outras além do escopo desta tese.

Uma potencial aplicação prática futura é na inversão sísmica, um problema inverso que consiste em encontrar um volume de propriedades petrossísmicas (ex.: impedância acústica) a partir da resposta (levantamento sísmico) a um estímulo (ondas sísmicas, modeladas por uma ondaleta). No processo de inversão, normalmente se compara diretamente o volume sísmico obtido em campo (real) com o sintetizado a partir do modelo de propriedades de rocha de forma a minimizar a diferença para encontrar um modelo que explique os dados observados. Decompor ambos os volumes sísmicos, real e sintético, e comparar fator-a-fator tem potencial para melhorar o processo de inversão de sísmica.

Bibliografia

Allen, J. B. & Rabiner, L. R. 1977. A unified approach to short-time Fourier analysis and synthesis. *Proceedings of the IEEE*, v. 65, n. 11, pp. 1558-1564.

Carvalho, P. R. M.; Rasera, L. G.; Costa, J. F. C. L.; Araújo, M. G. S. & Varella, L. E. S. 2018. Variogram modeling of broadband artifacts of a seafloor map for filtering with Factorial Kriging. *Journal of Applied Geophysics*, v. 161, n. 02/2019, pp. 92-104. doi: 10.1016/j.jappgeo.2018.12.009

Cauchy, A. 1847. Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes Rendus Hebdomaires des Séances de l'Academie des Sciences*, n. 25, pp. 46-89.

Cerny, V. 1985. Thermodynamical approach top the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, v. 45, n. 1, pp. 41-51. doi: 10.1007/BF00940812

Chen, Y. & Feng, M. Q. 2003. A technique to improve the empirical mode decomposition in the Hilbert-Huang transform. *Earthquake Engineering and Egineering Vibration*, v. 2, n. 1, pp. 75-85. doi: 10.1007/BF02857540

Cheng, Y. 1995. Mean Shift, Mode Seeking, and Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 17, n. 8: 790–799. doi:10.1109/34.400568

Chollet, F. & outros. 2015. *Keras*. https://keras.io (acessado em 17/11/2019).

Chopra, S. & Marfurt, K. J. 2007. *Seismic Attributes for Prospect Identification and Reservoir Characterization*. Society of Exploration Geophysicists, Tulsa, Oklahoma, EE.UU. v. 11. (SEG Geophysical Developments).

Chui, C. K. 1992. *An introduction to Wavelets*. Academic Press, São Diego, Califórnia, EE.UU. 266p.

Cooley, J. W. & Tuckey, J. W. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, v. 19, n. 90, pp. 297-301. doi: 10.1090/S0025-5718-1965-0178586-1

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. & Stein, C. 2009. *Introduction to algorithms 3rd edition*. The MIT Press, EE.UU. 1292p.

Daubechies, I. 1990. The wavelet transform, time-frequency localization and signal analysis. IEEE Transactions on Information Theory, v. 36, n. 5, pp. 961-1005. doi: 10.1109/18.57199

Deutsch, C. V. 1992. *Annealing techniques applied to reservoir modeling and the integration of geological and engineering (well test) data*. PhD. Tese, Universidade de Stanford, Palo Alto, EE.UU.

Deutsch, C. V. & Journel, A. G. 1998. *GSLIB: Geostatistical Software Library and User's Guide*. Oxford University Press, Nova Iorque. 369p.

Ester, M., Kriegel, H., Sander, J. & Xu, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. in: Simoudis, E., Han, J. & Fayyad, U. M. (eds.). *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 226–231.

Feichtinger, H. G. & Strohmer, T. 1998. *Gabor Analysis and Algorithms: Theory and Applications*. Birkhäuser Basel, Nova Iorque. 496p.

Forgy, E. W. 1965. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*. v. 21, n. 3, pp. 768–769.

Fornberg, B. & Piret, C. 2007. A stable algorithm for flat radial basis functions on a sphere. *SIAM Journal of Scientific Computing*, v. 30, n. 1, pp. 60-80. doi:10.1137/060671991

Forsythe, G. E. & Moler, C. B. 1968. Computer Solution of Linear Algebraic Systems. *SIAM Review*, v. 10, n. 3, pp. 384-385. doi: 10.1137/1010075

Fourier, J.-B. J. 1822. *Théorie analytique de la chaleur*. F. Didot, Paris.

Fraser, A. & Burnell, D. 1970. *Computer Models in Genetics*. McGraw-Hill, Nova Iorque. 192p.

Frey, B. J. & Dueck, D. 2007. Clustering by passing messages between data points. *Science*, v. 315 n. 5814, pp. 972–976. doi: 10.1126/science.1136800

Fukushima, K. 1980. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*. v. 36, n. 4, pp. 193–202. doi:10.1007/BF00344251

Gabor, D. 1946. Theory of Communication. Part 1: The analysis of information. *Journal of the Institution of Electrical Enginners*, v. 93, n. 26, pp. 429-441. doi: 10.1049/ji-3-2.1946.0074

Galli, A. & Sandjivy, L. 1985. Analyse Krigeante et Analyse Spectrale. Sciences de la Terre, *Informatique Géologique*, n. 21, pp. 115-124.

Goovaerts, P. & Sonnet, P. 1993. Study of Spatial and Temporal Variations of Hydrogeochemical Variables Using Factorial Kriging Analysis. in: Soares, A (ed.). *Geostatistics Tróia '92*, 2. Springer, Holanda, pp. 745-756. doi: 10.1007/978-94-011-1739-5_59

Goovaerts, P. 1997. *Geostatistics for Natural Resources Evaluation*. Oxford University Press, New York. 483p.

Graps, A. 1995. An introduction to wavelets. *IEEE Computational Science and Egineering*, v. 2, n. 2, pp. 50-61. doi: 10.1109/99.388960

Green, D. G. 1993. Emergent behavior in biological systems. in: Green, D. G. & Bossomaier, T. J. (eds.). *Complex Systems: From Biology to Computation*. IOS Press, pp. 24-35.

Grosan, C. & Abraham, A. 2009. A novel global optimization technique for high dimensional functions. *International Journal of Intelligent Systems*, v. 24, n. 4, pp. 421-440. doi: 10.1002/int.20343

Haar, A. 1910. Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, v. 69, n. 3, pp. 331-371. doi: 10.1007/BF01456326

Higham, N. J. 1986. Computing the polar decomposition with applications. *Journal of Scientific and Statistical Computing*, v. 7, n. 4, pp. 1160-1174. doi: 10.1137/0907079

Hinton, G. E. & Salakhutdinov, R. R. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science*, v. 313, n. 5786, pp. 504–507. doi:10.1126/science.1127647.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, EE.UU. 183p.

Huang, N. E.; Long, S. R. & Shen, Z. 1996. The Mechanism for Frequency Downshift in Nonlinear Wave Evolution. *Advances in Applied Mechanics*, v. 32, pp. 59-117.

Hubel, D. H.; Wiesel, T. N. 1968. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*. v. 195, n. 1, pp. 215–243. doi:10.1113/jphysiol.1968.sp008455

Hunt, B. 1971. A matrix theory proof of the discrete convolution theorem. *IEEE Transactions on Audio and Electroacoustics*, v. 19, n. 4, pp. 285-288. doi: 10.1109/TAU.1971.1162202

Isaaks, E. H. & Srivastava, R. M. 1989. *An Introduction to Applied Geostatistics*. Oxford University Press, Nova Iorque. 561p.

Jaquet, O. 1989. Factorial Kriging Analysis Applied to Geological Data from Petroleum Exploration. *Mathematical Geology*, v. 21, n.7, pp. 683-691. doi:10.1007/BF00893316

Johnson, S. G. & Frigo, M. 2008. Implementing ffts in practice. *Fast Fourier Transforms*. Rice University, Houston, Texas, EE.UU.

Kalogirou, S. A. 2014. *Solar Energy Engineering: Processes and Systems (second edition)*. Academic Press, Amsterdã. 840p. doi:10.1016/C2011-0-07038-2

Kennedy, J. & Eberhart, R. 1995. Particle Swarm Optimization. in: IEEE Neural Networks Council (ed.). *Proceedings of IEEE International Conference on Neural Networks IV*. IEEE, EE.UU., pp. 1942–1948. doi:10.1109/ICNN.1995.488968.

Knuth, D. E. 1998. *The Art of Computer Programming, Volume 3: Sorting and Searching, Second Edition*. Addison-Wesley, Massachusetts, EE.UU. 780p.

Lally, A. & Fodor, P. 2011. Natural Language Processing With Prolog in the IBM Watson System. Association for Logic Programming.

Laptev, D.; Savinov, N.; Buhmann, J. M. & Pollefeys, M. 2016. TI-Pooling: transformation-invariant pooling for feature learning in convolutional neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 289–297. arXiv:1604.06318

Larrondo, P.; Neufeld, C. T. & Deutsch, C. 2003. VARFIT: a program for semi-automatic variogram modelling. *Center for Computational Geostatistics Report Five*. Universidade de Alberta, Edmonton, Alberta, Canadá.

Linderhed, A. 2009. Image Empirical Mode Decomposition: A new tool for image processing, *Advances in Adaptive Data Analysis*, v. 1, n. 2, pp. 265-294. doi: 10.1142/S1793536909000138

Long, J.; Shelhamer, E. & Darrell, T. 2014. *Fully convolutional networks for semantic segmentation*. arXiv:1411.4038

Luan, S.; Chen, C.; Zhang, B.; Han, J. & Liu, J. 2018. Gabor Convolutional Networks. *IEEE Transactions on Image Processing*, v. 27, n. 9, pp. 4357-4366. doi: 10.1109/TIP.2018.2835143

Mallat, S. G. 1989. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 11, n. 7, pp. 674-693. doi: 10.1109/34.192463

Mallet, J-L. 1989. Discrete Smooth Interpolation. *Journal of ACM Transactions on Graphics*, v. 8, n. 2, pp. 121-144. doi: 10.1145/62054.62057

Marĉelja, S. 1980. Mathematical description of the responses of simple cortical cells. Journal of the Optical Society of America, v. 70, n. 11, pp. 1297-1300. doi: 10.1364/JOSA.70.001297

Marcotte, D. 1996. Fast variogram computing with FFT. *Computers & Geosciences*, v. 62, n. 10, 1175-1186. doi: 10.1016/S0098-3004(96)00026-X

Matheron, G. 1963. Principles of Geostatistics. *Economic Geology*, v. 58, n. 8, pp. 1246-1266. doi: 10.2113/gsecongeo.58.8.1246

Matheron, G. 1982. Pour une analyse krigeante de données régionalisées. *Technical Report*, N-732. Centre de Géostatistique, Fontainebleau, France.

Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N. & Teller A. H. 1953. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, v. 21, n. 6, pp. 1087-1092. doi: 10.1063/1.1699114

Meyer, Y. 1995. *Wavelets and Operators*. Volume 1. Cambridge University Press. 244p.

Morettin, P. A. 1999. *Ondas e Ondaletas: da Análise de Fourier à Análise de Ondaletas*. Editora da Universidade de São Paulo. 320p.

Neufeld, C. & Wilde, B. 2005. A Global Kriging Program for Artifact-Free Maps. *CCG Annual Report 7*, pp. 403- 1-8.

Ng, A. Y., Jordan, M. I. & Weiss, Y. 2001. On spectral clustering: analysis and an algorithm. *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, pp. 849–856.

Pardo-Iguzquiza, E. & Chica-Olmo, M. 1993. The Fourier Integral Method: An Efficient Spectral Method for Simulation of Random Fields. *Mathematical Geology*, v. 25, n. 2, pp. 177-217. doi: 10.1007/BF00893272

Press, W. H.; Teukolsky, S. A.; Vatterling, W. T. & Flannery, B. P. 1992. *Numerical Recipes in C: The Art of Scientific Computing (2ª edição)*. Cambridge University Press. 994p.

Press, W. H.; Teukolsky, S. A.; Vatterling, W. T. & Flannery, B. P. 2007. *Numerical Recipes: The Art of Scientific Computing (3ª edição)*. Cambridge University Press. 1235p.

Ramachandran, P.; Zoph, B. & Le, Q. V. 2017. *Searching for Activation Functions*. arXiv:1710.05941

Rasera, L. G. 2014. *Geoestatística de múltiplos pontos aplicada à simulação de modelos geológicos em grids estratigráficos*. MSc. Dissertação, Universidade Federal do Rio Grande do Sul, Porto Alegre.

Richter, H. 1952. Zur Elastizitätstheorie endlicher Verformungen. *Mathematische Nachrichten*, n. 8, pp. 65-73. doi: 10.1002/mana.19520080109

Ronneberger, O.; Fischer, P. & Brox, T. 2015. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv:1505.04597

Rosa, A. L. R. 2010. *Análise do Sinal Sísmico*. Sociedade Brasileira de Geofísica, Rio de Janeiro. 668p.

Rosenblatt, F. 1957. *The Perceptron--a perceiving and recognizing automaton*. Report 85-460-1, Cornell Aeronautical Laboratory.

Rumelhart, D. E.; Hinton, G. E. & Williams, R. J. 1986. Learning Internal Representations by Error Propagation. in: Rumelhart, D. E.; McClelland, J. L. & PDP research group. (eds.). *Parallel distributed processing: Explorations in the microstructure of cognition*, volume 1. Foundation MIT Press, 1986.

Sandjivy, L. 1984. The factorial kriging analysis of regionalized data. Its application to geochemical prospecting. in: Verly, G., David, M., Journel, A. G. e Marechal, A. (eds.). *Geostatistics for Natural Resources Characterization*, parte 1. Springer, Holanda, pp. 559-571. doi: 10.1007/978-94-009-3699-7_32

Sarwar, S. S.; Panda, P. & Roy, K. 2017. Gabor filter assisted energy efficient fast learning Convolutional Neural Networks. in: IEEE. 2017. *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. doi: 10.1109/ISLPED.2017.8009202

Shamsipour, P.; Chouteau, M. & Marcotte, D. 2017. Data analysis of potential field methods using geostatistics. *Geophysics*, v. 82, n. 2, pp. G35-G44. doi: 10.1190/geo2015-0631.1

Shepard, D. 1968. A two-dimensional interpolation function for irregularly-spaced data. in: Stone, M. C. *Proceedings of the 1968 ACM National Conference.* ACM, Nova Iorque, pp. 517–524. doi:10.1145/800186.810616

Tanembaum, A. S. 2003. *Redes de Computadores (trad. 4ª ed.)*. Ed. Campus, São Paulo. 945p.

Telford, W. M.; Geldart, L. P. & Sheriff, R. E. 1990. *Applied Geophysics* (2ª ed.). Cambridge University Press, Melbourne, Austrália.

Theis, L.; Shi, W.; Cunningham, A. & Huszár, F. 2017. Lossy image compression with compressive autoencoders. ICLR 2017. arXiv:1703.00395.

Turing, A. M. 1950. Computing machinery and intelligence. *Mind.* LIX 238, pp. 433-460. doi:10.1093/mind/LIX.236.433

van Dyk, D. A. & Meng, X. 2001. The Art of Data Augmentation. *Journal of Computational and Graphical Statistics*, v.10, n.1, pp. 1-50. doi: 10.1198/10618600152418584

Wackernagel, H. 2003. *Multivariate Geostatistics*. Springer, Berlin, Germany. 388p.

Ward, J. H., Jr. 1963. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, n. 58, pp. 236–244.

Wen, R. & Sinding-Larsen, R. 1997. Image Filtering by Factorial Kriging – Sensitivity Analysis and Application to GLORIA Side-Scan Sonar Images. *Mathematical Geology*, v. 29, no.4, pp. 433-468. doi: 10.1007/BF02775083

Xu, W. 1996. Conditional Curvilinear Stochastic Simulation Using Pixel-based Algorithms. *Mathematical Geology*, v. 28, n. 7, pp. 937-949. doi: 10.1007/BF02066010

Yuan, Y. 1999. Step-Sizes for the Gradient Method. *Studies in Advanced Mathematics*, v. 42, n. 2, pp. 1-13.

Zhang, W., Wang, X., Zhao, D. and Tang, X. 2012. Graph Degree Linkage: Agglomerative Clustering on a Directed Graph. in: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y. and Schmid, C. (eds.). *Computer Vision – ECCV 2012: Lecture Notes in Computer Science*, pp. 428–441. doi:10.1007/978-3-642-33718-5_31

Anexo 1: Script 1

Script de calculadora do software GammaRay para separar altas e baixas frequências espaciais em um espectro de amplitudes obtido com FFT.

```
var dx:= X_ - 130;
var dy:= Y_ - 150;
var radius:= sqrt(dx*dx+dy*dy);
if( radius < 10 )
{
   mag_low := Magnitude;
   mag_high := 0.0;
}
else
{
   mag_low := 0.0;
   mag_high := Magnitude;
}
```

Anexo 2: arquivo para SVD com combinação linear global

Arquivo de pesos para o botão “Custom Analysis” do diálogo “SVD Analysis” do software GammaRay.

```
0.25	0.25	0.25	0.25
0.25	0.25	0.25	0.25
0.25	0.25	0.25	0.25
0	0.33	0.33	0.34
0	0.33	0.33	0.34
0	0.33	0.33	0.34
0	0.33	0.33	0.34
0	0.33	0.33	0.34
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0.5	0.5
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
```

0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1

Anexo 3: Script 2

Script de calculadora do software GammaRay para calcular um núcleo de Gabor em uma malha regular.

```
//---------the Gabor kernel parameters--------
var azimuth := 155.0;
var lambda; lambda := 20.0;
var mx; mx := 50.0;
var my; my := 50.0;
var sigmax; sigmax := 20.0;
var sigmay; sigmay := 20.0;
//-----------------------------------------
var theta; theta := (90.0-azimuth) * 3.1416/180;
var x; x := (X_-mx) * cos(theta) + (Y_-my) * sin(theta);
var y; y := -(X_-mx) * sin(theta) + (Y_-my) * cos(theta);
var g;
g := exp( -( x*x / (2*sigmax*sigmax) + y*y / (2*sigmay*sigmay) ) );
g := g * cos( 2 * 3.1416 * x / lambda );
gabor := g;
```

Anexo 4: Pseudocódigos dos métodos de otimização

i) SA:

```
function temperature (int stepNumber) {
    return f_Tinitial * std::exp( -stepNumber / (double)1000 *
                          (1.5 * std::log10( Tinitial ) ) );
};
function probAcceptance( double eCurrent, double eNewLocal, double T ) {
   if ( eNewLocal > eCurrent )
      return ( T - Tfinal ) / ( Tinitial - Tfinal );
   else
      return 1.0 - ( T - Tfinal ) / ( Tinitial - Tfinal );
};
double eNew = std::numeric_limits<double>::max();
double lowestEnergyFound = std::numeric_limits<double>::max();
array ParamsLowestEnergyFound;
for(int k = 0; k < i_kmax; ++k ){
    double T = temperature( k );
    if( T < Tfinal )
       break;
    array ParamsNew = ParamsCurrent;
    for( int i = 0; i < i_nPar; ++i ){
       double Tmp = 0.0;
       while( true ){
          delta = ParamsMax[i] - ParamsMin[i];
          double LO = ParamsCurrent[i] - (factorSearch * delta);
          double HI = ParamsCurrent[i] + (factorSearch * delta);
          Tmp = LO + std::rand() / (RAND_MAX/(HI-LO)) ;
          if ( Tmp >= ParamsMin[i] && Tmp <= ParamsMax[i] )
              break;
       }
       ParamsNew[i] = Tmp;
    }
    double eCurrent = objectiveFunction( ParamsCurrent );
    eNew = objectiveFunction( ParamsNew );
    double probMov = probAcceptance( eCurrent, eNew, T );
    if( probMov >= ( (double)std::rand() / RAND_MAX ) ) {
        ParamsCurrent = ParamsNew;
        if( eNew < lowestEnergyFound ){
             lowestEnergyFound = eNew;
             ParamsLowestEnergyFound = ParamsCurrent;
        }
    }
}
```

ii) GD:

```
array params = ParamsLowestEnergyFound;
for(int iOptStep = 0 ; iOptStep < maxNumberOfOptimizationSteps; ++iOptStep ){

    array gradient( params.size() );

    for(int iParameter = 0; iParameter < parameterCount; ++iParameter ){
        array paramsRight(params);
        paramsRight[iParameter] = paramsRight(iParameter) + epsilon;
        array paramsLeft(params);
        paramsLeft[iParameter]  = paramsLeft(iParameter)  - epsilon;
        gradient[iParameter] = ( objectiveFunction( paramsRight )
                                      -
                                 objectiveFunction( paramsLeft ))
                                      /
                                   ( 2 * epsilon );
    }
```

```
    double currentF = std::numeric_limits<double>::max();
    double nextF = 1.0;
    double alpha = initialAlpha;
    for(int iAlphaReductionStep = 0; iAlphaReductionStep <
           maxNumberOfAlphaReductionSteps; ++iAlphaReductionStep ){
        array new_params = params - gradient * alpha;
        for( int i = 0; i < new_params.size(); ++i){
            if( new_params[i] < 0.0 )
                new_params[i] = 0.0;
            if( new_params[i] > 1.0 )
                new_params[i] = 1.0;
        }
        currentF = objectiveFunction( params );
        nextF =    objectiveFunction( new_params );
        if( nextF < currentF ){
            params = new_params;
            break;
        }
        alpha = alpha / 2.0;
    }

    double ratio = currentF / nextF;
    if( ratio  < (1.0 + convergenceCriterion) )
        break;


}
```

iii) LSRS:

```
min[] := dominio_min()
max[] := dominio_max()

para r := 1 até numero_de_reinicios

   //inicializar o conjunto de pontos de partida
   para j := 1 até numero_de_pontos_de_partida
      para i := 1 até numero_de_variaveis
         vx[j][i] := min[i] + random() * ( max[i] - min[i] )
      fim para
   fim para

   //LSRS
   para t := 1 até numero_de_iteracoes
      para j := 1 até numero_de_pontos_de_partida
         para i := 1 até numero_de_variaveis
            p := random() (ou -1 conforme sugerido pelo autor)
            a := 2 + 3 / ( 2 ^ (t^2 + 1) )
            x[i] := vx[j][i] + p * a;
         fim para
         se F(x[]) < F(vx[j][]) entao
            vx[j][] := x[]
            xOtimo[] := x[]
         fim se
      fim para
   fim para

   epsilon := 0.0001

   //reduzir o dominio para o reinício
   para i := 1 até numero_de_variaveis
      x_dir[i] := xOtimo[i] + epsilon
      x_esq[i] := xOtimo[i] - epsilon
      dx := ( F(x_dir[i]) - F(x_esq[i]) ) /  ( 2 * epsilon ) //derivada parcial em x[i]
      se dx > 0 então max[i] := xOtimo[i]
      se dx < 0 então min[i] := xOtimo[i]
   fim para
fim para
```

iv) PSO:

```
min[] := dominio_min()
```

```
max[] := dominio_max()

//inicializar o conjunto de partículas (posições e velocidades)
para j := 1 até numero_de_particulas
   para i := 1 até numero_de_variaveis
      x[j][i] := min[i] + random() * ( max[i] - min[i] )
      v[j][i] := 0.0
   fim para
   xMin[j][] := x[j][]
fim para

//inicializar a melhor posição global
para j := 1 até numero_de_particulas
   se F(xMin[j]) < F(gxMin)
      gxMin[] := xMin[j][]
   fim se
fim para

para t := 1 até numero_de_passos
   para j := 1 até numero_de_particulas
      x_cadidata[], v_candidata[] := aplicar_Equação 2-4( x[j][], v[j][] )
      se F( x_candidata[] ) < F( x[j][] )
         x[j][] := x_candidata[]
         v[j][] := v_candidata[]
      fim se
      se F( x_candidata[] ) < F( xMin[j][] )
         xMin[j][] := x_candidata[]
      fim se
      se F( x_candidata[] ) < F( gxMin[] )
         gxMin[] := x_candidata[]
      fim se
   fim para
fim para
```

v) GA:

```
std::vector< Individual > population;
for( int iGen = 0; iGen < maxNumberOfGenerations; ++iGen ){

    //inicializar ou completar a população com
    //indivíduos gerados aleatoriamente
    while( population.size() < nPopulationSize ){
        spectral::array pw( (spectral::index)totalNumberOfParameters );
        for( int i = 0; i < pw.size(); ++i ){
            double LO = ParamsMin[i];
            double HI = ParamsMax[i];
            pw[i] = LO + std::rand() / (RAND_MAX/(HI-LO));
        }
        Individual ind( pw );
        population.push_back( ind );
    }

    //avaliar a função-objetivo em cada indivíduo
    for( int iInd = 0; iInd <= nPopulationSize; ++iInd ){
        Individual& ind = population[iInd];
        ind.fValue = objectiveFunction( ind.parameters );
    }

    //ordenar a população por ordem crescente por individuo.fValue
    //ou seja, os melhor avaliados ficam em primeiro na lista
    std::sort( population.begin(), population.end() );

    //corta a população para o tamanho-alvo, descartando os piores
    while( population.size() > nPopulationSize )
        population.pop_back();

    //realiza a seleção pelo método do torneio binário (ver texto)
    std::vector< Individual > selection;
    for( uint iSel = 0; iSel < nSelectionSize; ++iSel ){

        std::vector< Individual > tournament;
        {
```

```
            //sorteia dois indivíduos diferentes para o torneio
            int tournCandidate1 = std::rand() / (double)RAND_MAX * (
                                                population.size() - 1 );
            int tournCandidate2 = tournCandidate1;
            while( tournCandidate2 == tournCandidate1 )
                tournCandidate2 = std::rand() / (double)RAND_MAX * (
                                                population.size() - 1 );
            //adiciona os indivíduos selecionados ao torneio
            tournament.push_back( population[tournCandidate1] );
            tournament.push_back( population[tournCandidate2] );
            //ordena os indivíduos na lista do torneio por ordem crescent
            //de individuo.fValue
            std::sort( tournament.begin(), tournament.end());
        }
        //adiciona o vencedor do torneio para o pool de seleção
        selection.push_back( tournament.front() );
    }

    //realiza cruzamento e mutação nos indivídios selecionados
    std::vector< Individual > nextGen;
    while( ! selection.empty() ){
        //sorteia dois indivíduos para cruzar.
        int parentIndex1 = std::rand() / (double)RAND_MAX *
                                            ( selection.size() - 1 );
        int parentIndex2 = parentIndex1;
        while( parentIndex2 == parentIndex1 )
            parentIndex2 = std::rand() / (double)RAND_MAX *
                                            ( selection.size() - 1 );
        Individual parent1 = selection[ parentIndex1 ];
        Individual parent2 = selection[ parentIndex2 ];
        //remove os pais do pool de seleção
        selection.erase( selection.begin() + parentIndex1 );
        selection.erase( selection.begin() + parentIndex2 );
        //sorteia um valor entre 0.0 e 1.0 de uma distribuição uniforme
        double p = std::rand() / (double)RAND_MAX;
        //se deve fazer o cruzamento...
        if( p < probabilityOfCrossOver ){
            //faz o cruzamento, gerando dois descendentes
            std::pair< Individual, Individual> offspring =
                        parent1.crossOver( parent2, pointOfCrossover );
            Individual child1 = offspring.first;
            Individual child2 = offspring.second;
            //faz a mutação tanto nos pais quanto nos filhos (sujeito
            //a uma probabilidade)
            child1.mutate( mutationRate, ParamsMin, ParamsMax );
            child2.mutate( mutationRate, ParamsMin, ParamsMax );
            parent1.mutate( mutationRate, ParamsMin, ParamsMax );
            parent2.mutate( mutationRate, ParamsMin, ParamsMax );
            //adiciona todos ao pool para a próxima geração
            nextGen.push_back( child1 );
            nextGen.push_back( child2 );
            nextGen.push_back( parent1 );
            nextGen.push_back( parent2 );
        } else { //não houve cruzamento
            //apenas faz a mutação dos pais que não cruzaram e os insere
            //no pool da próxima geração
            parent1.mutate( mutationRate, ParamsMin, ParamsMax );
            parent2.mutate( mutationRate, ParamsMin, ParamsMax );
            nextGen.push_back( parent1 );
            nextGen.push_back( parent2 );
        } // if do cruzamento e da mutação
    } // loop do cruzamento e da mutação

    //faz a próxima geração
    population = nextGen;

} //loop da geração
```