

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

Reconstrução Fractal de Sinais^{*}

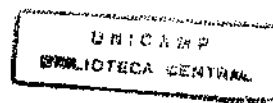
Autor: RICARDO CAETANO AZEVEDO BILOTI

Orientador: PROF. DR. LÚCIO TUNES DOS SANTOS

Dissertação apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas, como parte dos pré-requisitos para obtenção do Título de Mestre em Matemática Aplicada.

Março de 1998

^{*} Pesquisa financiada pela FAPESP, proc. 95/09080-7.



UNIDADE	BC
N.º CHAMADA:	
	33734
	395198
PREÇO	R\$ 11,00
DATA	08/05/98
N.º CPU	

CM-00110689-7

**FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DO IMECC DA UNICAMP**

Biloti, Ricardo Caetano Azevedo

B497r Reconstrução fractal de sinais / Ricardo Caetano Azevedo Biloti
-- Campinas, [S.P. :s.n.], 1998.

Orientador : Lúcio Tunes dos Santos

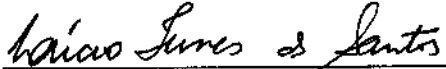
Dissertação (mestrado) - Universidade Estadual de Campinas,
Instituto de Matemática, Estatística e Computação Científica.

1. Fractais. 2. Teoria da aproximação. 3. Processamento de
sinais. 4. Interpolação. 5. Meteorologia. I. Santos, Lúcio Tunes dos.
II. Universidade Estadual de Campinas. Instituto de Matemática,
Estatística e Computação Científica. III. Título.

RECONSTRUÇÃO FRACTAL DE SINAIS

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Ricardo Caetano Azevedo Biloti e aprovada pela comissão julgadora.

Campinas, 16 de março de 1998.



Prof. Dr. Lúcio Tunes dos Santos
Orientador

Dissertação apresentada ao Instituto de Matemática, Estatística e Computação Científica, Unicamp, como requisito parcial para obtenção do Título de MESTE em Matemática Aplicada.

Dissertação de Mestrado defendida e aprovada em 16 de março de 1998

pela Banca Examinadora composta pelos Profs. Drs.

Lúcio Tunes dos Santos

Prof (a). Dr (a). LÚCIO TUNES DOS SANTOS

José Mário Martínez Pérez

Prof (a). Dr (a). JOSÉ MÁRIO MARTÍNEZ PÉREZ

José Luiz Boldrini

Prof (a). Dr (a). JOSÉ LUIZ BOLDRINI

Agradecimentos

Agradeço, em primeiro lugar, aos meus pais, Arnaldo Azevedo Biloti e Marcia Caetano. Certamente não teria chegado a este ponto sem a participação deles. Sempre me deram o suporte necessário, em todos os sentidos, para que eu pudesse crescer. Aos meus pais devo toda a confiança que depositaram em mim. Sem dúvida, uma das forças que me impulsionaram até aqui foi a vontade de corresponder às expectativas deles.

À minha irmã Fabiane, por toda a alegria que dá e por sempre ter confiado em mim.

À minha mulher Débora, por estar a meu lado, apoiando e motivando-me em tudo.

Ao meu orientador, Prof. Dr. Lúcio Tunes dos Santos, pela oportunidade que me deu. Por ter tido a coragem de apostar em mim quando eu ainda estava apenas ingressando na universidade. Agradeço por ter me acompanhado durante todo esse tempo, me orientando, no sentido mais amplo da palavra. Agradeço pela amizade, que sem dúvida tornou o trabalho mais agradável.

Agradeço aos meus amigos, em particular, Marcello e Wellington, que para mim são como irmãos, e Cantão e Portugal pela amizade e pelas incontáveis horas de discussão e troca de idéias.

À Fátima e Luciana, pelo empenho em ajudar a vencer todas as barreiras burocráticas que surgem no decorrer de uma pesquisa.

À FAPESP pelo suporte financeiro, tanto na graduação como no mestrado.

Ao IMECC e a Unicamp, pela estrutura e pelo ambiente onde me formei.

Índice

Agradecimentos	ii
Índice	iii
Índice de Figuras	v
I Introdução	1
II Interpolação Fractal	4
II.1 Sistema iterativo de funções	4
II.2 Interpolação fractal	9
III Dimensão Fractal	15
III.1 O algoritmo <i>Box Counting</i>	16
III.2 Dimensão para funções de interpolação fractal	17
IV Aproximação Fractal de Funções	19
IV.1 Construindo o gráfico de uma FIF	20
IV.2 Estimando os fatores de escala vertical	21
IV.3 Estimando os coeficientes de orientação	22
IV.4 Escolhendo os pontos de corte	24
IV.5 Exemplos reais trabalhados	27
V Reconstrução Fractal de Funções	31
V.1 Escolhendo os pontos de corte	32

<i>ÍNDICE</i>	iv
V.2 Estimando a dimensão fractal	32
V.3 Estimando os parâmetros do intervalo obscuro	40
V.4 Exemplos trabalhados	40
Difratograma sintético de raios X (situação ideal)	40
Difratograma sintético de raios X	47
Direção do vento	53
Velocidade vertical do vento	59
VI Conclusão	66
A Demonstração dos Resultados	69
Demonstrações do Capítulo II	69
Demonstrações do Capítulo III	78
B Listagem das rotinas	80
Bibliografia	123

Índice de Figuras

II.1	Curva de Koch.	6
II.2	Conjunto \mathcal{L} e união das imagens \mathcal{L}_i , $i = 1, 2, 3, 4$	7
II.3	Folha de Samambaia.	8
II.4	Interpretação geométrica dos fatores de escala vertical. Aqui, $s_1 = h_1/h$, onde denotamos $h = h_f[x_0, x_N](x_g)$ e $h_1 = h_f[x_0, x_1](x_l)$	11
II.5	Função de interpolação fractal para os pontos de Λ , marcados com asterisco no gráfico.	13
IV.1	Teste de recuperação dos fatores de escala vertical. (a) gráfico original da FIF encontrada em [6]. (b) gráfico construído com os fatores de escala vertical estimados e os pontos de interpolação reais.	22
IV.2	Interpretação geométrica do coeficiente de orientação. No intervalo $[0.2, 0.4]$, $o_n = 1$ e no intervalo $[0.5, 0.7]$, $o_n = -1$	23
IV.3	Teste do algoritmo de seleção dos pontos para o caso ideal. (a) Difratograma sintético de raios X. (b) Gráfico construído com os pontos de corte, fatores de escala vertical e coeficientes de orientação estimados.	27
IV.4	Direção do vento, Reserva Ducke, Floresta Amazônica - (a) gráfico original com 1156 pontos e (b) gráfico aproximado com 30 pontos e módulos máximos dos fatores de escala vertical limitados por 0.3 até o ponto 0.360, 0.15 até o ponto 0.566 e 0.3 até o ponto 1.156.	28
IV.5	Velocidade vertical do vento, Reserva Ducke, Floresta Amazônica - (a) gráfico original com 1156 pontos e (b) gráfico aproximado com 44 pontos e limitação no módulo dos fatores de escala vertical de 0.2.	29

V.1	Exemplo de malhas geradas pelo algoritmo de <i>Box Counting</i> para estimar a dimensão fractal.	34
V.2	Exemplo de estimativa gerada pelo algoritmo <i>Box Counting</i>	35
V.3	Comportamento do estimador da dimensão para vários valores de n_i . Para a curva sólida, $n_i = 1$ e $n_f = 9$, com $D = 1.2372$; para a curva tracejada, $n_i = 2$ e $n_f = 9$, com $D = 1.3387$ e, para a curva pontilhada, $n_i = 3$ e $n_f = 9$, com $D = 1.4263$	36
V.4	Comportamento do módulo do fator de escala vertical do subintervalo ausente em função da dimensão fractal do gráfico.	38
V.5	Comportamento do módulo do fator de escala vertical do subintervalo ausente em função da dimensão fractal do gráfico, apenas para o intervalo aceitável da dimensão.	39
V.6	Difratograma sintético de raios X. Em destaque, um subintervalo onde o gráfico é auto-similar e o qual tentaremos reconstruir.	41
V.7	(a) gráfico reconstruído usando $ s_6 = 0.5246$. (b) gráfico reconstruído usando $ s_6 = 0.7351$. Em (a) e (b) a curva pontilhada é o gráfico real. . . .	43
V.8	Subintervalo reconstruído. Gráfico real (pontilhado) e reconstruído (traço sólido) com os parâmetros do intervalo ausente $s_6 = -0.219$ e $\sigma_6 = 1$	44
V.9	(a) difratograma sintético de raios X. (b) gráfico reconstruído.	45
V.10	Comparação da aproximação conseguida para o gráfico, no subintervalo ausente, através de <i>splines</i> cúbicas (curva tracejada) e da reconstrução fractal suavizada (traço sólido), com o gráfico real suavizado (curva pontilhada). . .	46
V.11	Difratograma sintético de raios X. Em destaque, a porção a ser reconstruída.	47
V.12	Trecho reconstruído. O traço sólido representa a aproximação, enquanto que a curva pontilhada, o gráfico real. Em (a) $s_{13} = 0.0647$, em (b) $s_{13} = 0.1700$, em (c) $s_{13} = 0.2408$ e em (d) $s_{13} = 0.9448$	50
V.13	(a) difratograma sintético de raios X. (b) gráfico reconstruído.	51
V.14	Comparação da aproximação conseguida para o gráfico, no subintervalo ausente, através de <i>splines</i> cúbicas (curva tracejada) e da reconstrução fractal suavizada (traço sólido), com o gráfico original suavizado (curva pontilhada).	52

V.15 Sinal meteorológico real da direção do vento. Em destaque, a região que estamos supondo não ter sido amostrada.	53
V.16 (a) gráfico reconstruído usando $ s_8 = 0.1857$. (b) gráfico reconstruído usando $ s_8 = 0.6678$. Em (a) e (b) a curva pontilhada é o gráfico real. . . .	55
V.17 O traço sólido representa o gráfico reconstruído usando $s_8 = 0.2850$ e $o_8 = 1$ e a curva pontilhada, o gráfico real.	56
V.18 (a) gráfico original da direção do vento. (b) gráfico reconstruído.	57
V.19 Comparação da aproximação conseguida para o gráfico, no subintervalo ausente, através de <i>splines</i> cúbicas (curva tracejada) e da reconstrução fractal suavizada (traço sólido), com o gráfico real suavizado (curva pontilhada). . .	58
V.20 Sinal meteorológico da velocidade vertical do vento. Em destaque, o subintervalo no qual suporemos que não houve leitura.	59
V.21 (a) gráfico reconstruído usando $ s_{12} = 0.1046$. (b) gráfico reconstruído usando $ s_{12} = 0.6810$. Em (a) e (b) a curva pontilhada é o gráfico real. . .	61
V.22 O traço sólido representa o gráfico reconstruído usando $s_{12} = -0.29$ e $o_{12} = 1$ e a curva pontilhada, o gráfico real.	62
V.23 (a) gráfico original da velocidade vertical do vento. (b) aproximação conseguida através da reconstrução fractal.	63
V.24 Comparação da aproximação conseguida para o gráfico, no subintervalo ausente, através de <i>splines</i> cúbicas (curva tracejada) e da reconstrução fractal suavizada (traço sólido), com o gráfico original suavizado (curva pontilhada). .	64

Capítulo I

Introdução

Um problema rotineiro nas ciências experimentais é a falha de amostragem, ou leitura, de dados. São comuns situações nas quais, por falha humana, de equipamento ou por dificuldades técnicas, um sinal tenha deixado de ser amostrado durante um certo intervalo de tempo. Esses “buracos” que surgem nas leituras, mesmo quando não muito grandes, podem conter informações cruciais à análise do fenômeno em questão.

As técnicas convencionais, para a reconstrução desse tipo de falha, nem sempre se adequam satisfatoriamente a sinais com rápidas e bruscas variações. Nesses casos, é comum não contarmos com propriedades de suavidade, condição indispensável à maioria dos métodos clássicos. Esse é o caso de muitas medidas meteorológicas, para as quais, muitas vezes, só podemos nos valer da continuidade.

Nosso objetivo é reconstruir um sinal com uma falha intermediária de amostragem. Esse sinal será suposto apenas contínuo. Partindo da Interpolação Fractal [1], uma técnica recente e ainda não muito explorada, desenvolveremos a *Reconstrução Fractal*.

Neste trabalho apresentamos algoritmos eficazes para todas as etapas do processamento do sinal defeituoso. Como subproduto, temos também toda uma técnica para a aproximação de funções. Compararemos os resultados obtidos a partir desse método com os conseguidos usando *splines* cúbicas [7].

O problema

Temos um sinal, que foi amostrado regularmente, com uma porção intermediária faltante. Mais precisamente, sejam $f \in C[a, b]$ e $f_k = f(z_k)$ onde $z_k = a + k(b - a)/M$ com $k = 0, 1, \dots, M$, para algum $M > 3$. Conhecemos o sinal $\{f_k\}_{k \in K}$, $K = \{0, 1, \dots, M\} \setminus \bar{K}$ onde $\bar{K} = \{k_1 + 1, \dots, k_2 - 1\}$, $0 \leq k_1 < k_2 \leq M$. Queremos aproximar f_k para $k \in \bar{K}$.

Trabalharemos alguns exemplos reais onde f representará medidas climatológicas tais como velocidade vertical e horizontal do vento, direção do vento, temperatura, concentração de CO_2 , etc. Esses dados foram fornecidos pelo Instituto Nacional de Pesquisas Espaciais (INPE). Ressaltamos que esse tema de pesquisa surgiu de uma necessidade real por parte de um grupo de pesquisadores do INPE.

A reconstrução fractal

Essa é uma nova ferramenta para a reconstrução de sinais, com o tipo de falha descrita acima. Apenas continuidade e compacidade do sinal são necessários para que possamos usá-la. Uma característica que potencializa a eficiência do método é o grau de *auto-similaridade* do sinal. Fisicamente, podemos dizer que fenômenos auto-similares relacionam-se a situações onde as características dos processos em macro-escala envolvidos são semelhantes às dos processos em escalas menores. Esta propriedade pode ser encontrada em várias situações físicas, químicas, biológicas, etc. Daremos exemplos para sinais totalmente auto-similares e para outros nem tanto e veremos como se comporta nossa metodologia.

A reconstrução fractal consiste em gerar uma contração, num espaço métrico completo auxiliar, com uma propriedade muito particular: seu ponto fixo é o gráfico de uma função contínua, definida no mesmo domínio do sinal original, porém sem falha alguma. Esta função é tomada como aproximação para o sinal na porção não amostrada. Além disso, usando a reconstrução fractal, podemos nos valer de uma informação quase nunca utilizada: a dimensão fractal. Caso esse dado esteja disponível, ou possa ser estimado com precisão, podemos construir aproximações mais fiéis. No entanto, a impossibilidade de utilizarmos a dimensão fractal não inviabiliza de forma alguma o uso da reconstrução fractal.

***Splines* cúbicas**

Com o intuito de comparar os resultados obtidos com a reconstrução fractal, aproximaremos a porção não amostrada através de *splines* cúbicas. Esta é uma técnica tradicional, de fácil implementação e de bons resultados na prática [7]. Consiste de interpolar um conjunto de pontos, usando um polinômio cúbico por partes e impondo a condição adicional de que o resultado seja uma função com, pelo menos, derivada segunda contínua. Para que possamos utilizar esta técnica nos exemplos apresentados, devemos primeiramente suavizar o sinal em questão, pois estamos assumindo apenas continuidade. A comparação será feita da seguinte maneira: reconstruiremos o sinal usando a reconstrução fractal, em seguida suavizaremos o resultado obtido e também o sinal original, que será dado como entrada para o procedimento de reconstrução através de *splines* cúbicas. Por fim, compararemos os resultados de ambas as metodologias.

Nesse trabalho veremos que a técnica de reconstrução fractal propicia uma excelente alternativa para a reconstrução de sinais de variação brusca, amostrados regularmente, a exceção de uma falha intermediária, quando o objetivo for conseguir, sobretudo, informações qualitativas a respeito da porção não lida. Informações quantitativas podem ser conseguidas à medida que mais precisamente conheçamos a dimensão fractal do gráfico completo e que a propriedade de auto-similaridade esteja mais presente.

Capítulo II

Interpolação Fractal

A interpolação fractal é uma das aplicações da metodologia de geração de fractais através de IFS (*Sistema Iterativo de Funções*), desenvolvida por Barnsley *et al.* [3, 9]. Na primeira parte desse capítulo faremos uma revisão da teoria básica de IFS. Na segunda parte, veremos como a interpolação fractal surge quando consideramos IFS's particulares.

II.1 Sistema iterativo de funções

De maneira geral, temos um espaço métrico (M, d) onde gostaríamos de criar nossos fractais. Para isso vamos definir um espaço auxiliar

$$\mathcal{H}(M) = \{A \subset M \mid A \text{ é compacto e não vazio}\}$$

e uma função $h_d : M \times M \rightarrow \mathbb{R}$, dada por

$$h_d(A, B) = \max\{D(A, B), D(B, A)\},$$

onde

$$D(A, B) = \max_{x \in A} \min_{y \in B} \{d(x, y)\}.$$

Proposição II.1: *O par $(\mathcal{H}(M), h_d)$ é um espaço métrico.*

Prova: Apêndice, página 69. □

A função h_d é conhecida como *métrica de Hausdorff*. Quando o espaço métrico (M, d) for completo, o que será uma hipótese nossa, prova-se que $(\mathcal{H}(M), h_d)$ também o será (veja [8], por exemplo). No contexto de nosso trabalho, denominaremos por *fractal* qualquer elemento de $(\mathcal{H}(M), h_d)$.

Daqui em diante, denotaremos $\mathbf{N} = \{1, 2, \dots, N\}$. Sejam $w_n : M \rightarrow M$, $n \in \mathbf{N}$, contrações em (M, d) e α_n seus respectivos fatores de contração. Definimos $W : \mathcal{H}(M) \rightarrow \mathcal{H}(M)$ por

$$W(A) = \bigcup_{n \in \mathbf{N}} w_n(A), \quad (\text{II.1})$$

onde $w_n(A) = \{w_n(x) \mid x \in A\}$.

Proposição II.2: W é uma contração em $(\mathcal{H}(M), h_d)$ com fator de contração $\alpha \equiv \max_{n \in \mathbf{N}} \alpha_n$.

Prova: Apêndice, página 71. □

Pelo Teorema do Ponto Fixo de Banach, W tem um único ponto fixo em $\mathcal{H}(M)$, digamos \mathcal{A} .

Definição II.1: Um **Sistema Iterativo de Funções (IFS)** é um espaço métrico completo (M, d) mais um conjunto finito de contrações $w_n : M \rightarrow M$, $n \in \mathbf{N}$ com seus respectivos fatores de contração α_n . A notação usada para IFS é $\{(M, d); w_n, n \in \mathbf{N}\}$ (o espaço métrico pode ser omitido quando não houver perigo de ambigüidade). Denominam-se **atrator** e **fator de contração** do IFS o ponto fixo \mathcal{A} da contração W definida por (II.1) e o máximo dos fatores de contração α_n , respectivamente.

Exemplo II.1[Curva de Koch]: Sejam $M = \mathbb{R}^2$ e d a métrica euclidiana. Consideremos o IFS formado pelas contrações

$$w_n \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{3} \begin{pmatrix} \cos \theta_n & -\sin \theta_n \\ \sin \theta_n & \cos \theta_n \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a_n \\ b_n \end{pmatrix},$$

onde a tabela

n	θ_n	a_n	b_n
1	0	0	0
2	$\pi/3$	1	0
3	$-\pi/3$	$3/2$	$\sqrt{3}/2$
4	0	2	0

define os parâmetros de cada w_n .

O atrator deste IFS é a famosa curva de Koch, vista na Figura II.1.

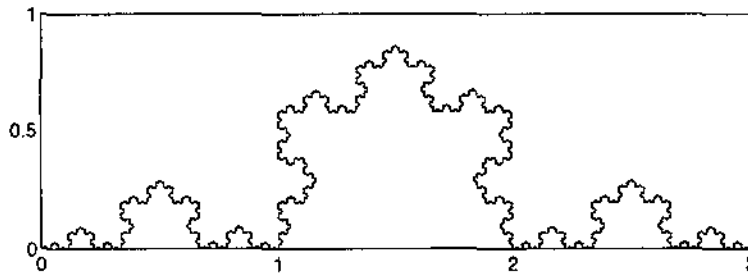


Figura II.1: Curva de Koch.

O interessante agora é usar IFS para nos ajudar a modelar e resolver nossos problemas. É nessa linha que surge o seguinte:

Teorema II.1[Teorema da Colagem]: *Seja (M, d) um espaço métrico completo. Sejam $\{(M, d); w_n, n \in \mathbf{N}\}$ um IFS, com fator de contração α , e $\mathcal{L} \in \mathcal{H}(M)$ tal que*

$$h_d \left(\mathcal{L}, \bigcup_{n \in \mathbf{N}} w_n(\mathcal{L}) \right) \leq \varepsilon,$$

para algum $\varepsilon > 0$. Então

$$h_d(\mathcal{L}, \mathcal{A}) \leq \frac{\varepsilon}{1 - \alpha},$$

onde \mathcal{A} é o atrator do IFS.

Prova: Apêndice, página 72. □

Se quisermos usar o atrator de um IFS como aproximação, ou modelo, para um dado conjunto \mathcal{L} , o teorema acima nos diz de que maneira devemos escolher as contrações w_n . Elas devem ser tais que a união das imagens de \mathcal{L} por estas funções não seja muito “diferente” de \mathcal{L} . Uma maneira de fazermos isto seria partir o conjunto \mathcal{L} em vários pedaços, tais que, cada um seja aproximadamente a imagem de \mathcal{L} por uma transformação w_n . O Teorema II.1 leva o nome de *Teorema da Colagem* justamente porque seriam estas imagens que *coladas* formariam o conjunto \mathcal{L} completo.

Exemplo II.2: *Suponha que quiséssemos construir um IFS, de tal maneira que seu atrator \mathcal{A} fosse uma aproximação da região do plano $\mathcal{L} = \{(x, y) \mid 0 \leq x, y \leq 1\}$. Poderíamos escolher, por exemplo, quatro transformações w_n da seguinte forma:*

$$w_n(x, y) = \frac{1}{2}(x + a_n, y + b_n),$$

onde a_n e b_n assumem os valores

n	a_n	b_n
1	0	0
2	1/2	0
3	0	1/2
4	1/2	1/2

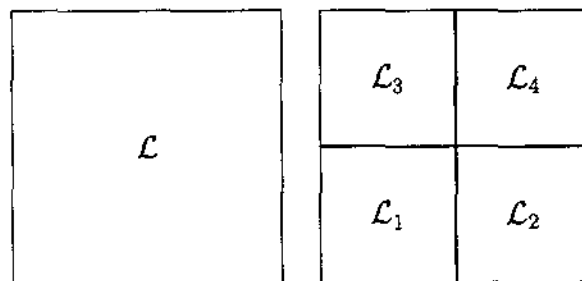


Figura II.2: Conjunto \mathcal{L} e união das imagens \mathcal{L}_i , $i = 1, 2, 3, 4$.

A primeira transformação mapeia o conjunto \mathcal{L} em $\mathcal{L}_1 = \{(x, y) \mid 0 \leq x, y \leq 1/2\}$, a segunda em $\mathcal{L}_2 = \{(x, y) \mid 1/2 \leq x \leq 1, 0 \leq y \leq 1/2\}$, a terceira em $\mathcal{L}_3 = \{(x, y) \mid 0 \leq x \leq 1/2, 1/2 \leq y \leq 1\}$ e a quarta em $\mathcal{L}_4 = \{(x, y) \mid 1/2 \leq x, y \leq 1\}$. Como \mathcal{L} é exatamente

igual a $\cup_{n=1}^4 \mathcal{L}_n$, o atrator \mathcal{A} é exatamente o conjunto \mathcal{L} . É como se tivéssemos “colado” os vários pedaços (\mathcal{L}_1 , \mathcal{L}_2 , \mathcal{L}_3 e \mathcal{L}_4) para formar \mathcal{L} .

Exemplo II.3[Folha de Samambaia]: Um exemplo clássico de como podemos usar o Teorema da Colagem é a Folha de Samambaia [4]. Esse IFS demonstra o quão poderoso é o teorema e como podemos conseguir ótimas aproximações para os conjuntos mais diversos. O IFS é formado por quatro transformações afins:

$$w_n \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r_n \cos \theta_n & -s_n \sin \phi_n \\ r_n \sin \theta_n & s_n \cos \phi_n \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a_n \\ b_n \end{pmatrix},$$

onde

n	r_n	s_n	θ_n	ϕ_n	a_n	b_n
1	0.00	0.16	0.0	0.0	0.0	0.00
2	0.85	0.85	-2.5	-2.5	0.0	1.60
3	0.30	0.34	49.0	49.0	0.0	1.60
4	0.30	0.37	120.0	-50.0	0.0	0.44

Seu atrator pode ser visto na Figura II.3.



Figura II.3: Folha de Samambaia.

II.2 Interpolação fractal

Vejam agora como a interpolação fractal se encaixa no esquema da seção anterior. O problema de interpolação é: Dado o conjunto $\Lambda = \{(x_n, y_n), n = 0, \dots, N\}$ (assumiremos sempre que $N > 1$), encontrar uma função $f : [x_0, x_N] \rightarrow \mathbb{R}$, contínua, tal que

$$f(x_n) = y_n, \quad n = 0, 1, \dots, N.$$

Tentaremos construir um IFS tal que seu atrator seja o gráfico de uma função contínua que interpole o conjunto de dados. Neste caso, $M = \mathbb{R}^2$ e d é a métrica euclidiana.

Já que queremos usar interpolação fractal, não seria estranho supor que a função interpolante tivesse características fractais, tais como auto-similaridade. Por um momento, suponhamos que uma função f , que interpola Λ , seja auto-similar, no sentido de que o seu gráfico em cada intervalo $[x_{n-1}, x_n]$ é a imagem de uma transformação atuando no gráfico todo. Mais claramente, suponhamos que para cada intervalo $[x_{n-1}, x_n]$ exista uma transformação $w_n : [x_0, x_N] \times \mathbb{R} \rightarrow [x_{n-1}, x_n] \times \mathbb{R}$ que satisfaça

$$w_n(\{(x, f(x)) \mid x_0 \leq x \leq x_N\}) = \{(x, f(x)) \mid x_{n-1} \leq x \leq x_n\}. \quad (\text{II.2})$$

Vamos tentar usar esta condição para determinar w_n . No entanto, note que, não dispomos realmente de (II.2). A única informação que temos com certeza é sobre os extremos dos intervalos, através do conhecimento de Λ . Impondo isto, temos

$$\begin{aligned} w_n(x_0, y_0) &= (x_{n-1}, y_{n-1}), \\ w_n(x_N, y_N) &= (x_n, y_n). \end{aligned} \quad (\text{II.3})$$

Uma tentativa para as w_n 's seria

$$w_n \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_n & 0 \\ c_n & s_n \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e_n \\ f_n \end{pmatrix}. \quad (\text{II.4})$$

Em virtude de (II.4), a condição (II.3) traduz-se em

$$\begin{aligned} a_n x_0 + e_n &= x_{n-1} \\ a_n x_N + e_n &= x_n \\ c_n x_0 + s_n y_0 + f_n &= y_{n-1} \\ c_n x_N + s_n y_N + f_n &= y_n \end{aligned} \quad (\text{II.5})$$

Essa é a condição clássica que encontramos em [1], mas poderíamos também deduzir de (II.2), sem problema algum, que

$$\begin{aligned} w_n(x_0, y_0) &= (x_n, y_n), \\ w_n(x_N, y_N) &= (x_{n-1}, y_{n-1}). \end{aligned} \quad (\text{II.6})$$

Isto foi uma generalização que fizemos. Quando formos construir cada w_n , poderemos optar agora por qual for mais conveniente. Vamos assim, introduzir um novo parâmetro o_n , que chamaremos de *coeficiente de orientação*. Ele nos indicará se estamos usando uma interpretação de (II.2) ou outra. Quando $o_n = 1$, estaremos considerando que w_n satisfaz (II.3) e quando $o_n = -1$ é porque w_n satisfaz (II.6).

Se $o_n = -1$, o sistema linear para os coeficientes de w_n torna-se

$$\begin{aligned} a_n x_0 + e_n &= x_n \\ a_n x_N + e_n &= x_{n-1} \\ c_n x_0 + s_n y_0 + f_n &= y_n \\ c_n x_N + s_n y_N + f_n &= y_{n-1} \end{aligned} \quad (\text{II.7})$$

Tanto o sistema (II.5) quanto o (II.7) são indeterminados. Escolhendo s_n como parâmetro livre, obtemos, se $o_n = 1$,

$$\begin{aligned} a_n &= \delta (x_n - x_{n-1}), \\ e_n &= \delta (x_N x_{n-1} - x_0 x_n), \\ c_n &= \delta ((y_n - y_{n-1}) - s_n (y_N - y_0)) \text{ e} \\ f_n &= \delta ((x_N y_{n-1} - x_0 y_n) - s_n (x_N y_0 - x_0 y_N)), \end{aligned} \quad (\text{II.8})$$

e, se $o_n = -1$,

$$\begin{aligned} a_n &= \delta (x_{n-1} - x_n), \\ e_n &= \delta (x_N x_n - x_0 x_{n-1}), \\ c_n &= \delta ((y_{n-1} - y_n) - s_n (y_N - y_0)) \text{ e} \\ f_n &= \delta ((x_N y_n - x_0 y_{n-1}) - s_n (x_N y_0 - x_0 y_N)), \end{aligned} \quad (\text{II.9})$$

onde $\delta = (x_N - x_0)^{-1}$.

A escolha de s_n como parâmetro livre não foi arbitrária. Na verdade, s_n pode ser interpretado geometricamente. Definamos a quantidade $h_f[x_0, x_N](x)$ como sendo a diferença

$f(x) - r(x)$, onde r é a reta que une os pontos $(x_0, f(x_0))$ e $(x_N, f(x_N))$. Se denotarmos por

$$x_g = \arg \max_{x_0 \leq x \leq x_N} |h_f[x_0, x_N](x)| \quad (\text{II.10})$$

e por

$$x_l = \arg \max_{x_{n-1} \leq x \leq x_n} |h_f[x_{n-1}, x_n](x)|, \quad (\text{II.11})$$

então podemos dizer que

$$s_n = \frac{h_f[x_{n-1}, x_n](x_l)}{h_f[x_0, x_N](x_g)}. \quad (\text{II.12})$$

Vejam na Figura II.4 (atrator do IFS de [2], pág. 217) como interpretaríamos s_n para cada um dos três subintervalos (aqui, $n = 3$) e em particular para o primeiro deles.

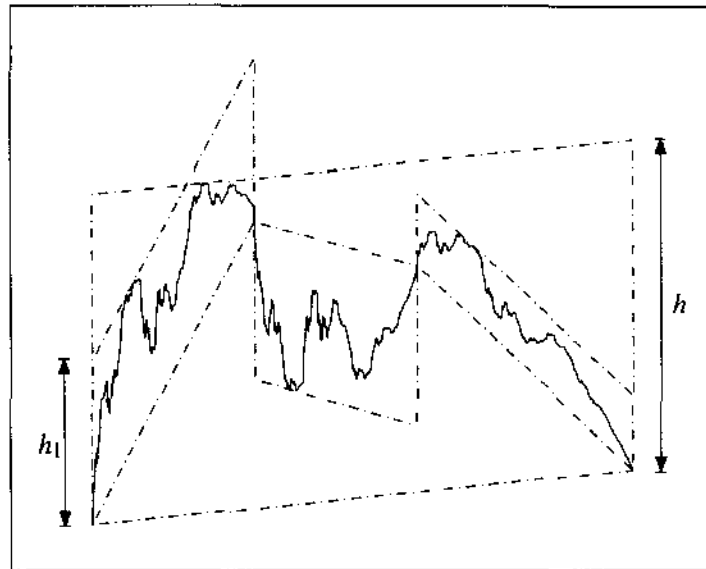


Figura II.4: Interpretação geométrica dos fatores de escala vertical. Aqui, $s_1 = h_1/h$, onde denotamos $h = h_f[x_0, x_N](x_g)$ e $h_1 = h_f[x_0, x_1](x_l)$.

Gostaríamos de construir um IFS usando estas w_n , mas primeiro precisamos provar que cada w_n é contração em (\mathbb{R}^2, d) . Infelizmente isto nem sempre é verdade.

Teorema II.2: *Existe uma métrica \tilde{d} , equivalente à métrica euclidiana, tal que w_n , como definida em (II.4) e (II.8) (ou (II.9)), é contração em $(\mathbb{R}^2, \tilde{d})$ se $|s_n| < 1$, para todo $n \in \mathbb{N}$.*

Prova: Apêndice, página 73. □

Sabemos agora que, com essas w_n e com o espaço $(\mathbb{R}^2, \tilde{d})$, temos um IFS. Mas será que o atrator desse IFS tem algo a ver com nosso problema? Vamos provar agora que o atrator é, efetivamente, o gráfico de uma função contínua que interpola Λ .

Teorema II.3: *Considere o IFS, associado a Λ , $\{(\mathbb{R}^2, \tilde{d}); w_n, n \in \mathbf{N}\}$, com w_n definidas em (II.4) e (II.8) (ou (II.9)). Suponha que todos os fatores de escala vertical s_n tenham módulo menor que um. Denote por G o atrator do IFS. Então G é o gráfico de uma função contínua $f_\Lambda : [x_0, x_N] \rightarrow \mathbb{R}$ que interpola Λ , isto é*

$$G = \{ (x, f_\Lambda(x)) \mid x \in [x_0, x_N] \},$$

com

$$f_\Lambda(x_n) = y_n, \quad \text{para todo } (x_n, y_n) \in \Lambda.$$

Prova: Apêndice, página 74. □

Definição II.2: *Seja $\Lambda = \{(x_n, y_n), n = 0, \dots, N\}$ um conjunto de dados. Considere o IFS associado a Λ , definido por $\{(\mathbb{R}^2, \tilde{d}); w_n, n \in \mathbf{N}\}$, onde cada w_n tem a forma (II.4) e os coeficientes a_n, c_n, e_n e f_n são dados por (II.8) se $o_n = 1$ e por (II.9) se $o_n = -1$, $o_n, n \in \mathbf{N}$, são os coeficientes de orientação e $s_n, n \in \mathbf{N}$, são os fatores de escala vertical, satisfazendo $|s_n| < 1$. Chamamos de **Função de Interpolação Fractal (FIF)** a função $f_\Lambda : [x_0, x_N] \rightarrow \mathbb{R}$ cujo gráfico é o atrator desse IFS.*

Exemplo II.4: *Consideremos um exemplo de interpolação fractal. Suponha que temos os pontos $\Lambda = \{(0, 30); (20, 20); (30, 25); (50, 30)\}$ e desejamos interpolá-los por uma função contínua. O primeiro passo é construir as três contrações w_n , mas para isso é preciso que antes escolhamos s_n e o_n , para $n = 1, 2, 3$. Temos toda a liberdade para isso, desde que respeitemos que $|s_n| < 1$ e $|o_n| = 1$. Por exemplo, escolhemos os parâmetros:*

n	s_n	o_n
1	-0.45	-1
2	-0.20	1
3	-0.50	1

Podemos agora construir a contração W . O ponto fixo de W , gráfico de uma função contínua que interpola os pontos de Λ , pode ser visto na Figura II.5.

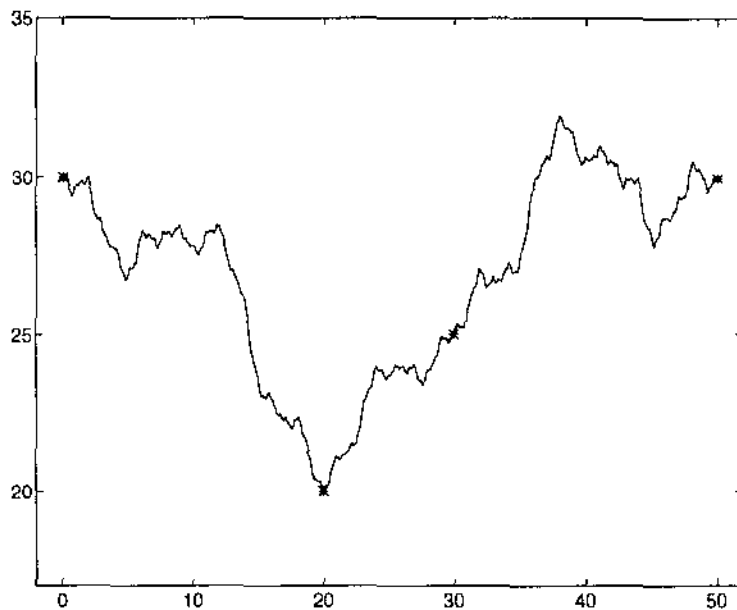


Figura II.5: Função de interpolação fractal para os pontos de Λ , marcados com asterisco no gráfico.

Ao fim desta seção, vemos que é possível interpolar um conjunto de dados Λ usando IFS. De fato, há uma maneira explícita de construir esse IFS. Provamos acima que seu atrator é o gráfico de uma função contínua f_Λ que passa por todos os pontos de Λ . Além disso, do ponto de vista da implementação da interpolação fractal, o único passo obscuro é como construir o atrator do IFS. Uma maneira seria utilizar o método do Teorema do Ponto Fixo, gerando uma seqüência convergente ao gráfico de f_Λ . Na prática isso não é eficiente. Na Seção IV.1 veremos um processo melhor. Desse modo, vemos que o problema

de interpolar um conjunto de pontos A usando a interpolação fractal já está completamente resolvido. O que ainda não sabemos é como usar a interpolação fractal para aproximar uma função conhecida. Se tivéssemos uma tabela com os valores de uma função f amostrados regularmente, como poderíamos criar uma FIF, de tal maneira que seu gráfico fosse uma boa aproximação do gráfico de f ? Isso é o que responderemos no Capítulo IV.

Capítulo III

Dimensão Fractal

Nesta seção nos ateremos aos resultados sobre dimensão fractal para funções de interpolação fractal. Para uma abordagem mais detalhada sobre o assunto, veja [8]. O cálculo da dimensão fractal terá um papel importante no nosso problema. Será através desta informação extra sobre os dados, que reconstruiremos a porção ausente.

Vamos começar definindo o número $\mathcal{N}(A, \epsilon)$ como o menor número de bolas fechadas de raio ϵ necessárias para cobrir $A \in \mathcal{H}(M)$, onde (M, d) é um espaço métrico completo. Esse número certamente existe e é finito, pois A é compacto. Intuitivamente, gostaríamos que um conjunto tivesse dimensão D , sempre que

$$\mathcal{N}(A, \epsilon) \approx C\epsilon^{-D},$$

onde C é alguma constante positiva, generalizando assim o conceito euclidiano de dimensão. Para que fique claro, o símbolo \approx tem o seguinte sentido

$$f(\epsilon) \approx g(\epsilon) \iff \lim_{\epsilon \rightarrow 0^+} \left(\frac{\ln f(\epsilon)}{\ln g(\epsilon)} \right) = 1.$$

A dimensão fractal de um conjunto seria definida, então, pelo número D tal que

$$D = \lim_{\epsilon \rightarrow 0^+} \left(\frac{\ln(\mathcal{N}(A, \epsilon))}{\ln(1/\epsilon)} \right).$$

Vejamos um teorema que nos ajudará mais a frente.

Teorema III.1: *Considere o espaço métrico (\mathbb{R}^m, d) , onde d é a métrica euclidiana. Sejam $A, B \in \mathcal{H}(\mathbb{R}^m)$. Se $D(B) \leq D(A)$, então $D(A \cup B) = D(A)$.*

Prova: Veja [2]. □

III.1 O algoritmo *Box Counting*

Acima, vimos como é definida a dimensão fractal para um elemento arbitrário A de $\mathcal{H}(M)$. No entanto, é patente que a definição formal de dimensão está longe de poder ser usada na prática. Calcular, ou pelo menos estimar, a quantidade $\mathcal{N}(A, \epsilon)$ só é viável para uma classe muito restrita de elementos de $\mathcal{H}(M)$.

Consideremos um teorema preliminar, que diz ser suficiente tomarmos uma seqüência discreta ϵ_n , que convirja a zero, para calcular a dimensão.

Teorema III.2: *Seja $A \in \mathcal{H}(M)$, onde (M, d) é um espaço métrico. Sejam $\epsilon_n = Cr^n$, para qualquer r que satisfaça $0 < r < 1$, e $C > 0$ com $n = 1, 2, 3, \dots$. Então*

$$D = \lim_{n \rightarrow \infty} \left(\frac{\ln \mathcal{N}(A, \epsilon_n)}{\ln(1/\epsilon_n)} \right)$$

é a dimensão fractal de A .

Prova: Apêndice, página 78. □

Para que haja esperança de conseguirmos algo melhor que a definição, vamos nos concentrar num caso mais simples. Ao invés de considerar $\mathcal{H}(M)$, com (M, d) um espaço métrico completo qualquer, pensemos apenas no caso em que M é o \mathbb{R}^m e d é a métrica euclidiana. Para esse caso temos o seguinte resultado.

Teorema III.3[*Box Counting*]: *Seja $A \in \mathcal{H}(\mathbb{R}^m)$, com d a métrica euclidiana em \mathbb{R}^m . Cubra o \mathbb{R}^m por hipercubos fechados, que se interceptem apenas nas faces, de lados $(1/2)^n$.*

Seja $\mathcal{N}_n(A)$ o número de hipercubos que interceptam A . Então

$$D = \lim_{n \rightarrow \infty} \left(\frac{\ln \mathcal{N}_n(A)}{\ln 2^n} \right) \quad (\text{III.1})$$

é a dimensão fractal de A .

Prova: Apêndice, página 79. □

O teorema de *Box Counting* nos diz que, para estimarmos a dimensão fractal de um elemento A qualquer de $\mathcal{H}(\mathbb{R}^m)$, devemos contar quantas células de aresta 2^{-n} , de uma malha regular, interceptam A , à medida que refinamos a malha. Isso é muito mais simples do que a definição pura de dimensão e pode inclusive ser implementado de maneira eficiente. No caso em que A for o gráfico de uma função contínua, a implementação fica, ainda, muito mais simples, gerando assim bons algoritmos para estimar a dimensão fractal.

III.2 Dimensão para funções de interpolação fractal

Para calcular a dimensão fractal do gráfico de uma função de interpolação fractal agimos de maneira diferente.

Teorema III.4: *Considere um IFS associado ao conjunto de dados Λ , como na Definição II.2. Seja G o atrator deste IFS. Se*

$$\sum_{n=1}^N |s_n| > 1$$

e os pontos de Λ não forem colineares, então a dimensão fractal de G é a única solução D de

$$\sum_{n=1}^N |s_n| |a_n|^{D-1} = 1. \quad (\text{III.2})$$

Caso contrário, a dimensão fractal de G é 1.

Prova: Veja [1]. □

Esse teorema junto com o Teorema de *Box Counting* são suficientes para os nossos objetivos. Com o segundo, estimaremos a dimensão fractal de um gráfico dado. Esse gráfico pode, até mesmo, com algumas adaptações, ter uma parte intermediária faltando, pois há um resultado que afirma que a dimensão fractal de um conjunto A é igual a dimensão fractal de $w(A)$ se w for bijetora [2]. Como já comentamos, estamos supondo que nossos gráficos tenham uma certa auto-similaridade, logo, dentro do próprio gráfico, haverá porções similares ao gráfico todo e dessa forma, tendo a mesma dimensão fractal.

Capítulo IV

Aproximação Fractal de Funções

A interpolação fractal é mais uma maneira de resolver o problema clássico de interpolação, isto é, dado um conjunto de pontos Λ , encontrar uma função contínua que passa por esses pontos. O problema central dessa tese é outro. Temos uma quantidade grande de pontos, suficientemente densos para representar, de maneira discreta, uma função, a exceção de um intervalo médio, do qual não temos informação. Queremos então encontrar uma função, ou o gráfico de uma, que aproxime o que ocorre nesse intervalo médio. Esse é um problema qualitativamente diferente do problema de interpolação.

Para abordar a questão da reconstrução de sinais, tivemos que passar por um problema intermediário. Suponha que temos uma função amostrada, de maneira suficientemente densa, de tal forma que isso já seja uma boa representação discreta, sem falha alguma. Precisamos descobrir como faríamos para aproximá-la por uma FIF, antes de pensar no caso da reconstrução de sinais. Isso é o que chamaremos de problema de *Aproximação Fractal de Funções*.

Basicamente, a estratégia para aproximar uma função por uma FIF seria eleger, dentre todos os pontos disponíveis, um subconjunto Λ de pontos, que seria usado então na interpolação fractal. A estes pontos em Λ daremos o nome de *pontos de corte*. Além disso, utilizar os pontos remanescentes como uma informação extra na determinação dos parâmetros livres na interpolação fractal, visando assim que o gráfico da FIF seja o mais próximo do gráfico da função. Esta função de interpolação fractal especial denominaremos por *Função de Aproximação Fractal*.

Para levarmos a cabo essa fase intermediária do projeto, precisamos esclarecer todos os passos envolvidos na construção da função de aproximação fractal. Este é o objetivo desse capítulo.

IV.1 Construindo o gráfico de uma FIF

Para trabalharmos com funções de interpolação fractal no MATLAB e gerá-las usando IFS, foi necessário que houvesse maneiras de computar eficientemente seu gráfico. Pelo teorema do ponto fixo de Banach e pela definição de fractal, o gráfico de uma função de interpolação fractal é o limite de uma seqüência iterada

$$\begin{cases} A^0 \in \mathcal{H}(\mathbb{R}^2) \text{ qualquer,} \\ A^{k+1} = W(A^k), \quad k = 1, 2, \dots \end{cases}$$

onde W é dada por (II.1), (II.4), (II.8) e (II.9). O conjunto A^0 mais simples que poderíamos escolher seria $A^0 = \{\mathbf{x}\}$ com $\mathbf{x} \in \mathbb{R}^2$. A cada iteração, o número de pontos em A^k se multiplicaria por N (número de subintervalos pelo qual dividimos o intervalo $[a, b]$). Logo haveria problemas de armazenamento. Outro problema seria a lentidão do processo. No k -ésimo passo de nosso suposto algoritmo, deveríamos aplicar N transformações a N^k pontos! Assim, mesmo para N pequeno, tal processo é praticamente inviável.

Baseados em Berger [5], implementamos um algoritmo que aproxima o atrator do IFS de outra forma. A idéia é descobrir sucessivos pontos pertencentes ao atrator. Sejam $G \in \mathcal{H}(\mathbb{R}^2)$ o gráfico de f_Λ , a função de interpolação fractal do conjunto de dados Λ , e W a contração construída em $\mathcal{H}(\mathbb{R}^2)$, como em (II.1). Sabemos que

$$W(G) = G. \quad (\text{IV.1})$$

De G , a princípio conhecemos apenas os pontos que estão em Λ . Seleccionamos um ponto qualquer de Λ , digamos $\mathbf{x}^0 = (x_j, y_j)$, para algum $0 \leq j \leq N$. Por (IV.1), sabemos que $W(\{\mathbf{x}^0\}) \subset G$, como

$$W(\{\mathbf{x}^0\}) = \bigcup_{n \in \mathbf{N}} w_n(\{\mathbf{x}^0\}),$$

sorteamos um n em \mathbf{N} e definimos $\mathbf{x}^1 = w_n(\mathbf{x}^0)$. Temos agora um \mathbf{x}^1 que certamente pertence a G . Aplicando o mesmo processo iterativamente, geramos uma seqüência de

pontos $\{\mathbf{x}^k\}$ em G . Podemos gerar tantos pontos quantos nos convir, e o fazemos até que tenhamos uma quantidade razoável de pontos, de maneira a representar suficientemente bem a função. A rotina implementada nos retorna o valor de f_Λ sob uma malha que definimos em $[x_0, x_N]$. Dado o carácter aleatório do algoritmo, não podemos assegurar que todos os pontos tenham sido efetivamente calculados. Em geral, ficamos satisfeitos quando 80% da malha tenha sido preenchida. Os pontos que, porventura, não foram atingidos, são aproximados por interpolação linear, através de seus vizinhos mais próximos que foram computados. Isto é o melhor que podemos esperar, pois só o que temos é continuidade da f_Λ , assim não podemos, e nem precisamos, tentar uma interpolação mais suave que a linear.

Implementamos um procedimento em MATLAB que gera as contrações relativas ao IFS interpolante de um conjunto de dados Λ e cujos fatores de escala vertical e coeficientes de orientação são fornecidos. Esse procedimento é, basicamente, a implementação de (II.4), (II.8) e (II.9).

IV.2 Estimando os fatores de escala vertical

Suponha que, dentre todos os pontos disponíveis, já escolhemos os pontos que pertecerão a Λ (esclareceremos isso mais adiante). Os parâmetros que ainda faltam ser determinados são os fatores de escala vertical e os coeficientes de orientação, para cada subintervalo.

Implementamos uma rotina para estimar os fatores de escala vertical. A aproximação de s_n é feita através de sua interpretação geométrica (II.12). Para tanto, precisamos do gráfico da função que queremos aproximar e dos pontos de corte.

Comprovamos a eficácia dessa aproximação com o seguinte teste. Primeiramente, construímos uma função de interpolação fractal, da qual conhecemos os fatores de escala vertical. Usando o algoritmo descrito na seção anterior, obtemos uma representação discreta da função. Isto foi usado como entrada para a rotina. A estimativa gerada foi muito próxima dos valores reais.

Apresentamos um exemplo que ilustra o desempenho dessa estratégia. Utilizamos a FIF encontrada em [6], denominada *difratograma sintético de raios X* (usaremos esta FIF em muitos outros exemplos, não devido a alguma característica especial que possa ter, mas

dada a praticidade de termos toda informação, sobre ela, compilada em [6]). Na Figura IV.1(a) podemos ver o gráfico real da FIF e na Figura IV.1(b) o gráfico construído com os fatores de escala vertical aproximados.

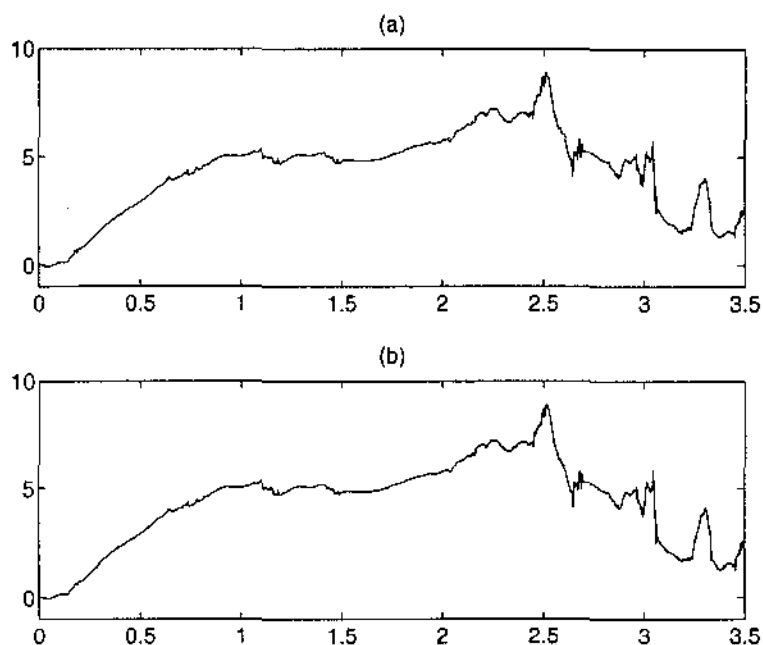


Figura IV.1: Teste de recuperação dos fatores de escala vertical. (a) gráfico original da FIF encontrada em [6]. (b) gráfico construído com os fatores de escala vertical estimados e os pontos de interpolação reais.

IV.3 Estimando os coeficientes de orientação

Como já dissemos anteriormente, o coeficiente de orientação é um parâmetro novo, introduzido por nós, generalizando a maneira como manipulamos a condição (II.2). Sua interpretação geométrica é simples. Se $o_n = 1$, a contração w_n mapeia o gráfico de f_Λ diretamente para o n -ésimo subintervalo. Se $o_n = -1$, antes de mapearmos o gráfico de f_Λ , fazemos uma reflexão em torno do eixo das ordenadas. Assim, o gráfico de f_Λ , no n -ésimo subintervalo, é uma versão espelhada do gráfico completo de f_Λ . Podemos ver isso na Figura IV.2. No primeiro subintervalo destacado, $[0.2, 0.4]$, $o_n = 1$. Claramente, o gráfico

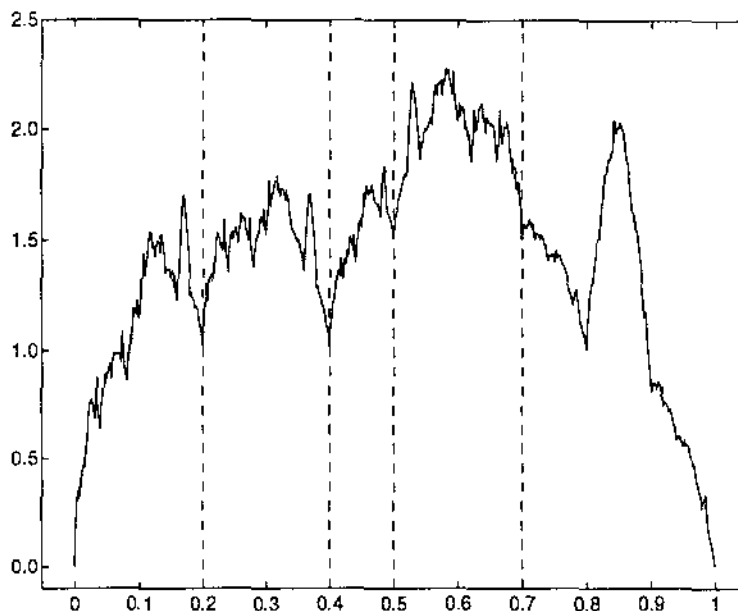


Figura IV.2: Interpretação geométrica do coeficiente de orientação. No intervalo $[0.2, 0.4]$, $o_n = 1$ e no intervalo $[0.5, 0.7]$, $o_n = -1$.

nesse subintervalo é uma versão reduzida do gráfico completo. No segundo subintervalo destacado, $[0.5, 0.7]$, $o_n = -1$, podemos ver que o gráfico foi primeiro refletido antes de mapeado.

Com a interpretação geométrica de o_n , podemos imaginar uma maneira de estimar os coeficientes de orientação. Em cada subintervalo $[x_{n-1}, x_n]$, fazemos um teste simples. Verificamos se x_g (II.10) está a direita ou a esquerda do ponto médio do intervalo $[a, b]$ e o mesmo para x_l (II.11), em relação ao intervalo $[x_{n-1}, x_n]$. Se ambos estiverem do mesmo lado, fixamos $o_n = 1$; caso contrário $o_n = -1$. Para validar o algoritmo, construímos várias funções de interpolação fractal, com diferentes conjuntos de coeficientes de orientação. Geramos seus gráficos e os fornecemos para nossa rotina juntamente com os pontos de corte. Para todos os testes, este algoritmo recuperou perfeitamente os coeficientes de orientação.

IV.4 Escolhendo os pontos de corte

Para construir a função de aproximação fractal já temos praticamente tudo automatizado. Esquemáticamente, agimos assim:

1. Temos o gráfico de uma função contínua $f : [a, b] \rightarrow \mathbb{R}$, discretizado sobre uma malha regular $a = z_0 < z_1 < \dots < z_M = b$,
2. Escolhemos os pontos de corte $\Lambda = \{(x_j, f(x_j)) \mid a = x_0 < \dots < x_N = b\}$,
3. Estimamos os fatores de escala vertical s_n , para $n = 1, \dots, N$ (Seção IV.2),
4. Estimamos os coeficientes de orientação o_n , para $n = 1, \dots, N$ (Seção IV.3),
5. Construimos o IFS (Seção IV.1),
6. Geramos uma tabela de pontos com os valores de f_Λ , numa partição regular de $[a, b]$ arbitrária (Seção IV.1).

De todos os passos acima, o único ainda não discutido é o segundo. Ressaltamos que não havia na literatura trabalho algum que se dispusesse a resolver tal questão. De que maneira devemos escolher os pontos de Λ , para que o gráfico de f_Λ seja uma boa aproximação do gráfico de f ? A resposta é simples e difícil ao mesmo tempo. Se nos lembrarmos do Teorema da Colagem, veremos que ele nos diz que o atrator de um IFS está próximo de um conjunto dado se a imagem desse conjunto pela transformação W está próxima de si mesma. Pensemos no caso ideal. Suponha que nos deram o gráfico de uma função contínua $f : [a, b] \rightarrow \mathbb{R}$, na verdade o atrator de um IFS. O gráfico **realmente** é um fractal. Suponha que o IFS gerador tenha sido construído com os pontos de $\Lambda = \{(x_j, f(x_j)) \mid a = x_0 < \dots < x_N = b\}$, os fatores de escala vertical s_n e os coeficientes de orientação o_n , $n \in \mathbb{N}$. Sabemos então que, para cada subintervalo, (II.2) se verifica, isto é,

$$w_n(\{(x, f(x)) \mid x_0 \leq x \leq x_N\}) = \{(x, f(x)) \mid x_{n-1} \leq x \leq x_n\}. \quad (\text{IV.2})$$

Foi com base em (IV.2) que desenvolvemos um algoritmo para encontrar os pontos de corte ótimos.

Algoritmo IV.1: Seleção dos pontos de corte

- (0) Seja $\{a = z_0 < z_1 < \dots < z_M = b\}$ uma malha regular de $[a, b]$. Defina $x_0 = z_0$, $k = 1$ e $i = 0$.
- (1) Para $j = i + 1, \dots, M$:
- (a) Construa a transformação w , como em (II.4), associada ao intervalo $[x_{k-1}, z_j]$ usando s estimado, como na Seção IV.2, e o estimado, como na Seção IV.3,
 - (b) Faça $w^{-1}(\{(x, f(x)) \mid x_{k-1} \leq x \leq z_j\}) = \{(u, v) \mid (u, v) = w^{-1}(x, f(x))\} \subset [a, b] \times \mathbb{R}$,
 - (c) Interpole linearmente os pontos $\{(u, v)\}$,
 - (d) Compute a norma relativa da diferença entre o gráfico $\{(u, v)\}$ e o gráfico $\{(x, f(x))\}$.
- (2) Escolha, como x_k , o ponto $z_{j'}$ para o qual obtivemos a menor norma da diferença no passo (1.d).
- (3) Faça $k = k + 1$, $i = j'$.
- (4) Se $i = M$, então encerre o algoritmo com $\{x_0, \dots, x_N\}$ sendo os pontos de interpolação selecionados; caso contrário volte a (1).

Inicialmente, definimos que o primeiro ponto de corte é o extremo esquerdo. Suponha que já descobrimos $(k - 1)$ pontos. Para descobrir o próximo, fazemos uma varredura em todos os pontos que estão à direita de x_{k-1} . Para cada ponto z_j , construímos a contração $w : [a, b] \times \mathbb{R} \rightarrow [x_{k-1}, z_j] \times \mathbb{R}$, como descrita por (II.4) e (II.8) (ou (II.9)), usando s e o estimados. Aplicamos a sua inversa à porção do gráfico relativa ao subintervalo $[x_{k-1}, z_j]$, mapeando-a em $[a, b]$. Como temos apenas alguns pontos no subintervalo, após mapeá-los em $[a, b]$, interpolamo-os linearmente para que possamos ter uma aproximação, na malha completa, do que seria a imagem inversa do gráfico em $[x_{k-1}, z_j]$ por w . Comparamos essa aproximação com o gráfico de f . Sabemos que, pelo fato do gráfico de f ser uma FIF,

existe um ponto $z_{j'}$, à direita de x_{k-1} , tal que a condição de auto-similaridade se verifica, isto é,

$$w_n(\{(x, f(x)) \mid a \leq x \leq b\}) = \{(x, f(x)) \mid x_{k-1} \leq x \leq z_{j'}\}. \quad (\text{IV.3})$$

Logo, escolhemos para $z_{j'}$ o z_j tal que, quando mapeamos $[x_{k-1}, z_j]$ em $[a, b]$, obtivemos o menor erro na norma 1. Também usamos a norma dois nessa medida, porém nenhuma diferença significativa nos pontos encontrados foi constatada. Assim a norma 1 foi a escolhida por ser mais barata. Repetimos o processo até que, por fim, tenhamos chegado a $x_N = b$.

Verificamos a eficácia dessa técnica aplicando-a justamente ao gráfico de uma função de interpolação fractal. Esse teste foi feito novamente com o difratograma sintético de raios X [6]. Na Tabela IV.1, colocamos os dados com os quais o IFS foi construído e os dados encontrados pela nossa metodologia. Os coeficientes de orientação são todos iguais a um

n	Dados reais			Dados recuperados		
	x_n	y_n	s_n	x_n	y_n	s_n
0	0.000	0.000	–	0.000	0.000	–
1	0.200	0.700	–0.0600	0.200	0.700	–0.0595
2	0.820	4.600	0.0800	0.820	4.600	0.0799
3	1.210	4.800	0.0900	1.210	4.800	0.0899
4	1.500	4.800	0.0750	1.500	4.800	0.0739
5	2.260	7.200	–0.1200	2.260	7.200	–0.1204
6	2.520	8.710	–0.2300	2.520	8.710	–0.2304
7	2.700	5.300	–0.3000	2.700	5.300	–0.2944
8	2.880	4.100	0.0550	2.880	4.100	0.0542
9	3.000	3.900	0.2000	3.000	3.900	0.1846
10	3.070	2.600	0.4000	3.070	2.600	0.3835
11	3.240	1.700	–0.0800	3.240	1.700	–0.0783
12	3.310	4.100	0.1000	3.310	4.100	0.0990
13	3.340	1.650	0.1000	3.340	1.650	0.1050
14	3.500	2.500	–0.1300	3.500	2.500	–0.1285

Tabela IV.1: Comparação entre os parâmetros reais usados na construção da FIF de [6] e os recuperados pelo Algoritmo IV.1.

e foram todos perfeitamente recuperados. Na Figura IV.3 podemos ver, em (a), o gráfico original e, em (b), o aproximado usando a informação estimada acima.

Com esse teste, e com outros que fizemos, pode-se ver que o procedimento funciona muito bem no caso ideal, isto é, quando o gráfico fornecido é um fractal gerado por um

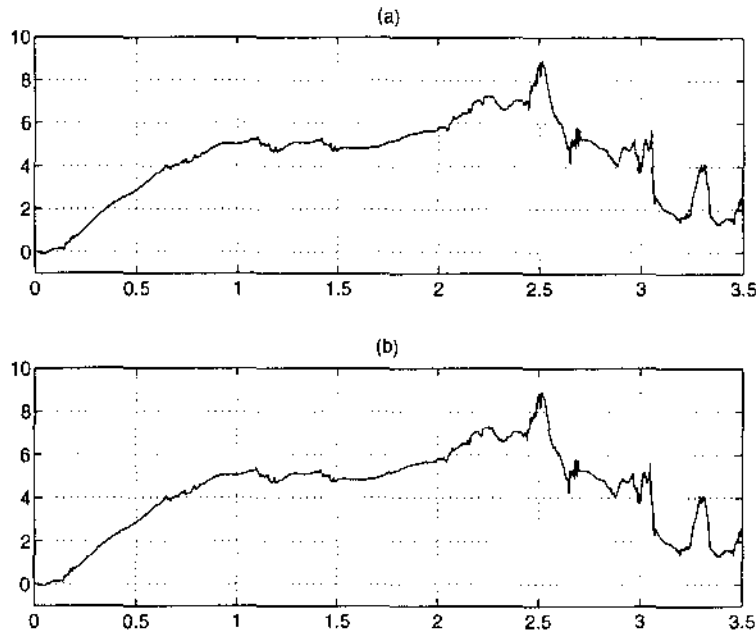


Figura IV.3: Teste do algoritmo de seleção dos pontos para o caso ideal. (a) Difratoograma sintético de raios X. (b) Gráfico construído com os pontos de corte, fatores de escala vertical e coeficientes de orientação estimados.

IFS. Precisamos ver se tal metodologia também funciona satisfatoriamente quando aplicada a um gráfico real.

Fizemos vários teste utilizando, a princípio, dados meteorológicos. Apesar de, na teoria, qualquer fator de escala vertical com módulo menor que um ser válido, na prática percebemos que isto não ocorre. É uma possibilidade do algoritmo poder limitar o módulo dos s_n . Esse recurso rendeu uma melhora significativa nos resultados.

IV.5 Exemplos reais trabalhados

Apresentamos aqui dois exemplos de funções meteorológicas reais que foram aproximadas usando nossa técnica. Ressaltamos que para ambos os casos, fixamos limitantes para o módulo máximo dos fatores de escala vertical. A escolha desse limitantes é empírica,

no sentido de que, primeiramente, aproximamos o gráfico sem limitante algum ($|s_n| < 1$, apenas). Caso o resultado tenha sido suficientemente bom, a nosso critério, damos-nos por satisfeitos. Se a aproximação resultante não for muito razoável, reduzimos o limitante. Não precisamos que essa limitação seja uniforme, isto é, podemos impor cotas superiores diferentes para subintervalos distintos do gráfico. Com isso, podemos melhorar a aproximação de maneira seletiva, concentrando-nos em regiões onde ainda não estamos satisfeitos.

Teste 1: Este é um gráfico da direção do vento, em relação ao norte, no sentido horário (Reserva Ducke, Floresta Amazônica), fornecido pelo INPE, com 1156 pontos (Figura IV.4(a)).

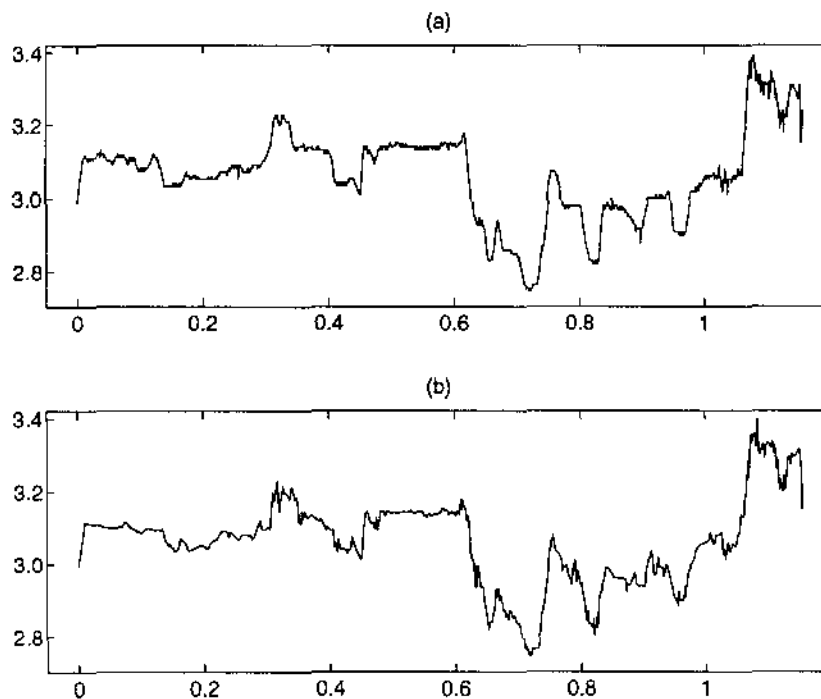


Figura IV.4: Direção do vento, Reserva Ducke, Floresta Amazônica - (a) gráfico original com 1156 pontos e (b) gráfico aproximado com 30 pontos e módulos máximos dos fatores de escala vertical limitados por 0.3 até o ponto 0.360, 0.15 até o ponto 0.566 e 0.3 até o ponto 1.156.

Através de nosso algoritmo, obtivemos uma aproximação com 30 pontos. O módulo máximo dos fatores de escala vertical foi controlado de maneira distinta para determinados intervalos, possibilitando assim um melhor ajuste. Para os subintervalos com extremo esquerdo menor ou igual ao ponto 0.360, impusemos um limitante de 0.3; para os subintervalos com extremo esquerdo a partir de 0.360 e menor ou igual a 0.566, o limitante foi 0.15; e, finalmente, para todos os subintervalos com extremo esquerdo à direita do ponto 0.566, impusemos um limitante de 0.3. Na Figura IV.4(b) podemos ver esta aproximação.

Teste 2: Esse é um gráfico da velocidade vertical do vento, à altura de 30m (Reserva Ducke, Floresta Amazônica), fornecido pelo INPE, com 1156 pontos (Figura IV.5(a)).

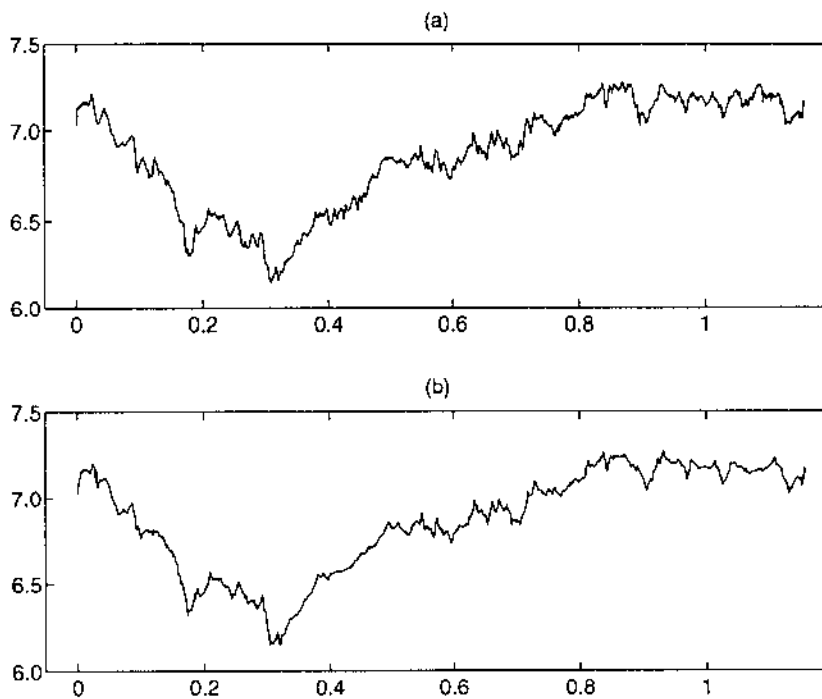


Figura IV.5: Velocidade vertical do vento, Reserva Ducke, Floresta Amazônica - (a) gráfico original com 1156 pontos e (b) gráfico aproximado com 44 pontos e limitação no módulo dos fatores de escala vertical de 0.2.

Nosso algoritmo obteve, impondo que os módulos dos fatores de escala vertical fossem

todos uniformemente limitados por 0.20, uma aproximação com 44 pontos de corte (Figura IV.5(b)).

Os exemplos apresentados são reais e servem para ilustrar o potencial da Aproximação Fractal de Funções. Sua utilidade não deve ser encarada como a mesma de outras técnicas aproximativas para funções. Esta é especialmente útil no caso de funções com comportamento fractal esperado. Os sinais meteorológicos são apenas um exemplo onde encontramos essa situação. Muitos outros podem ser encontrados, principalmente no âmbito das ciências experimentais, como Química, Física, Biologia entre tantas outras. Nesses casos, nem sempre representar o sinal “exatamente” é o objetivo. Para muitos propósitos, apenas manter as características principais (algo também a ser definido) pode ser mais que suficiente. Nessas situações, a Aproximação Fractal pode ser muito útil, possibilitando uma forma extremamente condensada de armazenar a informação amostrada.

Capítulo V

Reconstrução Fractal de Funções

No Capítulo IV, vimos como é possível aproximar funções usando a aproximação fractal. Veremos agora como utilizar isso na reconstrução de sinais. Basicamente, agiremos como no capítulo anterior, com algumas ressalvas. Em primeiro lugar, os pontos selecionados para Λ deverão, obrigatoriamente, conter os extremos do intervalo ausente e, em segundo lugar, precisaremos definir uma estratégia para escolher os parâmetros relativos a esse intervalo.

Nas técnicas tradicionais para a reconstrução de sinais, o fato de não termos amostra alguma num determinado intervalo implica que praticamente nada conhecemos sobre ele. Daí a dificuldade de aproximar o que acontece nessa porção obscura. A vantagem de usar reconstrução fractal, como mostraremos, é que, mesmo não conhecendo o gráfico num determinado subintervalo, temos ainda a informação, ou parte dela, referente a esse subintervalo, contida em outras porções do gráfico. Podemos afirmar isso, desde que estejamos assumindo que o gráfico tem a propriedade de uma certa auto-similaridade, uma de nossas premissas. A auto-similaridade afirma que um gráfico pode ser seccionado em vários pedaços, onde cada um desses é um mapeamento do gráfico todo. Quem faz o papel de ponte entre esses pedaços e o gráfico completo é a dimensão fractal. Será esta informação adicional que usaremos para estimar os parâmetros associados ao intervalo no qual o sinal não foi amostrado.

V.1 Escolhendo os pontos de corte

Como na aproximação fractal de funções, também é preciso escolher, para a reconstrução, um subconjunto Λ , dos pontos amostrados, para conter os pontos de corte. Porém, vamos forçar uma situação que não havia no caso anterior. Imporemos que os extremos do subintervalo ausente estejam em Λ . Lembrando do algoritmo utilizado na aproximação fractal, uma vez escolhidos os pontos de corte, utilizamos os pontos remanescentes para estimar o fator de escala vertical e o coeficiente de orientação, em cada subintervalo. Assim, acabamos de simplificar o problema. Inicialmente, não conhecíamos nada sobre um determinado subintervalo. Agora, não conhecemos apenas dois parâmetros para esse mesmo subintervalo. Se houver uma maneira de estimar essas quantidades, nosso problema estará resolvido.

Implicitamente, quando forçamos que os extremos do subintervalo ausente fizessem parte dos pontos de corte escolhidos, dissemos que a função é auto-similar nesse subintervalo. E se isso não for verdade? Bem, isso realmente não precisa ser verdade. É uma aproximação, e dado que nada conhecemos do gráfico nesse intervalo, é tudo o que podemos fazer. E se essa aproximação não for boa? Bom, isto também pode ocorrer. Contudo, esperamos que essa aproximação seja razoável para, pelo menos, subintervalos pequenos em relação ao intervalo completo.

Suponha que $[c, d]$ seja o subintervalo ausente. Usaremos o Algoritmo IV.1 com a alteração de que, se x_k estiver a esquerda de c , o indexador j variará até o índice correspondente ao ponto c . Se, porventura, o próprio ponto c for escolhido, pulamos diretamente para d e o fixamos como sendo o próximo ponto. Se c não foi escolhido, impomos que o seja e novamente pulamos para d .

V.2 Estimando a dimensão fractal

Quando estimamos a dimensão fractal de um gráfico temos duas situações diferentes. Se o objetivo fosse descobrir a dimensão fractal para o gráfico de uma função de interpolação fractal, tudo o que deveríamos fazer seria resolver a equação (III.2). Esse caso não traz

dificuldade alguma e foi implementado de maneira a aproximar a solução da equação pelo Método de Newton. Se, por outro lado, o gráfico em questão não fosse o de uma FIF, então a abordagem seria através do Teorema III.3 (*Box Counting*).

A implementação do algoritmo *Box Counting* é delicada. Para que tenhamos uma boa estimativa da dimensão fractal, devemos fazer a contagem das células que interceptam o gráfico de maneira cuidadosa. Precisamos levar em conta questões numéricas, inerentes à maneira como representamos as funções no computador.

Em primeiro lugar, para facilitar a manipulação dos dados, mapeamos o gráfico da função na caixa $[0, 1] \times [0, 1]$. Isso não altera sua dimensão e ainda ajuda a reduzir erros numéricos, tornando as escalas compatíveis em ambos os eixos. Para um determinado n , que define o tamanho 2^{-n} da célula, geramos uma tabela com os subintervalos de $[0, 1]$ correspondentes a cada coluna da malha. Para cada uma dessas colunas, calculamos em que células se encontram o máximo e o mínimo da função. Como a função é contínua, podemos dizer quantas células são interceptadas. Somando essas quantidades para todas as colunas, temos, ao final, o número total de células \mathcal{N}_n que interceptam o gráfico da função. Repetimos o processo para vários valores de n (na Figura V.1 podemos ver como ficam essas malhas). Fazendo uma regressão linear com o logaritmo de \mathcal{N}_n , estimamos a dimensão fractal do gráfico, pelo coeficiente angular da reta obtida. Na Figura V.2, podemos ver um exemplo de como ficam os pontos estimados por esse procedimento e como eles são ajustados por uma reta.

Há várias questões delicadas aqui. Somos nós que fixamos o n inicial (n_i), que o algoritmo usa para definir o tamanho inicial das células. A princípio, fixávamos n_i igual 1. As células tinham tamanho inicial de $1/2$ e, dessa forma, a região $[0, 1] \times [0, 1]$ era dividida em quatro células apenas. Por isso, \mathcal{N}_1 era muito instável. Uma leve mudança na função poderia resultar que seu gráfico cruzasse as quatro células ao invés de três, por exemplo, aumentando em 25% o valor de \mathcal{N}_1 . Isso pode acarretar um aumento, às vezes, significativo da dimensão estimada. Pensamos então que seria melhor começar com $n_i = 2$. Esperávamos que, desprezando o primeiro ponto, a aproximação seria mais estável, sem muito prejuízo pelo fato de termos menos pontos. O que ocorreu na prática foi que, para muitos casos, começar com $n_i = 2$ foi muito melhor que com $n_i = 1$. Porém, para um número igualmente significativo de casos, a escolha de $n_i = 1$, mostrou-se muito melhor.

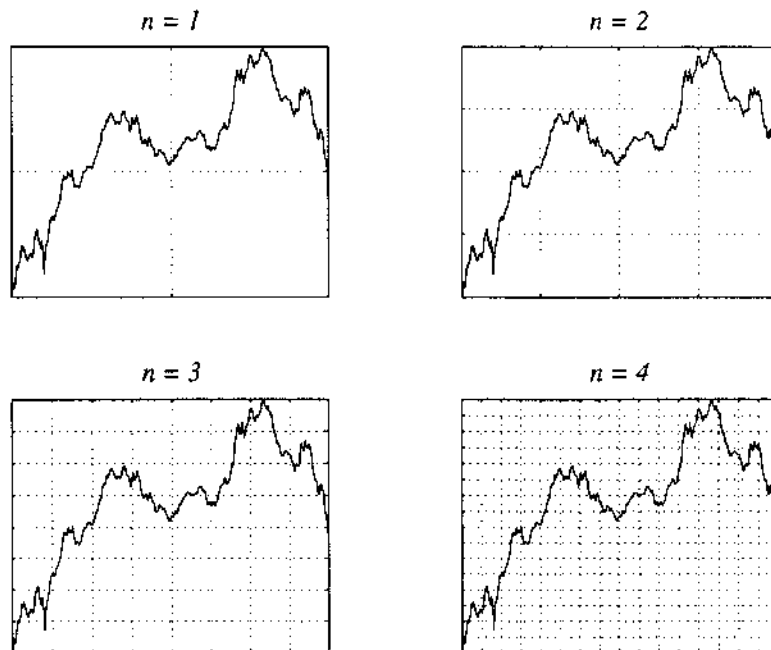


Figura V.1: Exemplo de malhas geradas pelo algoritmo de *Box Counting* para estimar a dimensão fractal.

Da mesma forma que a escolha do n_i inicial relaciona-se a melhores ou piores estimativas da dimensão fractal, a escolha do n_f final também. A princípio, poderíamos pensar que, quanto mais \mathcal{N}_n pudéssemos computar, melhor seria nosso resultado. Temos uma limitação imposta pela representação das funções. Induzidos pela maneira como o MATLAB trabalha, as funções são representadas como uma quantidade finita de pontos interpolados linearmente. Aumentando indiscriminadamente n_f , a partir de um certo \bar{n} , as células ficam pequenas demais para que, dentro de cada uma delas, haja um ou nenhum ponto, prejudicando assim o desempenho do algoritmo. Logo, \bar{n} deve ser menor que $\ln M / \ln 2$, onde M é a quantidade de pontos amostrados que temos. Fixamos \bar{n} como o maior inteiro menor que $\ln M / \ln 2$. Para os exemplos que mostraremos, em geral, com $1000 < M < 1500$, fixamos $n_f = \bar{n} = 9$ ou 10 .

Avaliamos o desempenho do algoritmo, estimando a dimensão fractal de gráficos de funções de interpolação fractal, já que para esses, conhecemos os valores reais de suas

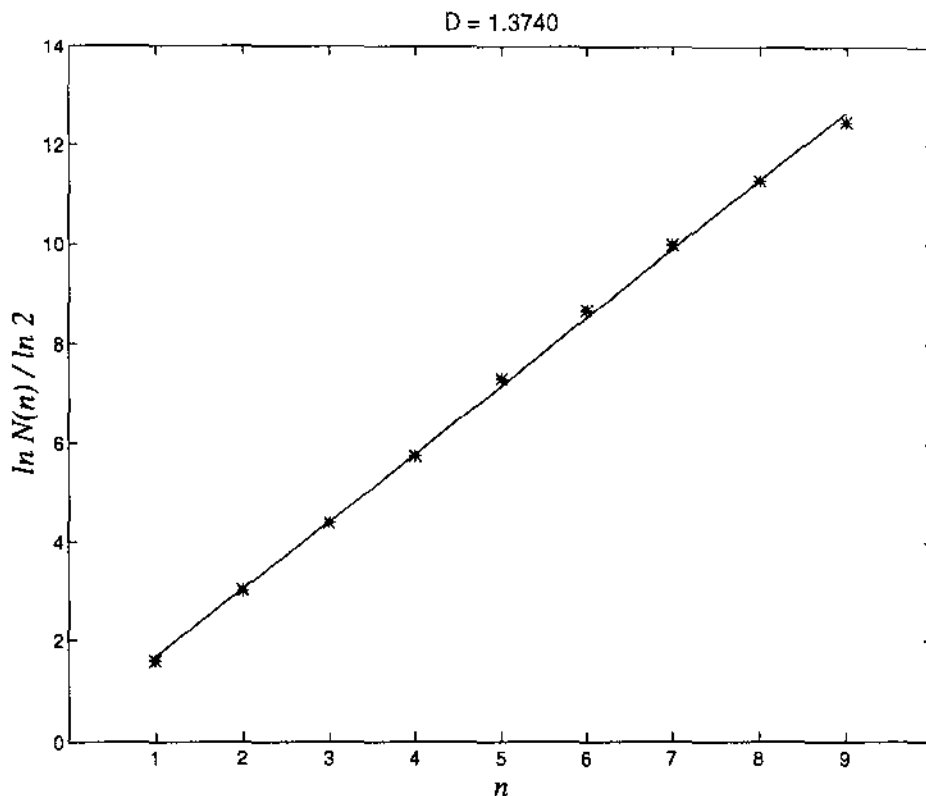


Figura V.2: Exemplo de estimativa gerada pelo algoritmo *Box Counting*.

dimensões (Seção III.2). Trabalhamos com $n_i = 2$ e $n_f = 9$ ou 10. Frequentemente, o algoritmo subestimava a dimensão. O erro podia chegar a até 8%, ficando, na média, em 5.2%. Notamos que, para alguns problemas, o erro era menor para um n_f menor que \bar{n} . Avaliamos para qual n_f conseguíamos o menor erro e, em geral, foi para aquele cuja estimativa da dimensão era a maior. Nossa estratégia passou a ser: estimar a dimensão fractal para $n_i = 2$ e n_f variando entre 3 e 9, tomando como aproximação final, a dimensão que resultasse ser a maior dentre todas. Isso melhorou algumas aproximações e piorou outras. Decidimos variar também n_i . Para cada problema teste, fizemos $n_i = 1, 2, 3$. Para cada n_i fixo escolhemos a dimensão que resultou ser a maior para $n_f = n_i + 1, \dots, 9$. Tendo esses três valores, escolhemos, como aproximação final, o menor deles. Com essa estratégia, o erro ficou, em média, em torno de 2.6%. Na Figura V.3, apresentamos um exemplo do comportamento da dimensão estimada quando variamos $n_i = 1, 2, 3$ e $n_f = n_i + 1, \dots, 9$.

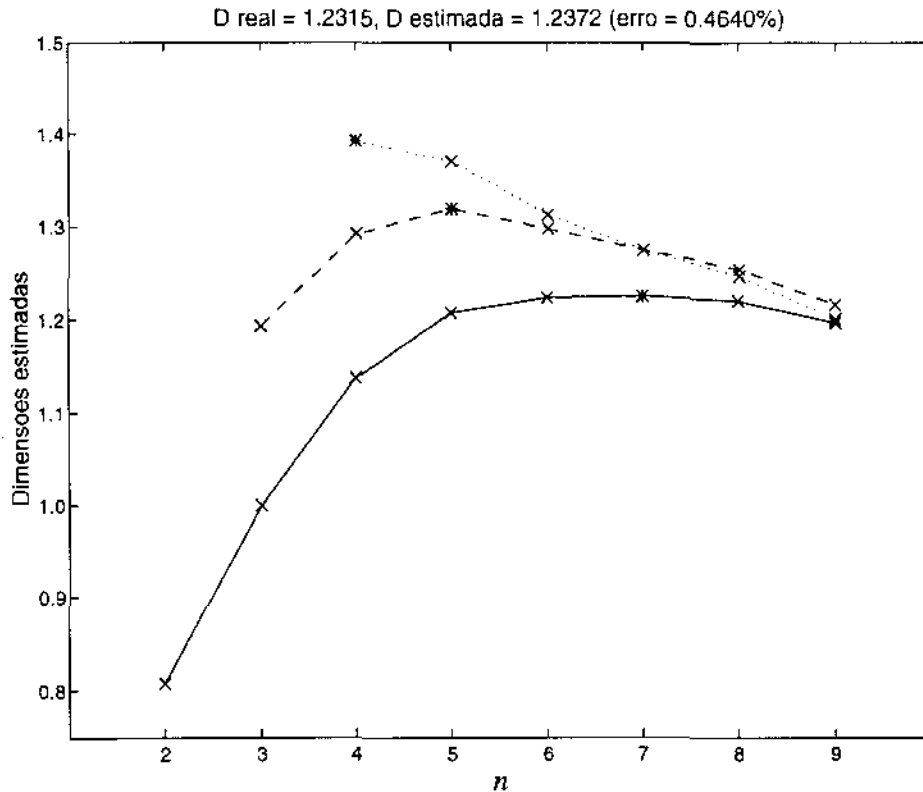


Figura V.3: Comportamento do estimador da dimensão para vários valores de n_i . Para a curva sólida, $n_i = 1$ e $n_f = 9$, com $D = 1.2372$; para a curva tracejada, $n_i = 2$ e $n_f = 9$, com $D = 1.3387$ e, para a curva pontilhada, $n_i = 3$ e $n_f = 9$, com $D = 1.4263$.

Na Figura V.3, podemos perceber também que, a máxima dimensão, para quaisquer um dos n_i escolhidos, nunca foi a calculada com $n_f = 9$ e essa era a aproximação que tomávamos a princípio. Isso ocorreu para todos os outros casos que estudamos. Na maioria deles, o máximo se concentrava em torno de $n_f = 5$. A explicação é que, para n_f ao redor de 5, o algoritmo já está considerando células pequenas o suficiente para conseguir uma boa aproximação, porém não tão pequenas, a ponto de perceber que, na verdade, o gráfico em questão é linear por partes, devido a sua representação no computador. À medida que as células reduzem de tamanho, e portanto, o algoritmo começa a considerar pequenas porções individuais do gráfico, o fato de ser linear por partes acarreta um decréscimo na dimensão estimada, pois, para funções desse tipo, a dimensão seria 1.

Temos assim, uma forma suficientemente boa de calcular a dimensão fractal de um gráfico. No entanto, o que realmente precisamos é uma forma de estimar a dimensão, no caso em que não conhecemos um determinado subintervalo.

Vamos considerar o problema de reconstrução e, da mesma forma que antes, comecemos imaginando que a função em questão é uma FIF. Além disso, suponhamos também que o intervalo ausente seja exatamente um dos subintervalos onde a função tem a propriedade de auto-similaridade. Como vimos a pouco, podemos determinar os pontos de corte sem dificuldade. Temos então, os subintervalos onde a propriedade de auto-similaridade se verifica. Em virtude disso, a dimensão fractal do gráfico em cada um desses subintervalos é igual a dimensão do gráfico todo. Como nossa intenção é estimar a dimensão do gráfico completo, poderíamos usar, como aproximação, as dimensões estimadas em cada subintervalo.

Na prática, essa idéia funciona tanto melhor quanto mais pontos tivermos amostrados em cada subintervalo. Nem sempre isso é possível. Em geral, a quantidade de pontos nos subintervalos é muito inferior que no gráfico todo. Se essa quantidade for muito baixa, podemos ter estimativas muito ruins da dimensão. Um exemplo de como isso ocorre é que, para a FIF em [6], tabelada em 1024 pontos, a estimativa da dimensão fractal do gráfico todo é bem razoável. No entanto, o maior de seus subintervalo teria 223 pontos, o que é muito pouco para que uma boa estimativa da dimensão possa ser feita.

Poderíamos agir de outra maneira. Denote por G_n o gráfico sobre o n -ésimo subintervalo e por G o gráfico todo. Sabemos que $D(G_n) = D(G)$, para todo n . Pelo Teorema III.1, $D(\cup_{n \in \mathcal{N}} G_n) = D(G)$, onde $\mathcal{N} \subset \mathbb{N}$. Em conseqüência, uma idéia seria tomar cada um dos lados do gráfico (à direita e à esquerda do intervalo ausente), estimar sua dimensões e utilizá-las como aproximação para a dimensão do gráfico completo. Isso já é melhor que a idéia anterior, na medida que utilizamos “pedaços” maiores de gráfico para estimar a dimensão. Em alguns casos, esta abordagem foi boa, porém, para uma quantidade ainda expressiva, continuamos tendo resultados inadequados.

Independentemente da maneira como iremos aproximar a dimensão fractal do gráfico com falha de amostragem, temos uma forma confiável de conseguir cotas inferior e superior para a dimensão. Se isto não é suficiente para estimarmos o fator de escala vertical e o coeficiente de orientação, no mínimo, serve para descartar “aproximações” grosseiras da dimensão. Vejamos como isso pode ser feito.

Suponha que temos um gráfico amostrado com falha. Após selecionarmos os pontos de corte, observamos que a porção ausente é o m -ésimo subintervalo. Usando a equação (III.2), do Teorema III.4, isolamos $|s_m|$, obtendo

$$|s_m| = \frac{1}{|a_m|^{D-1}} \left(1 - \sum_{\substack{n=1 \\ n \neq m}}^N |s_n| |a_n|^{D-1} \right). \quad (\text{V.1})$$

Assim, temos $|s_m|$ como função de D , a dimensão do gráfico. Vejamos, na Figura V.4, um exemplo de como seria o comportamento desta função, para $D \in [1, 2)$ (os possíveis valores para dimensão fractal de uma função contínua de uma variável a valores reais).

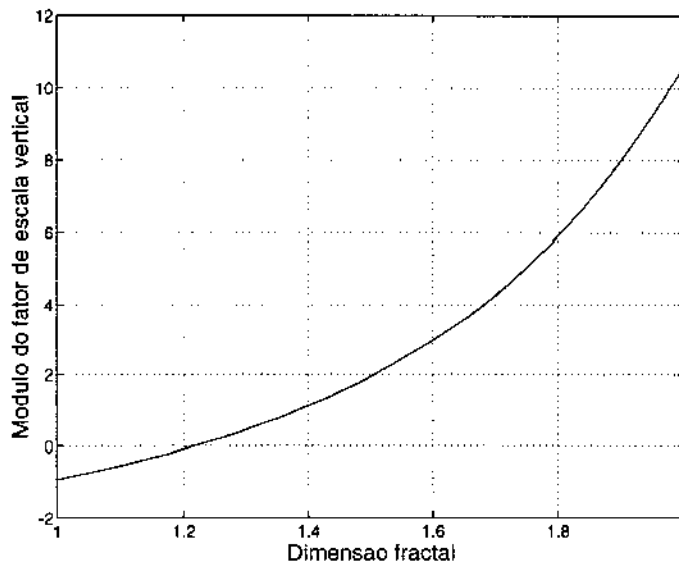


Figura V.4: Comportamento do módulo do fator de escala vertical do subintervalo ausente em função da dimensão fractal do gráfico.

Perceba que nem todos os valores de $|s_m|$ são aceitáveis. Sabemos que $0 \leq |s_m| < 1$. Assim, apenas alguns valores de D são válidos, neste exemplo. Determinando o valor de D para o qual $|s_m| = 0$ e aquele para o qual $|s_m| = 1$, temos os respectivos valores máximo e mínimo de D , possíveis para esta função. Dessa maneira, conseguimos cotas inferior e superior para a dimensão fractal do gráfico. Sabendo isso, podemos descartar estimativas que estejam fora desta faixa. Na Figura V.5, vemos a região aceitável do gráfico de $|s_m|$ em função da dimensão fractal, para o exemplo anterior.

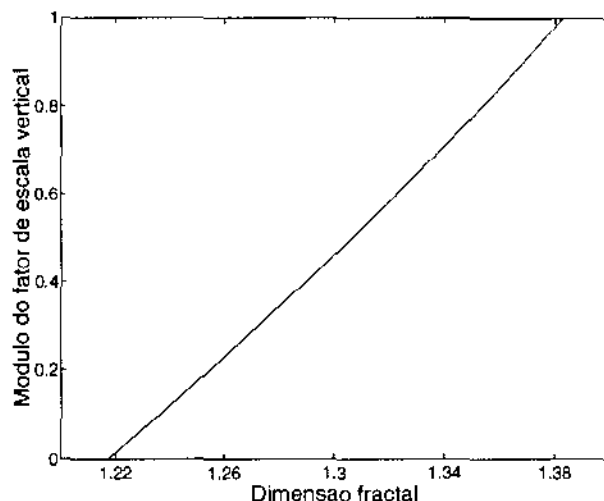


Figura V.5: Comportamento do módulo do fator de escala vertical do subintervalo ausente em função da dimensão fractal do gráfico, apenas para o intervalo aceitável da dimensão.

Devemos observar também que o módulo de s_m varia muito rápido. Enquanto D percorre um intervalo de comprimento 0.166, $|s_m|$ vai de 0 a 1. Essa característica acentua-se à medida que a_m diminui. Ou seja, quanto menor for o comprimento do subintervalo ausente, mais sensível é o módulo do fator de escala vertical a variações da dimensão fractal. Podemos ver isto pela expressão (V.1), na qual temos o fator $1/|a_m|^{D-1}$. Esse tipo de comportamento contradiz o senso comum. Seria natural esperar que o problema fosse mais bem condicionado, à medida que o intervalo ausente tivesse seu comprimento reduzido. No entanto, faz sentido, pois se desejássemos mudar a dimensão fractal de um gráfico, às custas apenas do fator de escala vertical de um intervalo pequeno, esse deveria variar muito para que seu efeito fosse perceptível.

Em vista dessa discussão, achamos importante que se invista em buscar maneiras mais eficientes de estimar a dimensão fractal, para o problema com falha de amostragem. Certamente, ganhos nesse sentido revelariam-se em melhorias grandes ao processo de reconstrução como um todo.

Aplicando as mesmas idéias, para o caso onde a função em questão não é uma FIF, conseguimos resultados semelhantes.

V.3 Estimando os parâmetros do intervalo obscuro

Precisamos estimar o fator de escala vertical e o coeficiente de orientação para o intervalo ausente. Feito isso, estaremos aptos a reconstruir a porção faltante do gráfico, aproximando-a pela sua análoga na função de reconstrução fractal.

Para estimar o fator de escala vertical procedemos como discutido na seção anterior. Estimamos a dimensão fractal do gráfico e, usando a fórmula (V.1), encontramos o módulo de s_m . Vimos também, que pela própria característica dessa equação, a estimativa de $|s_m|$ é muito sensível a variações da dimensão fractal. Sem dúvida, fica aberta aqui uma possibilidade para trabalhos futuros: conseguir uma relação mais estável entre o fator de escala vertical e a dimensão fractal.

Falta decidir sobre o sinal de s_m e sobre o_m . Fazemos isso, testando as quatro possibilidades e escolhendo a que resultou um erro menor na aproximação do gráfico real pelo gráfico da função de reconstrução fractal.

V.4 Exemplos trabalhados

Vamos agora nos concentrar em alguns exemplos de como reconstruir sinais que, eventualmente, tenham sido amostrados com falha.

Difratograma sintético de raios X (situação ideal): Este sinal é uma FIF que aproxima um difratograma de raios X, medido no Instituto de Química da Unicamp, obtida empiricamente em [6]. Nesse exemplo tentaremos reconstruir o gráfico em questão quando a porção ausente for exatamente um dos subintervalos onde o gráfico é auto-similar.

Este gráfico está definido em $[0, 3.5]$ e representado por 3501 pontos igualmente espaçados. Vamos considerar $[2.26, 2.52]$ o intervalo ausente, dentro do qual há 259 pontos, correspondendo a 7.40% da amostra completa. Isto pode ser visto na Figura V.6.

O primeiro passo é encontrar os pontos de corte. Não impusemos restrições adicionais ao módulo dos fatores de escala vertical, isto é, $|s_n| < 1$, para todo n . Utilizando nosso algoritmo, determinamos 19 subintervalos. Na Tabela V.1, podemos ver os pontos de corte, os fatores de escala vertical e os coeficientes de orientação encontrados.

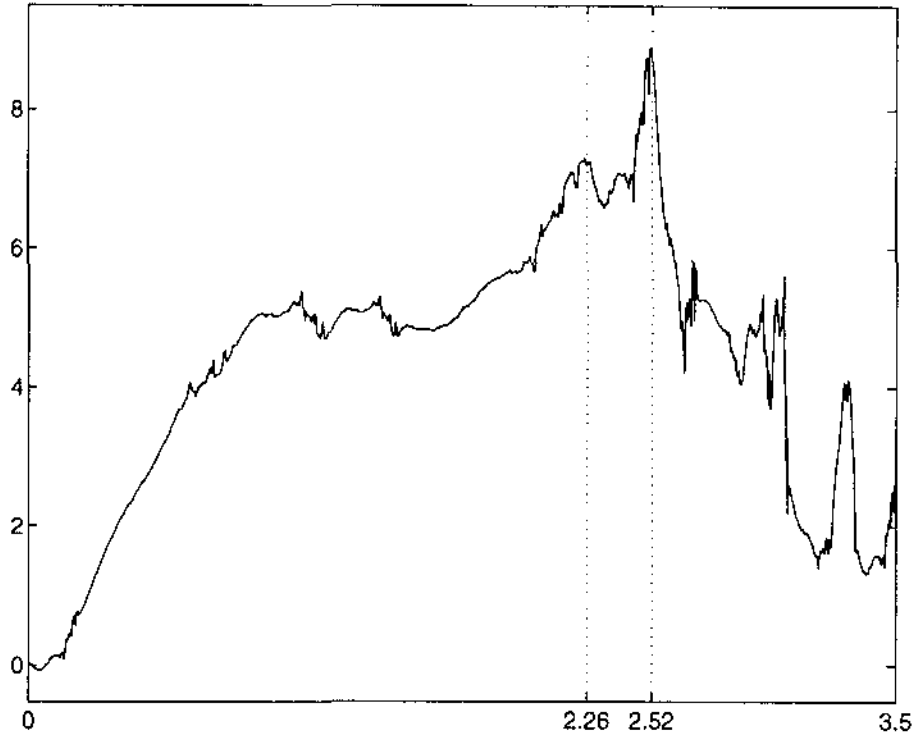


Figura V.6: Difratograma sintético de raios X. Em destaque, um subintervalo onde o gráfico é auto-similar e o qual tentaremos reconstruir.

De acordo com os pontos de corte determinados pelo algoritmo, o subintervalo ausente é o sexto subintervalo, para o qual ainda temos de estimar tanto o fator de escala vertical quanto o coeficiente de orientação.

Precisamos de uma boa estimativa da dimensão fractal do gráfico para que possamos calcular o módulo do fator de escala vertical de maneira satisfatória. Como discutimos na Seção V.2, é fácil conseguirmos limitantes inferior e superior para a dimensão fractal. Calculando-os, concluímos que a dimensão fractal deve estar entre 1.1990 e 1.3600. Vamos agora utilizar o algoritmo de *Box Counting* para calcular a dimensão para ambos os lados do gráfico (à esquerda e à direita do intervalo ausente) e representá-las por D_- e D_+ , respectivamente. As estimativas geradas para os diferentes valores de n_i (potência que determina o tamanho inicial das células, Seção V.2) são

n	x_n	y_n	s_n	o_n
0	0.0000	0.0000	-	-
1	0.2000	0.7000	-0.0593	1
2	0.8200	4.6000	0.0798	1
3	1.2100	4.8000	0.0899	1
4	1.5000	4.8000	0.0705	1
5	2.2600	7.2000	-0.1197	1
6	2.5200	8.7100	-	-
7	2.7000	5.3000	-0.2916	1
8	2.8800	4.1000	0.0545	1
9	3.0000	3.9000	0.1964	1
10	3.0700	2.6000	0.3732	1
11	3.2400	1.7000	-0.0783	1
12	3.3100	4.1000	0.0890	1
13	3.4060	1.4907	-0.2332	-1
14	3.4430	1.4731	0.0158	1
15	3.4560	1.4086	0.0234	1
16	3.4750	2.0124	0.0457	1
17	3.4910	2.2285	0.0509	-1
18	3.4960	2.5953	0.0263	1
19	3.5000	2.5000	0.0101	1

Tabela V.1: Parâmetros determinados na reconstrução do difratograma de raios X.

n_i	D_-	D_+
1	1.1095	1.1919
2	1.1475	1.3219
3	1.2895	1.1984

Perceba que há uma variação significativa dessas estimativas e, além disso, apenas duas se encontram dentro dos limitantes calculados. Se assumirmos que a dimensão fractal é 1.2895, obteremos $|s_6| = 0.5246$. Por outro lado, se considerarmos que 1.3219 é uma melhor aproximação, obteremos $|s_6| = 0.7351$. Lembrando que o fator de escala vertical real para este subintervalo tem módulo igual a 0.23 e a dimensão real é 1.2315, constatamos que nenhuma das duas aproximações foi boa. A Figura V.7 mostra como seria reconstruído o gráfico, no intervalo ausente, usando esses dois valores.

Como não conseguimos uma forma precisa de calcular a dimensão fractal para gráficos

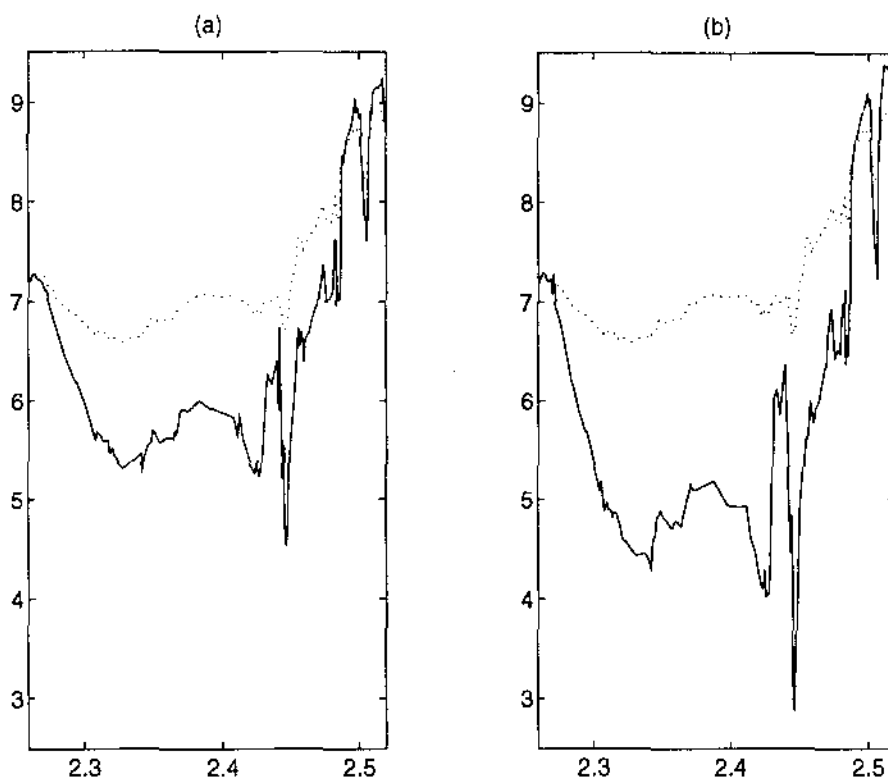


Figura V.7: (a) gráfico reconstruído usando $|s_6| = 0.5246$. (b) gráfico reconstruído usando $|s_6| = 0.7351$. Em (a) e (b) a curva pontilhada é o gráfico real.

com falhas de amostragem e como o cálculo do módulo do fator de escala vertical é muito sensível a dimensão, optamos por não usá-la. O que fizemos foi mais eficiente, apesar de menos elegante e mais caro computacionalmente. Para o módulo do fator de escala vertical variando de 0 a 0.9 com um passo de 0.01, usando as quatro possíveis combinações dos sinais de s_n e o_n , reconstruímos o gráfico e o comparamos com o gráfico original (desconsiderando o intervalo ausente). Escolhemos s_6 e o_6 que resultaram na melhor aproximação. Neste exemplo, $s_6 = -0.219$ e $o_6 = 1$. Conseguimos um resultado muito bom com essa estratégia. Na Figura V.8, podemos ver, no subintervalo ausente, o gráfico real e o reconstruído e, na Figura V.9, o resultado final da reconstrução fractal.

Para efeito de comparação, reconstruímos a mesma porção do gráfico usando, porém, *splines* cúbicas. Como assumimos apenas continuidade para as funções, antes de usarmos

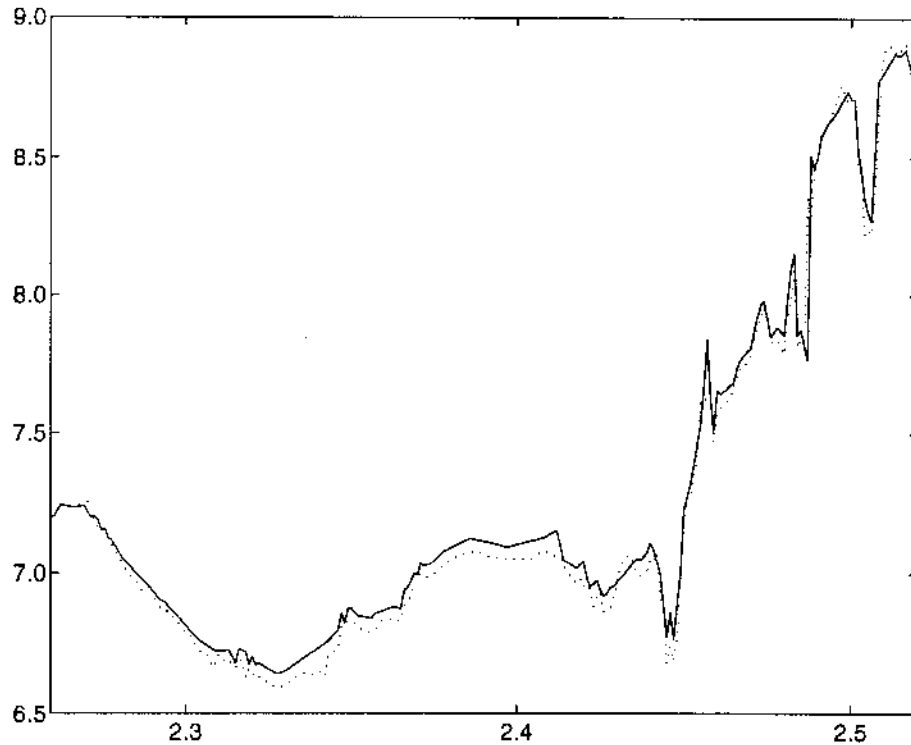


Figura V.8: Subintervalo reconstruído. Gráfico real (pontilhado) e reconstruído (traço sólido) com os parâmetros do intervalo ausente $s_6 = -0.219$ e $\sigma_6 = 1$.

splines, suavizamos o gráfico original, além disso, após fazermos a reconstrução fractal, suavizamos sua resposta para podermos compará-la. Na Figura V.10, vemos que a reconstrução fractal obteve um melhor desempenho. Isso era o mínimo que podíamos esperar, já que este é um exemplo onde o gráfico é realmente um fractal e, além disso, estamos considerando como perda, a informação num subintervalo no qual se verifica a propriedade de auto-similaridade.

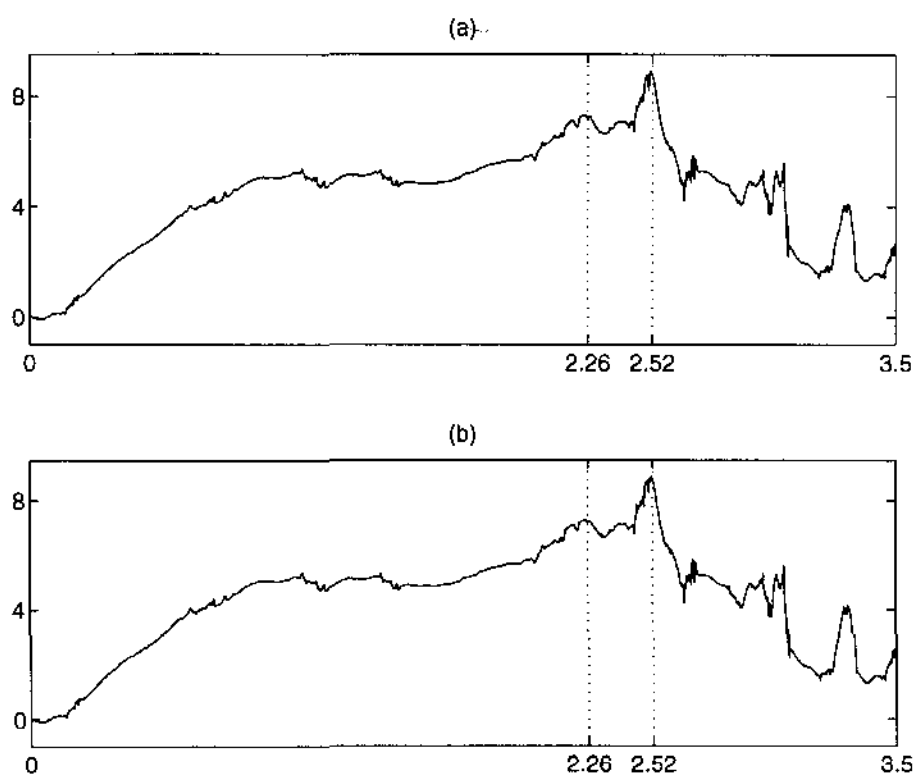


Figura V.9: (a) difratograma sintético de raios X. (b) gráfico reconstruído.

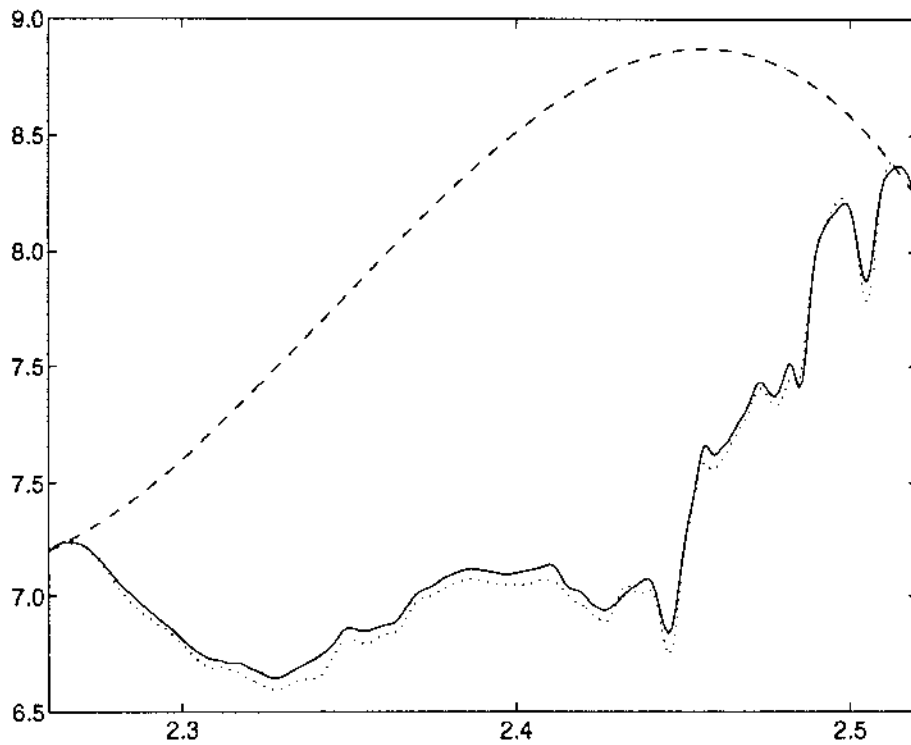


Figura V.10: Comparação da aproximação conseguida para o gráfico, no subintervalo ausente, através de *splines* cúbicas (curva tracejada) e da reconstrução fractal suavizada (traço sólido), com o gráfico real suavizado (curva pontilhada).

Difratograma sintético de raios X: Vamos novamente utilizar o gráfico da FIF do exemplo anterior, com 1024 pontos amostrados. Porém, no caso anterior havíamos retirado um dos subintervalos onde a propriedade de auto-similaridade se verifica exatamente. Ao invés disso, suponha que não temos amostras no intervalo $[2.0200, 2.3345]$, o que equivale a dizer que temos uma falha de 8.89% em relação ao intervalo amostrado. Esta situação pode ser vista na Figura V.11.

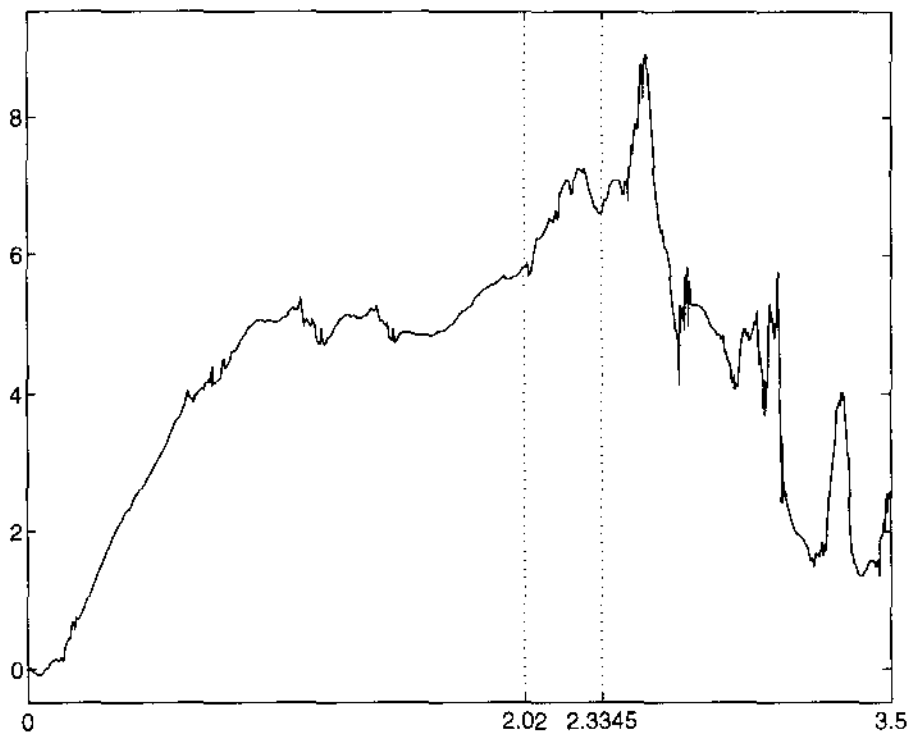


Figura V.11: Difratograma sintético de raios X. Em destaque, a porção a ser reconstruída.

Assumindo que os fatores de escala vertical tenham seus módulos limitados por 0.3, encontramos 26 pontos de corte descritos na Tabela V.2, com seus respectivos fatores de escala vertical e coeficientes de orientação.

Como também podemos ver na tabela, o subintervalo ausente foi o décimo terceiro. Precisamos estimar os parâmetros para esse subintervalo.

Os limitantes inferior e superior da dimensão fractal são facilmente calculados em 1.2201

n	x_n	y_n	s_n	o_n
0	0.0000	0.0000	-	-
1	0.2017	0.7133	-0.0551	1
2	0.8169	4.5810	0.0784	1
3	1.2100	4.8000	0.0901	1
4	1.4971	4.7644	0.0684	1
5	1.5142	4.8530	0.0029	1
6	1.5688	4.8457	0.0038	-1
7	1.6782	4.8865	-0.0073	1
8	1.7603	5.1481	-0.0077	1
9	1.8252	5.3849	-0.0068	1
10	1.9858	5.6684	0.0143	1
11	2.0029	5.7561	-0.0022	-1
12	2.0200	5.8149	0.0031	1
13	2.3345	6.6261	-	-
14	2.4063	7.0812	0.0217	1
15	2.4609	7.6138	-0.0918	1
16	2.5532	7.0336	0.2261	1
17	2.6934	5.6209	-0.2777	1
18	2.9600	5.0148	-0.1615	1
19	3.0181	5.1498	-0.1966	1
20	3.2402	1.8719	-0.2987	-1
21	3.3086	4.0144	0.0832	1
22	3.3975	1.4238	-0.2030	-1
23	3.4146	1.5760	0.0020	-1
24	3.4829	2.5387	-0.1067	-1
25	3.5000	2.5000	-0.0376	1

Tabela V.2: Parâmetros determinados na reconstrução do difratograma de raios X.

e 1.3814. Pelo algoritmo *Box Counting* temos as estimativas da dimensão fractal para cada um dos lados do gráfico:

n_i	D_-	D_+
1	1.0787	1.2319
2	1.2630	1.3736
3	1.1035	1.4671

Três estimativas encontram-se entre os limitantes calculados, 1.2319, 1.2630 e 1.3736.

Podemos calcular o módulo do fator de escala vertical para cada uma dessas estimativas da dimensão. Além disso, para cada caso, podemos testar as quatro combinações de sinais, do fator de escala vertical e do coeficiente de orientação, para decidir qual delas apresenta um erro menor. Essas informações estão resumidas na tabela que se segue, na qual a coluna *configuração* representa a melhor combinação de sinais de $|s_{13}|$ e $|o_{13}|$ conseguida. Nessa coluna, o valor 1 significa que ambos são positivos; 2, que apenas o segundo é positivo; 3, que apenas o primeiro é positivo e 4 significa que ambos são negativos.

D	$ s_{13} $	Configuração	Erro (%)
1.2319	0.0647	3	2.0181
1.2630	0.2408	3	2.0476
1.3736	0.9449	3	2.3836

Pode parecer estranho que o erro não varie muito quando variamos bruscamente o fator de escala vertical de 0.065 para 0.94, porém não há estranheza alguma aqui. O erro é calculado como a norma 1 da diferença entre os valores das ordenadas calculadas e das amostradas sobre a norma 1 da ordenadas amostradas. Claro que esta conta é feita apenas para os pontos fora do intervalo ausente. Assim, quando variamos o fator de escala vertical do intervalo ausente, tudo o que podemos medir são as perturbações que ele causa nos outros subintervalos. Essas perturbações dependem do comprimento do subintervalo ausente em relação ao do domínio todo e também de todos os outros fatores de escala vertical. Assim, quanto menor for a porcentagem não amostrada do gráfico mais difícil será mensurar o efeito da variação de s_n . Da mesma forma, quão menor os fatores de escala vertical dos outros subintervalos, mais serão atenuadas as variações da função nesse subintervalos. Portanto, novamente, menos perceberemos as alterações ocorridas por mudar o fator de escala vertical do subintervalo ausente.

Se, ao invés de usarmos a dimensão para calcular o fator de escala vertical, fizéssemos uma varredura para vários fatores de escala vertical obteríamos $s_{13} = 0.17$ e $o_{13} = -1$. Na Figura V.12 vemos como ficaria o trecho reconstruído para cada uma das quatro escolhas do fator de escala vertical. Podemos ver que a melhor aproximação foi para $s_{13} = 0.1700$. A Figura V.13 mostra como ficou a reconstrução do difratograma sintético de raios X.

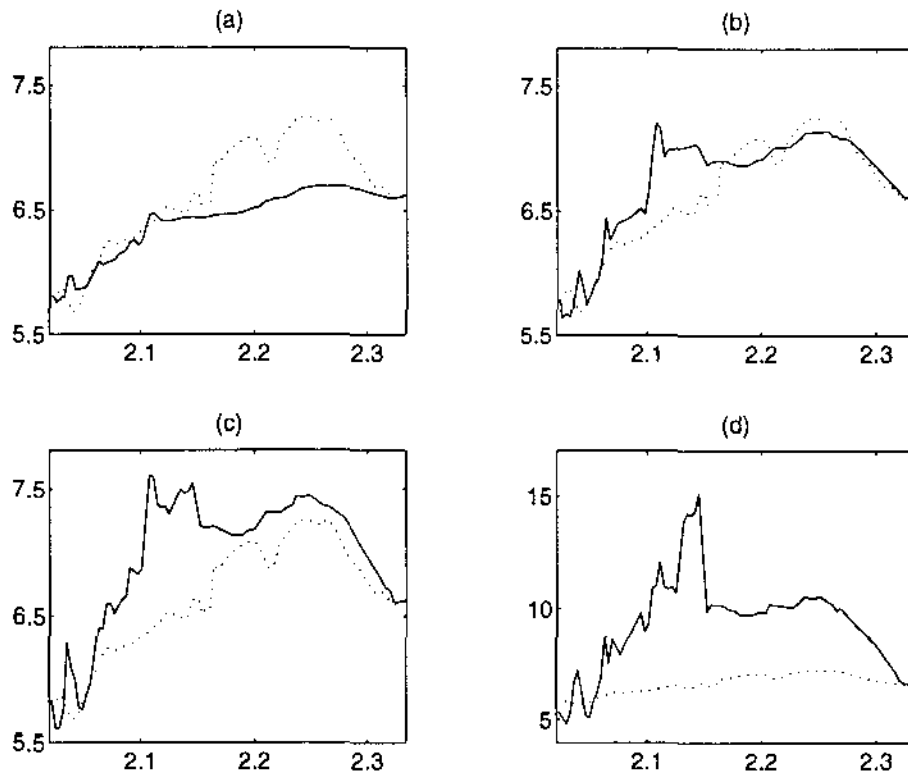


Figura V.12: Trecho reconstruído. O traço sólido representa a aproximação, enquanto que a curva pontilhada, o gráfico real. Em (a) $s_{13} = 0.0647$, em (b) $s_{13} = 0.1700$, em (c) $s_{13} = 0.2408$ e em (d) $s_{13} = 0.9448$.

Após suavizar o gráfico original, e também o obtido pela reconstrução fractal (para que possamos compará-lo), utilizamos as *splines* como forma alternativa de reconstrução. Comparando o resultado da reconstrução fractal ao obtido usando-se *splines* cúbicas, Figura V.14, vemos que a primeira ficou mais próxima de representar o comportamento real da função, senão de maneira quantitativa, pelo menos de maneira qualitativa.

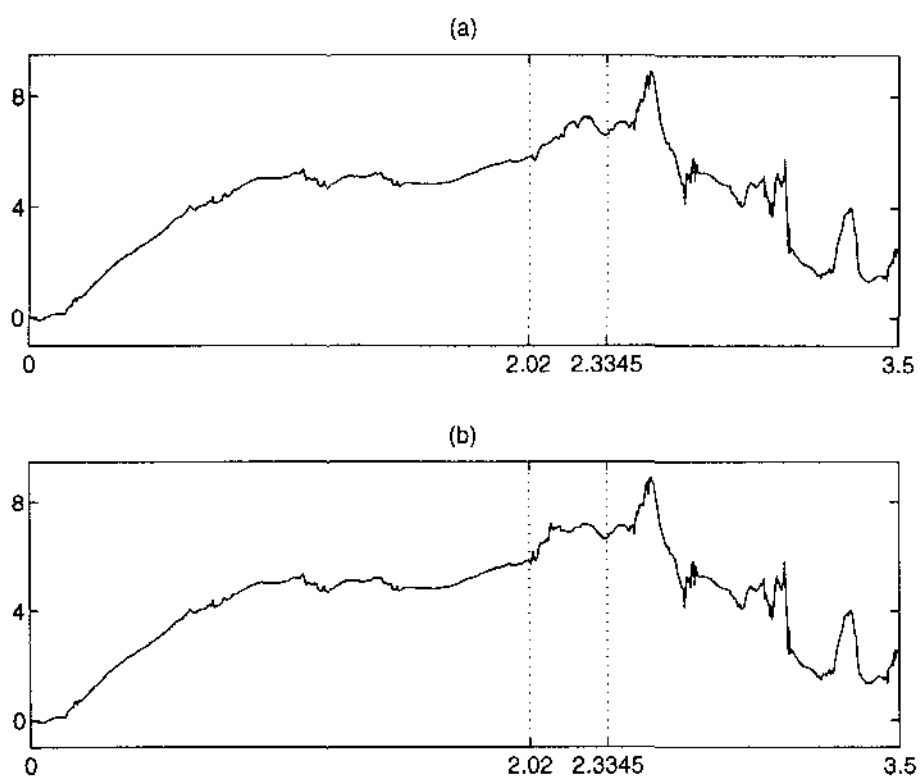


Figura V.13: (a) difratograma sintético de raios X. (b) gráfico reconstruído.

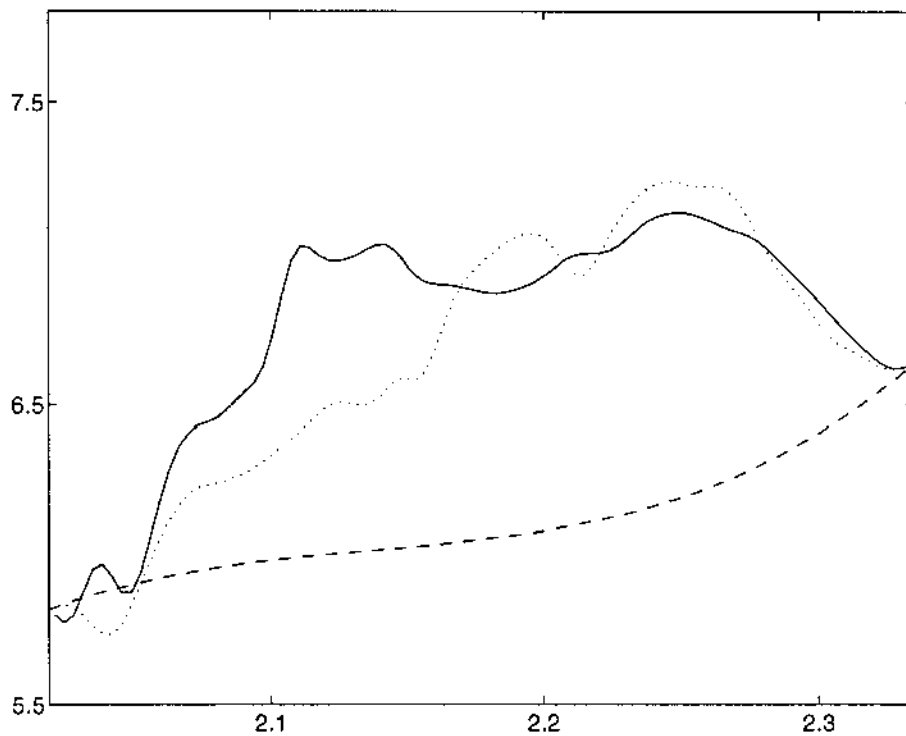


Figura V.14: Comparação da aproximação conseguida para o gráfico, no subintervalo ausente, através de *splines* cúbicas (curva tracejada) e da reconstrução fractal suavizada (traço sólido), com o gráfico original suavizado (curva pontilhada).

Direção do vento: Consideremos um exemplo meteorológico real. Esta é uma medição da direção do vento, em relação ao norte, contada no sentido horário, amostrada com 1156 pontos, numa frequência de 10 Hz, na Reserva Ducke, Floresta Amazônica, a aproximadamente 35 m do nível do mar. Suponhamos que haja uma falha de amostragem no intervalo $[0.352, 0.489]$, o que corresponde a 11.76% do tamanho da amostra. Na Figura V.15 podemos ver, em destaque, a porção do gráfico que estaremos supondo não lida.

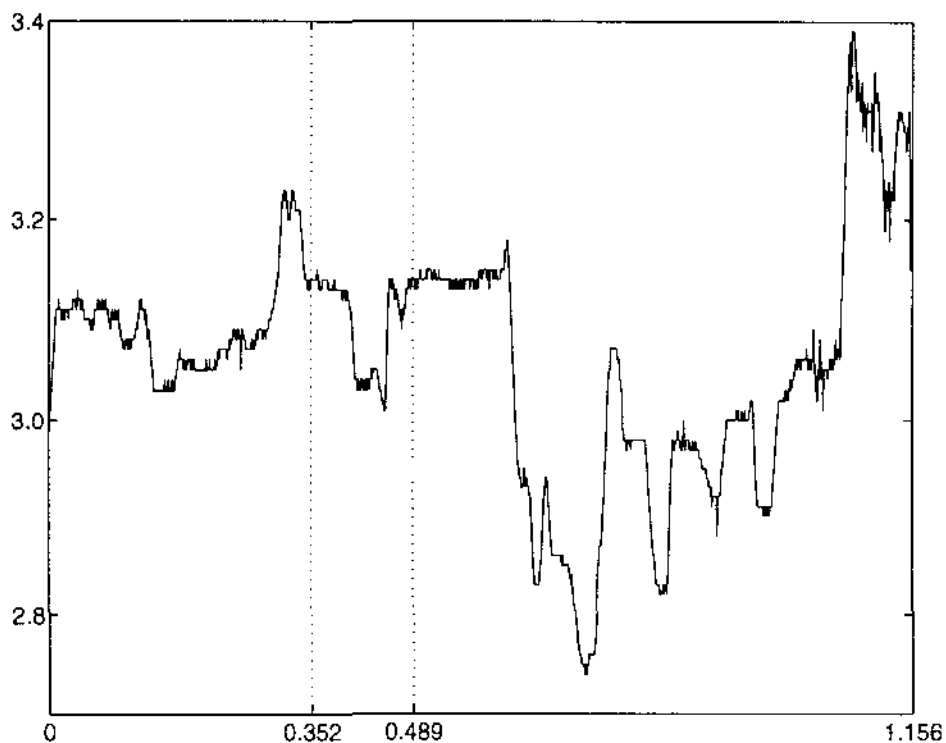


Figura V.15: Sinal meteorológico real da direção do vento. Em destaque, a região que estamos supondo não ter sido amostrada.

Seguindo nossa metodologia, devemos primeiro encontrar os pontos de corte. Para isso, impusemos a limitação nos módulos dos fatores de escala vertical em 0.2, para todos os subintervalos. Na Tabela V.3 podemos ver os subintervalos encontrados, com seus respectivos fatores de escala vertical e coeficientes de orientação (exceto para o oitavo subintervalo que é exatamente o que teremos de reconstruir).

n	x_n	y_n	s_n	o_n	n	x_n	y_n	s_n	o_n
0	0.001	2.990	-	-	20	0.820	2.820	0.037	-1
1	0.009	3.100	-0.022	-1	21	0.847	2.980	-0.146	1
2	0.246	3.080	0.145	1	22	0.854	2.970	-0.067	1
3	0.279	3.090	0.082	1	23	0.904	2.950	0.179	1
4	0.336	3.210	-0.158	1	24	0.923	3.010	-0.077	-1
5	0.345	3.140	0.033	1	25	0.928	3.000	-0.010	-1
6	0.350	3.130	-0.020	1	26	0.967	2.900	-0.150	-1
7	0.352	3.140	-0.012	1	27	0.980	3.020	0.044	-1
8	0.489	3.140	-	-	28	1.015	3.060	-0.037	1
9	0.504	3.140	0.025	-1	29	1.035	3.060	0.099	1
10	0.513	3.150	-0.016	1	30	1.043	3.050	0.114	-1
11	0.607	3.140	0.042	-1	31	1.048	3.060	0.030	-1
12	0.613	3.170	0.037	1	32	1.057	3.060	-0.049	1
13	0.633	2.940	-0.117	-1	33	1.062	3.120	0.118	1
14	0.662	2.860	0.135	1	34	1.072	3.360	0.089	-1
15	0.688	2.860	-0.197	-1	35	1.121	3.230	-0.198	1
16	0.725	2.760	0.098	1	36	1.142	3.300	0.164	-1
17	0.763	3.060	-0.200	1	37	1.147	3.290	-0.010	-1
18	0.769	3.030	-0.025	-1	38	1.152	3.310	0.059	-1
19	0.812	2.860	-0.179	1	39	1.156	3.240	0.200	1

Tabela V.3: Parâmetros determinados para o gráfico de direção do vento.

Calculando os limitantes inferior e superior, temos que a dimensão deve estar compreendida entre 1.3250 e 1.4580. Estimando a dimensão fractal para cada um dos lados, temos as seguintes aproximações:

n_i	D_-	D_+
1	1.2169	1.2937
2	1.2513	1.4150
3	1.3505	1.2824

Podemos observar que, dentre todas as estimativas, apenas duas se encaixam no intervalo aceitável para a dimensão. Se D fosse 1.3505, então o módulo do fator de escala vertical seria 0.1857, contudo, se D fosse 1.4150, obteríamos 0.6678. Na Figura V.16 podemos ver como ficaria o trecho reconstruído com cada um desses valores, fazendo a escolha dos sinais de s_n e o_n que resultou o menor erro.

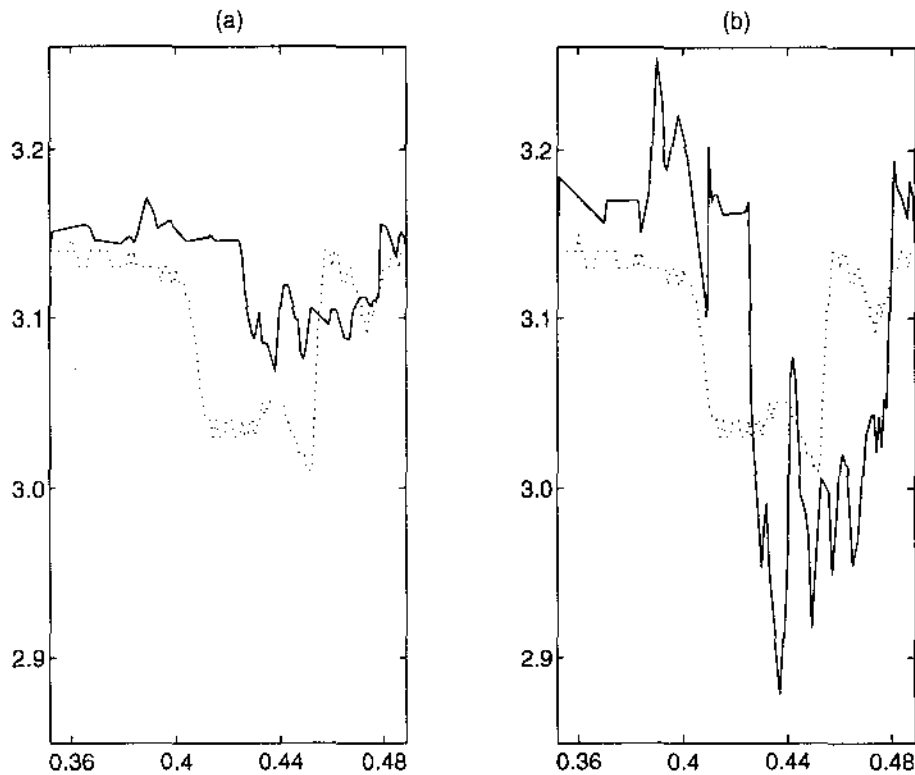


Figura V.16: (a) gráfico reconstruído usando $|s_8| = 0.1857$. (b) gráfico reconstruído usando $|s_8| = 0.6678$. Em (a) e (b) a curva pontilhada é o gráfico real.

Como antes, fizemos vários ensaios para diferentes valores do módulo do fator de escala vertical e para cada um deles computamos a norma 1 da diferença entre o gráfico real e o reconstruído, usando apenas a parte suposta conhecida do gráfico. Para cada um dos valores testados para $|s_8|$ consideramos as quatro combinações dos sinais do módulo do fator de escala vertical e do coeficiente de orientação. A melhor aproximação foi conseguida com $s_8 = 0.2850$ e $o_8 = 1$. Podemos ver, na Figura V.17, como ficaria o trecho reconstruído em comparação ao real. Para $s_8 = 0.2850$, a dimensão fractal fica $D = 1.3640$, contra a estimativa de 1.3541 do gráfico real (completo). Na Figura V.18, está o resultado final da reconstrução.

Como nos exemplos anteriores, suavizamos o gráfico real e o obtido pela reconstrução fractal. Comparando-os com o resultado da reconstrução por *splines* cúbicas, podemos ver

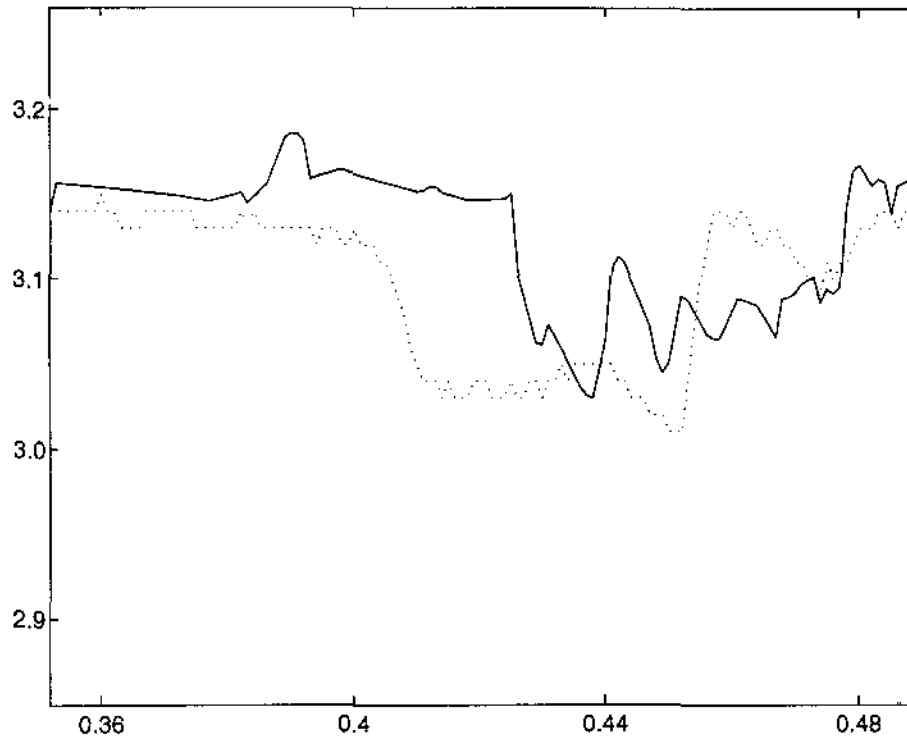


Figura V.17: O traço sólido representa o gráfico reconstruído usando $s_8 = 0.2850$ e $\sigma_8 = 1$ e a curva pontilhada, o gráfico real.

que, também nessa situação, a reconstrução fractal ficou bem mais próxima de representar o comportamento do gráfico original, Figura V.19.

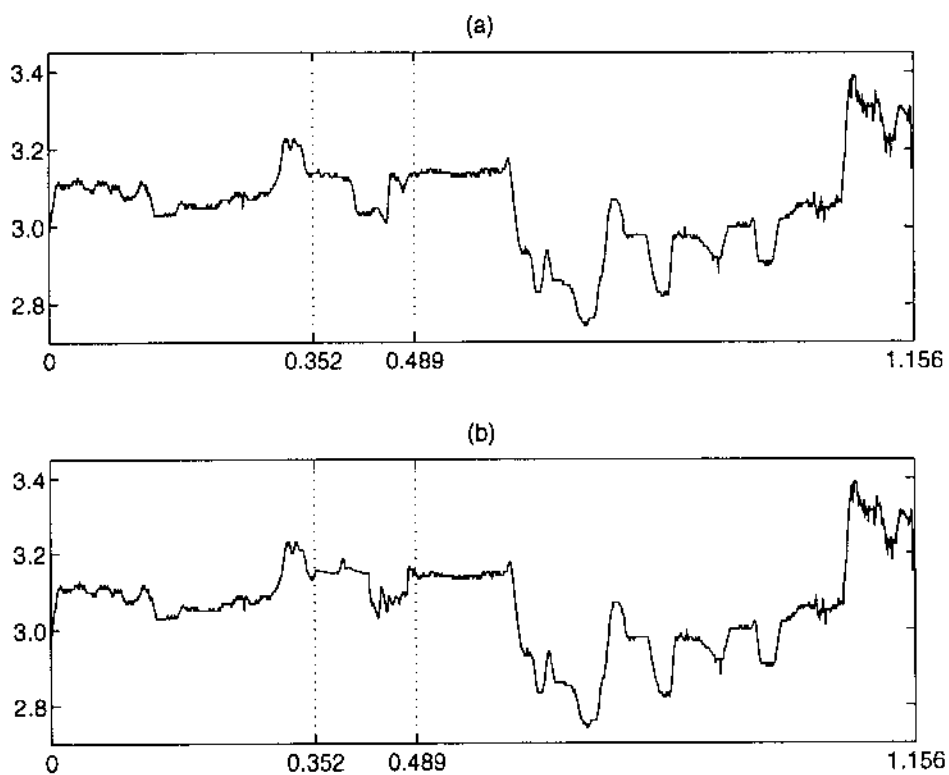


Figura V.18: (a) gráfico original da direção do vento. (b) gráfico reconstruído.

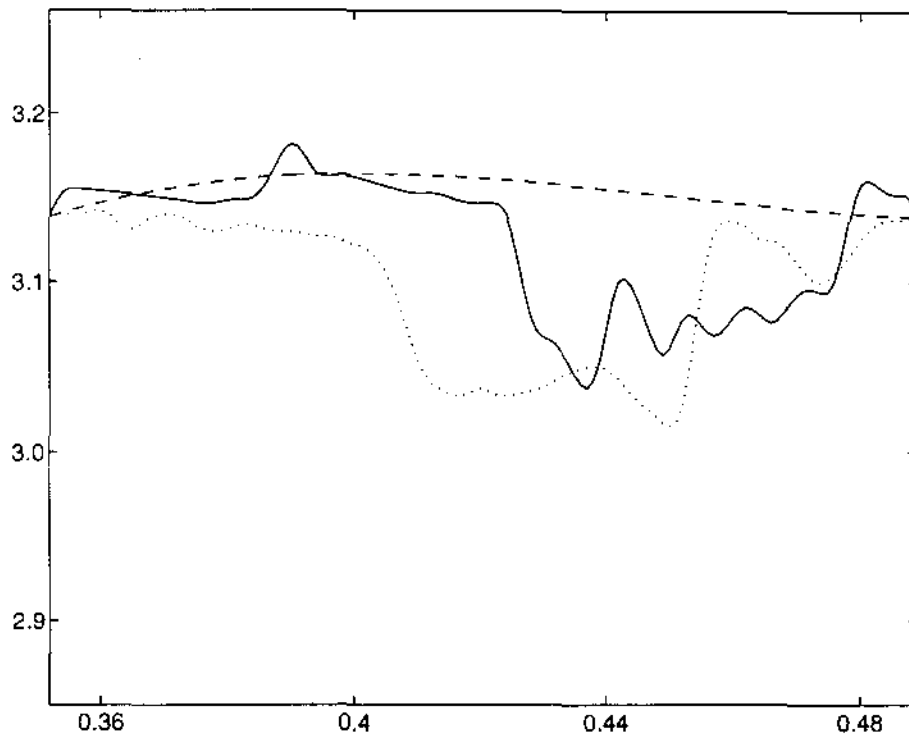


Figura V.19: Comparação da aproximação conseguida para o gráfico, no subintervalo ausente, através de *splines* cúbicas (curva tracejada) e da reconstrução fractal suavizada (traço sólido), com o gráfico real suavizado (curva pontilhada).

Velocidade vertical do vento: Este é mais um sinal meteorológico real, também medido na Reserva Ducke, Floresta Amazônica, pelo INPE. Consiste de um conjunto de 1156 amostras da velocidade vertical do vento, a uma altura de 30 m do solo (abaixo da copa das árvores). Tal qual o sinal anterior, este também teve suas medições feitas numa frequência de 10 Hz. Na Figura V.20 vemos o sinal amostrado e, em destaque, a porção que suporemos não ter sido lida, $[0.180, 0.320]$, correspondendo a 12.11% do gráfico completo.

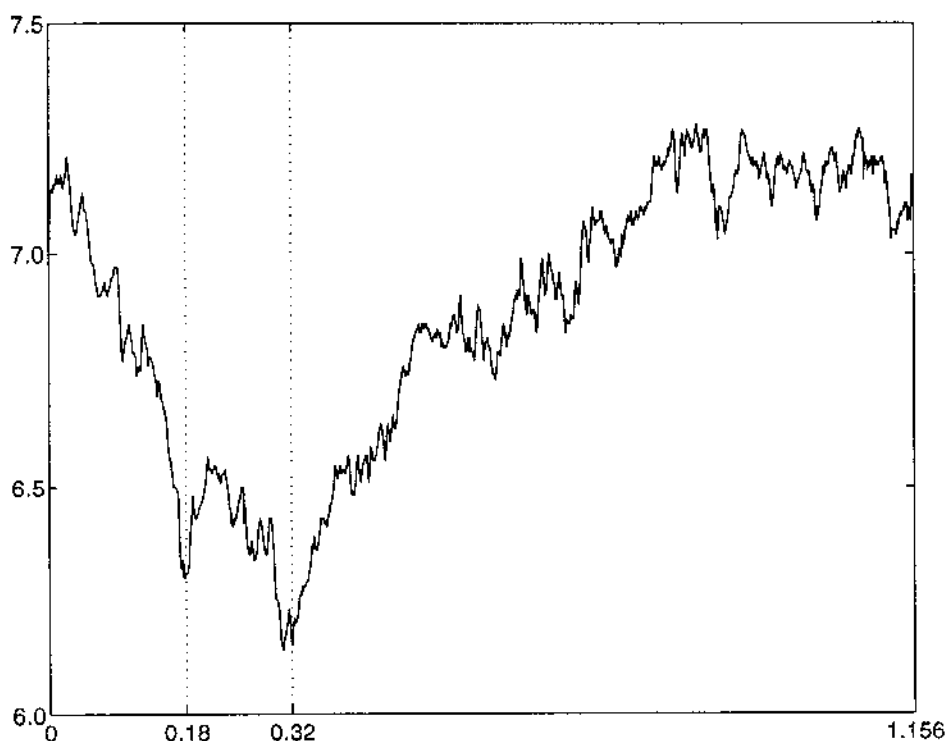


Figura V.20: Sinal meteorológico da velocidade vertical do vento. Em destaque, o subintervalo no qual suporemos que não houve leitura.

Os pontos foram selecionados impondo-se uma limitação de 0.2, para o máximo dos módulos dos fatores de escala vertical. Encontramos 41 subintervalos, mostrados na Tabela V.4, juntamente com seus respectivos fatores de escala vertical e coeficientes de orientação. O subintervalo faltante foi determinado como sendo o décimo segundo.

Com o cálculo dos limitantes para a dimensão fractal, obtivemos que $1.3070 < D <$

n	x_n	y_n	s_n	o_n	n	x_n	y_n	s_n	o_n
0	0.001	7.030	-	-	21	0.606	6.810	-0.034	1
1	0.006	7.150	-0.083	1	22	0.641	6.870	-0.145	-1
2	0.025	7.200	0.048	-1	23	0.659	6.930	0.091	-1
3	0.051	7.070	0.125	1	24	0.714	6.980	0.142	-1
4	0.064	6.910	0.033	1	25	0.727	7.030	-0.081	1
5	0.076	6.920	-0.024	-1	26	0.737	7.070	-0.063	1
6	0.085	6.960	0.016	1	27	0.771	7.020	0.070	-1
7	0.144	6.730	0.156	1	28	0.805	7.110	-0.048	1
8	0.156	6.610	-0.033	-1	29	0.834	7.230	-0.077	1
9	0.173	6.320	-0.111	-1	30	0.848	7.240	0.116	-1
10	0.179	6.300	-0.033	1	31	0.855	7.250	0.040	-1
11	0.180	6.310	0.000	1	32	0.923	7.180	0.192	-1
12	0.320	6.150	-	-	33	0.956	7.190	-0.096	1
13	0.329	6.230	-0.036	1	34	0.974	7.190	0.098	-1
14	0.334	6.280	-0.022	1	35	1.049	7.230	0.162	-1
15	0.399	6.540	-0.098	-1	36	1.057	7.160	-0.037	-1
16	0.531	6.800	-0.136	-1	37	1.094	7.200	-0.089	-1
17	0.537	6.840	-0.018	-1	38	1.099	7.190	0.020	1
18	0.549	6.910	0.062	-1	39	1.147	7.110	0.125	-1
19	0.568	6.770	0.074	1	40	1.152	7.070	0.035	1
20	0.599	6.750	-0.134	1	41	1.156	7.140	-0.090	1

Tabela V.4: Parâmetros determinados para o gráfico de velocidade vertical do vento.

1.4470. As estimativas para a dimensão, para cada um dos lados do gráfico (à esquerda e à direita do subintervalo ausente), são, para os diferentes valores de n_i :

n_i	D_-	D_+
1	1.0276	1.3217
2	1.0708	1.4037
3	1.3219	1.5850

Novamente, das seis estimativas, apenas três se encaixam nos limitantes calculados. Se assumíssemos que $D = 1.3217$, obteríamos que $|s_{12}| = 0.1032$, se assumíssemos que $D = 1.3219$, teríamos $|s_{12}| = 0.1046$, por fim, se assumirmos que $D = 1.4037$, $|s_{12}|$ seria 0.6810. A Figura V.21 mostra como ficaria o trecho reconstruído, utilizando os últimos dois valores (para o primeiro, o resultado é muito similar ao conseguido com o segundo).

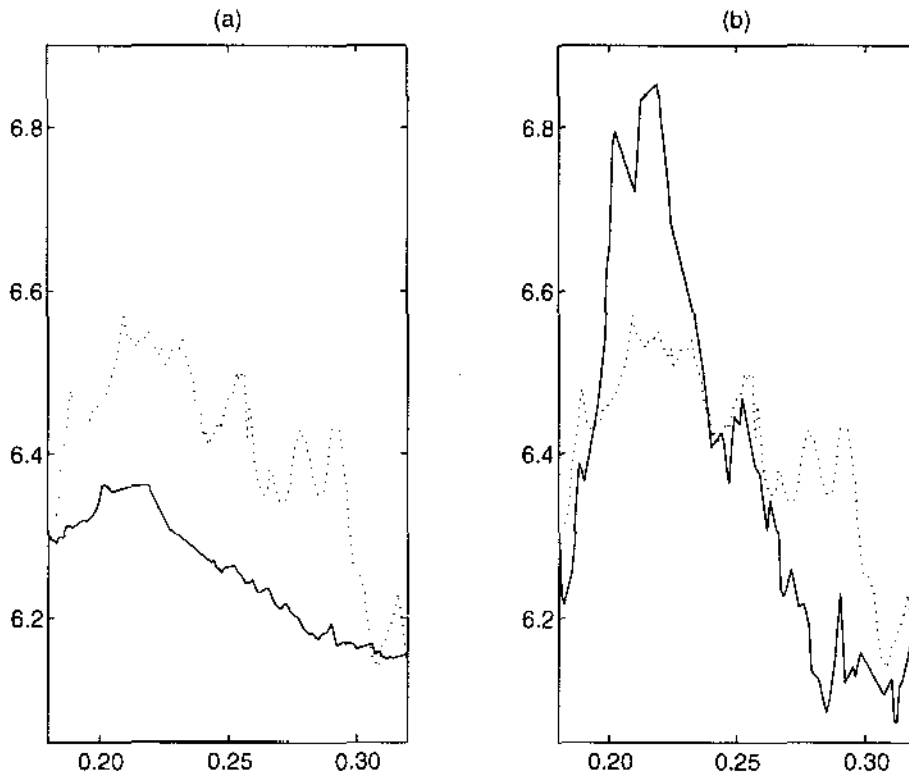


Figura V.21: (a) gráfico reconstruído usando $|s_{12}| = 0.1046$. (b) gráfico reconstruído usando $|s_{12}| = 0.6810$. Em (a) e (b) a curva pontilhada é o gráfico real.

Fizemos também vários testes, tomando o módulo do fator de escala vertical variando de 0.1 a 0.9 com incrementos de 0.01. Para cada um desses valores, testamos as quatro possibilidades de combinações de sinais de s_{12} e o_{12} . A melhor configuração que conseguimos foi $s_{12} = -0.29$ e $o_{12} = 1$. Na Figura V.22, vemos como ficou o trecho reconstruído usando esses valores e, na Figura V.23, o resultado final da reconstrução fractal.

Comparando, como nos exemplos anteriores, podemos ver na Figura V.24, como fica o resultado da reconstrução fractal e da reconstrução por *splines* cúbicas.

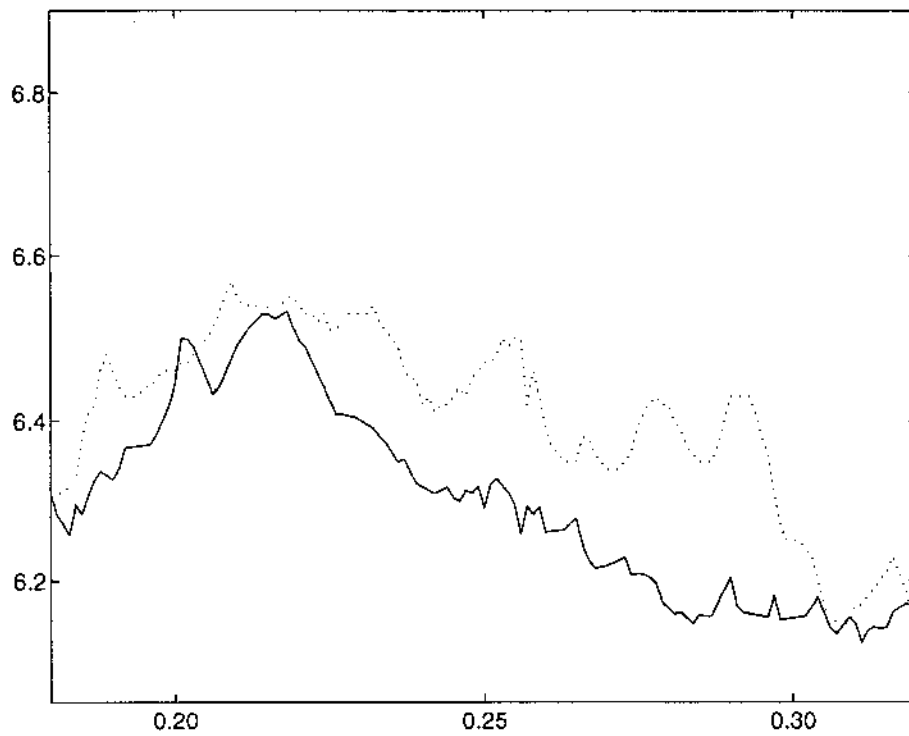


Figura V.22: O traço sólido representa o gráfico reconstruído usando $s_{12} = -0.29$ e $\sigma_{12} = 1$ e a curva pontilhada, o gráfico real.

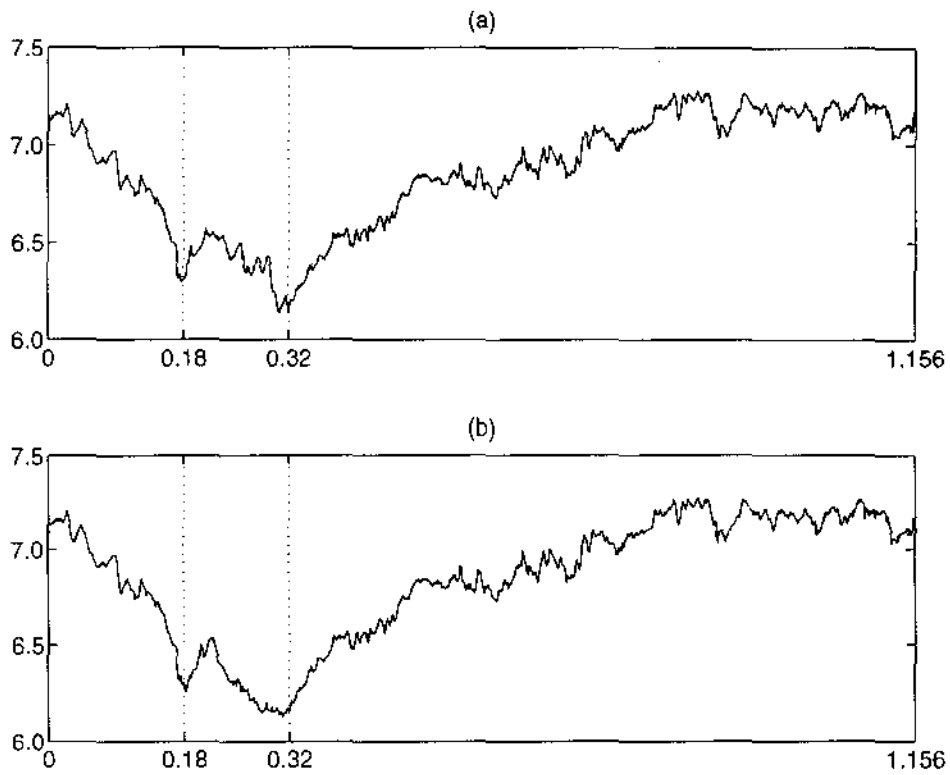


Figura V.23: (a) gráfico original da velocidade vertical do vento. (b) aproximação conseguida através da reconstrução fractal.

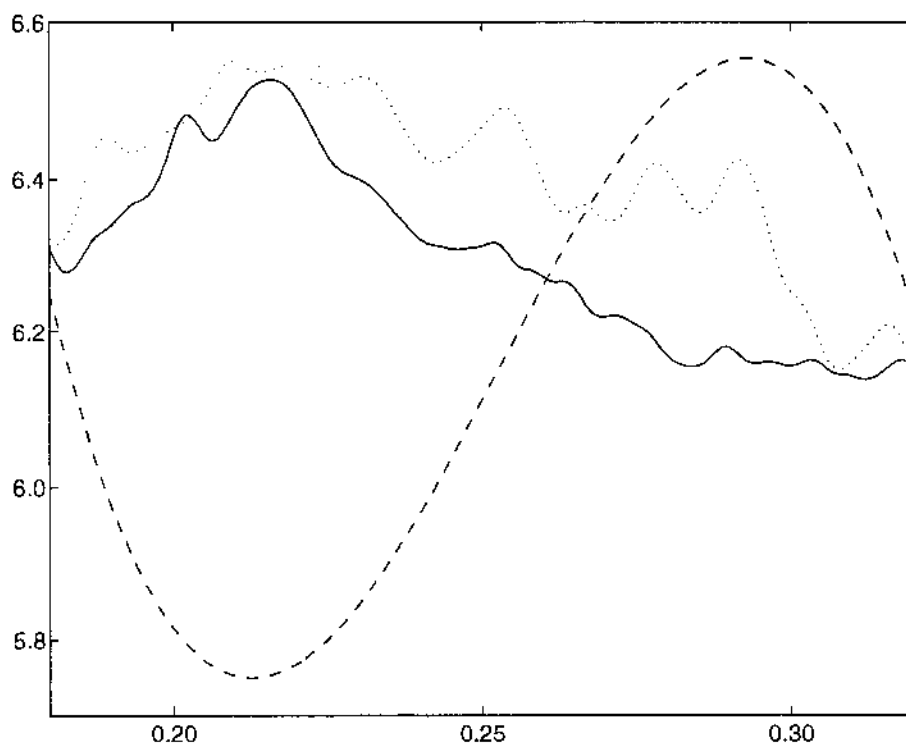


Figura V.24: Comparação da aproximação conseguida para o gráfico, no subintervalo ausente, através de *splines* cúbicas (curva tracejada) e da reconstrução fractal suavizada (traço sólido), com o gráfico original suavizado (curva pontilhada).

Com o primeiro exemplo, vimos que, no caso ideal, a reconstrução fractal comportou-se muito bem. Ou seja, se o gráfico for auto-similar e, além disso, estivermos tentando reconstruir um subintervalo onde a propriedade de auto-similaridade se verifica, então o resultado é muito bom, tanto qualitativa quanto quantitativamente. Nos outros exemplos, podemos ver que, quando o gráfico é auto-similar, mas o subintervalo a reconstruir não é um dos subintervalos no qual a propriedade se verifica, ou quando nem mesmo o gráfico é auto-similar no sentido estrito, os resultados ainda são bons. Nesses três últimos exemplos, a reconstrução fractal conseguiu recuperar a informação, a um nível qualitativo e, em certa medida, a um nível quantitativo também. No quarto caso, por exemplo, a amplitude do gráfico, na porção ausente, foi recuperada na reconstrução. Desta forma, a reconstrução fractal mostrou-se uma boa ferramenta para casos onde a falta de informação num dado subintervalo prejudica uma análise qualitativa do sinal, podendo ser usada (com ressalvas) para obter algum tipo de informação quantitativa.

Capítulo VI

Conclusão

Essa tese foi idealizada a partir de um problema real, proposto por pesquisadores do Instituto Nacional de Pesquisas Espaciais, INPE. O objetivo era reconstruir um sinal amostrado com falhas intermediárias. Para tal, usamos a Interpolação Fractal [1] como ponto de partida e desenvolvemos a técnica de Reconstrução Fractal de Sinais. A idéia original, era que, se pudéssemos escolher alguns pontos especiais dentre os amostrados, e com eles construir uma função de interpolação fractal que aproximasse de maneira razoável a função real, poderíamos então, utilizar alguma informação extra para reconstruir a falha de leitura. Essa informação adicional seria a dimensão fractal estimada a partir dos dados. A maneira de conectar a dimensão ao intervalo ausente poderia ser feita através da equação para a dimensão de uma função de interpolação fractal em função dos dados.

A princípio, tivemos alguns obstáculos relacionados à manipulação de funções fractais. Não havia pacotes eficientes na utilização da interpolação fractal. Optamos por fazer todas as implementações dentro do ambiente MATLAB. Nesta decisão pesaram sua extensa utilização, sua grande quantidade de bibliotecas já existentes, seus recursos gráficos e de manipulação matricial já implementados e largamente testados além de sua portabilidade e facilidade de implementação. Acabamos por desenvolver uma extensa biblioteca de rotinas relacionadas à interpolação, aproximação e reconstrução fractal de sinais. Uma desvantagem de programar em MATLAB é o fato das rotinas serem interpretadas e não compiladas, o que prejudica muito o tempo de execução.

O primeiro degrau foi vencido quando todas as rotinas necessárias à interpolação fractal

já tinham sido implementadas. Desse momento em diante, pudemos testar rapidamente nossas conjecturas e ganhar experiência de como as peças se juntavam por trás da teoria. Foi uma etapa importantíssima e, pelo fato de ter sido transposta logo no início do trabalho, permitiu-nos concluir esta tese em tempo hábil.

Para quem está interessado em reconstruir funções lidas com falha, usando interpolação fractal, a primeira pergunta a ser feita é: como aproximar funções lidas, sem falha alguma, usando FIF's? A resposta que encontramos foi, em primeiro lugar, selecionar alguns pontos, dentre todos os amostrados, de forma a definir subintervalos que satisfizessem a propriedade de auto-similaridade da melhor maneira possível; e em segundo lugar, utilizar os pontos remanescentes para estimar os parâmetros livres, de maneira a conseguir uma aproximação que fosse a mais fiel possível. Assim, a questão de como escolher os pontos de corte surgiu naturalmente. Resolver esse passo, significou desenvolver o algoritmo de seleção de pontos de corte e a Aproximação Fractal de Funções. Poderíamos achar que isto foi apenas uma etapa para a reconstrução de sinais, no entanto, essa foi a parte mais importante desse trabalho. Uma dificuldade encontrada foi que o algoritmo de seleção de pontos é um tanto quanto oneroso demais para a estrutura interpretada da linguagem do MATLAB. Os tempos de execução para alguns dos casos considerados foram, em grande parte, superiores a três horas. Esse problema foi eliminado implementando-se o algoritmo em C. Com isso, os tempos de execução caíram para algo em torno de dois a três minutos, ficando, para o caso mais lento, em dez minutos (eram gastas sete horas com a versão implementada no MATLAB!). Com a aproximação fractal desenvolvida, grande parte do trabalho necessário à reconstrução já estava feito.

Ressaltamos também que a aproximação fractal de sinais é importante por si mesma. Obtivemos uma forma de representar funções, muitas vezes irregulares e oscilantes a ponto de impedir o uso de técnicas convencionais. Outra possível aplicação dessa técnica seria, por exemplo, o cálculo da dimensão fractal de uma maneira alternativa ao uso do algoritmo de *Box Counting*. Podíamos aproximar uma função (impondo a limitação apropriada aos fatores de escala vertical) e depois calcular a dimensão fractal, da FIF obtida, usando a fórmula exata, vista no Capítulo III. Se a aproximação for realmente boa, este método seria melhor, pois eliminaria as oscilações do *Box Counting*.

Da aproximação à reconstrução, as mudanças que ocorreram na abordagem do pro-

blema foram que os extremos do subintervalo ausente deveriam obrigatoriamente estar presentes no conjunto dos pontos de corte e que precisávamos de outra forma de estimar os parâmetros associados a esse subintervalo. Nossa primeira idéia foi utilizar uma estimativa da dimensão fractal para calcular, através de (III.2), o módulo do fator de escala vertical. Em seguida, escolher, dentre as quatro opções de sinais do fator de escala vertical e do coeficiente de orientação, a que resultasse em uma menor norma da diferença entre as funções amostrada e a aproximada. Porém, na prática, isso mostrou-se um tanto quanto difícil, dado a alta sensibilidade do fator de escala vertical em relação à dimensão. Assim, se não pudermos confiar na estimativa da dimensão fractal, também não poderemos confiar na qualidade da estimativa dos parâmetros. Para transpor essa dificuldade, optamos por fazer uma busca dentre vários valores discretos do fator de escala e escolher aquele que resultou um menor erro. Apesar de mais lento, isso mostrou-se eficiente. Posteriormente, o problema da velocidade foi eliminado implementando-se, também esta etapa, em C. Isso fez o tempo computacional cair de dezenas de minutos para a casa dos segundos.

Por fim, gostaríamos de ressaltar que a contribuição desse trabalho foi o desenvolvimento das técnicas de aproximação e reconstrução fractal de sinais, as quais têm como núcleo o algoritmo de seleção de pontos de corte. Ressaltamos que não temos conhecimento de técnica alguma que se disponham a resolver o problema de reconstrução de sinais amostrados com falha intermediária. Outro ponto que merece destaque é que as duas técnicas apresentadas mostram-se mais eficientes quanto mais forte se realizar a propriedade de auto-similaridade do sinal. Essa característica está presente em muitos sinais reais, por exemplos os dados meteorológicos vistos aqui.

Acreditamos que um estudo mais detalhado sobre maneiras de estimar a dimensão fractal, para sinais amostrados discretamente e, possivelmente, com falhas, deva ser feito, podendo reverter em benefícios no momento de estimar os parâmetros livres, e assim para reconstrução como um todo. Além disso, deve-se investir em formas alternativas de estimar os parâmetros, usando outros tipos de informação que não a dimensão fractal.

Apêndice A

Demonstração dos Resultados

Demonstrações do Capítulo II

Proposição II.1: *O par $(\mathcal{H}(M), h_d)$ é um espaço métrico.*

Prova: Para provarmos que h_d define uma métrica em $\mathcal{H}(M)$, precisamos verificar as seguintes propriedades:

- $0 \leq h_d(A, B) < \infty$, para todo $A, B \in \mathcal{H}(M)$,
- $h_d(A, B) = 0$ se e somente se $A = B$,
- $h_d(A, B) = h_d(B, A)$, para todo $A, B \in \mathcal{H}(M)$ (simetria) e
- $h_d(A, C) \leq h_d(A, B) + h_d(B, C)$, para todo $A, B, C \in \mathcal{H}(M)$ (desigualdade triangular).

A primeira desigualdade é facilmente verificada. Como $d(x, y) \geq 0$ para todo $x, y \in M$,

$$0 \leq \min_{y \in B} d(x, y) \leq \max_{x \in A} \min_{y \in B} d(x, y) = D(A, B).$$

Por simetria, $D(B, A) \geq 0$ e assim $h_d(A, B) \geq 0$.

A finitude de $h_d(A, B)$ decorre do fato de A e B serem compactos. Considere a função $f : A \rightarrow \mathbb{R}$, dada por $f(x) \equiv \min_{y \in B} d(x, y)$. Pelo fato de B ser compacto e $d(x, y)$, para um x fixo, ser uma função contínua, podemos dizer que, pelo Teorema de Weierstrass,

exite um ponto $\bar{y} \in B$ tal que $f(x) = d(x, \bar{y})$. Vejamos que f é uma função contínua. Para $x, z \in A$, sejam $\bar{y} = \arg \min_{y \in B} d(x, y)$ e $\bar{w} = \arg \min_{w \in B} d(z, w)$, então temos que

$$|f(x) - f(z)| = |d(x, \bar{y}) - d(z, \bar{w})|. \quad (1)$$

Se $d(x, \bar{y}) \geq d(z, \bar{w})$, então (1) pode ser majorado por

$$|d(x, \bar{y}) - d(z, \bar{w})| \leq d(x, \bar{w}) - d(z, \bar{w}) \leq d(x, z).$$

Se, por outro lado, $d(x, \bar{y}) < d(z, \bar{w})$, então (1) pode ser majorado por

$$|d(x, \bar{y}) - d(z, \bar{w})| \leq d(z, \bar{y}) - d(x, \bar{y}) \leq d(x, z).$$

Portanto, dado $\epsilon > 0$, escolhendo $\delta = \epsilon$, se $d(x, z) < \delta$, temos que $|f(x) - f(z)| < \epsilon$. Como a função f também está definida sobre um compacto, novamente pelo teorema de Weierstrass, podemos afirmar que seu máximo é assumido para algum ponto $\bar{x} \in A$. Concluimos então que

$$D(A, B) = \max_{x \in A} \min_{y \in B} d(x, y) = d(\bar{x}, \bar{y}) < \infty,$$

e portanto, $h_d(A, B) < \infty$.

Considere agora o caso de $h_d(A, B)$ ser zero. Para que isto ocorra, devemos ter que $D(A, B)$ e $D(B, A)$ se anulem. Se $D(A, B)$ for zero, então $\min_{y \in B} d(x, y)$ também o será, para todo $x \in A$. Porém, se $\min_{y \in B} d(x, y)$ for zero então $x \in B$, pois B é compacto. Isto significa que $A \subset B$. Considerando que $D(B, A)$ é zero, obtemos que $B \subset A$. Portanto $A = B$.

Assumindo agora que $A = B$, temos que $\min_{y \in B} d(x, y)$ é zero, para qualquer que seja $x \in A$ (basta tomar $y = x$). Logo $D(A, B)$ é zero. Da mesma forma o será $D(B, A)$. Portanto $h_d(A, B)$ será zero, o que prova a segunda propriedade.

A propriedade de simetria é facilmente verificada a partir da própria definição de h_d .

$$h_d(A, B) = \max\{D(A, B), D(B, A)\} = \max\{D(B, A), D(A, B)\} = h_d(B, A).$$

Falta-nos provar a desigualdade triangular. Veja que, para um $x \in A$ fixo,

$$\begin{aligned} \min_{z \in C} d(x, z) &\leq \min_{z \in C} \{d(x, y) + d(y, z)\}, \text{ para todo } y \in B \\ &= d(x, y) + \min_{z \in C} d(y, z), \text{ para todo } y \in B, \end{aligned}$$

então,

$$\begin{aligned} \min_{z \in C} d(x, z) &\leq \min_{y \in B} d(x, y) + \max_{y \in B} \min_{z \in C} d(y, z) \\ &= \min_{y \in B} d(x, y) + D(B, C). \end{aligned}$$

Logo, tomando o máximo para $x \in A$, temos que

$$D(A, C) \leq D(A, B) + D(B, C).$$

Podemos obter uma desigualdade análoga para $D(C, A)$. Concluimos assim que

$$h_d(A, C) \leq h_d(A, B) + h_d(B, A).$$

Com isto, acabamos de provar que $(\mathcal{H}(M), h_d)$ formam um espaço métrico. Perceba que, apesar de não termos explicitado, a hipótese dos conjuntos pertencentes a $\mathcal{H}(M)$ serem não vazios é crucial. Sem ela, as frases “ $x \in A$ ”, “ $\min_{y \in B}$ ”, etc, não teriam sentido algum. \square

Proposição II.2: W é uma contração em $(\mathcal{H}(M), h_d)$ com fator de contração $\alpha \equiv \max_{n \in \mathbb{N}} \alpha_n$.

Prova: Sejam $A, B \in \mathcal{H}(M)$. Então

$$h_d(W(A), W(B)) = \max\{D(W(A), W(B)), D(W(B), W(A))\}. \quad (2)$$

Agora,

$$\begin{aligned} D(W(A), W(B)) &= \max_{u \in W(A)} \min_{v \in W(B)} d(u, v) = \max_{\substack{x \in A \\ n \in \mathbb{N}}} \min_{\substack{y \in B \\ m \in \mathbb{N}}} d(w_n(x), w_m(y)) \\ &\leq \max_{\substack{x \in A \\ n \in \mathbb{N}}} \min_{y \in B} d(w_n(x), w_n(y)) \leq \max_{\substack{x \in A \\ n \in \mathbb{N}}} \min_{y \in B} \{\alpha_n d(x, y)\} \\ &= \alpha \max_{x \in A} \min_{y \in B} d(x, y) = \alpha D(A, B). \end{aligned}$$

Portanto, de (2), vem que

$$h_d(W(A), W(B)) \leq \max\{\alpha D(A, B), \alpha D(B, A)\} \leq \alpha h_d(A, B). \quad \square$$

Teorema II.1[Teorema da Colagem]: *Seja (M, d) um espaço métrico completo. Sejam $\{(M, d); w_n, n \in \mathbf{N}\}$ um IFS, com fator de contração α , e $\mathcal{L} \in \mathcal{H}(M)$ tal que*

$$h_d\left(\mathcal{L}, \bigcup_{n \in \mathbf{N}} w_n(\mathcal{L})\right) \leq \varepsilon,$$

para algum $\varepsilon > 0$. Então

$$h_d(\mathcal{L}, \mathcal{A}) \leq \frac{\varepsilon}{1 - \alpha},$$

onde \mathcal{A} é o atrator do IFS.

Prova: A demonstração deste resultado é muito simples e, por esse motivo, poderíamos pensar que seria mais adequado chamá-lo de *proposição* do que de *teorema*. No entanto, pela importância que representa e pela interpretação que recebe no contexto dos fractais, a classificação *teorema* é justa.

Como \mathcal{A} é atrator do IFS, por definição, $W(\mathcal{A}) = \mathcal{A}$. Lembrando da definição de W , (II.1), e usando a Proposição II.2,

$$h_d(W(\mathcal{L}), \mathcal{A}) = h_d(W(\mathcal{L}), W(\mathcal{A})) \leq \alpha h_d(\mathcal{L}, \mathcal{A}).$$

Assim,

$$\begin{aligned} h_d(\mathcal{L}, \mathcal{A}) &\leq h_d(\mathcal{L}, W(\mathcal{L})) + h_d(W(\mathcal{L}), \mathcal{A}) \\ &\leq \varepsilon + \alpha h_d(\mathcal{L}, \mathcal{A}) \end{aligned}$$

Portanto,

$$(1 - \alpha) h_d(\mathcal{L}, \mathcal{A}) \leq \varepsilon.$$

Como α é o fator de contração de W , $0 \leq \alpha < 1$. Dividindo ambos os lados por $(1 - \alpha)$ obtemos o resultado desejado. \square

Teorema II.2: *Existe uma métrica \tilde{d} , equivalente à métrica euclidiana, tal que w_n , como definida em (II.4) e (II.8) (ou (II.9)), é contração em $(\mathbb{R}^2, \tilde{d})$ se $|s_n| < 1$, para todo $n \in \mathbb{N}$.*

Prova: No contexto desta demonstração, todos os máximos e mínimos são para $n \in \mathbb{N}$. Considere $\tilde{d} : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}$ definida por

$$\tilde{d}((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + \theta |y_1 - y_2|,$$

para algum θ positivo a ser definido. Pode-se verificar facilmente que \tilde{d} é uma métrica em \mathbb{R}^2 .

Sejam $a = \max\{|a_n|\}$, $s = \max\{|s_n|\}$ e $m = \max\{a, s\}$. Como $0 < a < 1$, temos que

$$\begin{aligned} \tilde{d}(w_n(x_1, y_1), w_n(x_2, y_2)) &= |a_n x_1 - a_n x_2| + \theta |c_n x_1 + s_n y_1 - c_n x_2 - s_n y_2| \\ &\leq |a_n| |x_1 - x_2| + \theta |c_n| |x_1 - x_2| + \theta |s_n| |y_1 - y_2| \\ &= (|a_n| + \theta |c_n|) |x_1 - x_2| + \theta |s_n| |y_1 - y_2|. \end{aligned} \quad (3)$$

Se todos os c_n forem nulos, para qualquer que seja θ , temos

$$\begin{aligned} \tilde{d}(w_n(x_1, y_1), w_n(x_2, y_2)) &\leq m (|x_1 - x_2| + \theta |y_1 - y_2|) \\ &= m \tilde{d}((x_1, y_1), (x_2, y_2)), \end{aligned}$$

caso contrário, isto é, se pelo menos um c_n for não nulo, seja

$$\theta = \frac{\min\{1 - |a_n|\}}{2 \max\{|c_n|\}}.$$

Denotando por $r = (1 + a)/2$, de (3) vem que

$$\begin{aligned} \tilde{d}(w_n(x_1, y_1), w_n(x_2, y_2)) &\leq r |x_1 - x_2| + \theta s |y_1 - y_2| \\ &= \max\{r, s\} \tilde{d}((x_1, y_1), (x_2, y_2)). \end{aligned}$$

Desta forma, provamos que todas as w_n são contrações na métrica \tilde{d} . Provemos agora a equivalência desta métrica com a euclidiana. Pela desigualdade de Cauchy-Schwartz, é fácil ver que

$$(|\xi_1| + \dots + |\xi_n|)^2 \leq n (|\xi_1|^2 + \dots + |\xi_n|^2).$$

Definindo $c = \max\{1, \theta\}$, temos que

$$\begin{aligned} \tilde{d}((x_1, y_1), (x_2, y_2)) &= |x_1 - x_2| + \theta |y_1 - y_2| \leq c (|x_1 - x_2| + |y_1 - y_2|) \\ &\leq c \sqrt{2} (|x_1 - x_2|^2 + |y_1 - y_2|^2)^{1/2} \\ &= c \sqrt{2} d((x_1, y_1), (x_2, y_2)). \end{aligned} \quad (4)$$

Por outro lado,

$$\begin{aligned} d((x_1, y_1), (x_2, y_2)) &= (|x_1 - x_2|^2 + |y_1 - y_2|^2)^{1/2} \\ &\leq (|x_1 - x_2|^2 + 2|x_1 - x_2||y_1 - y_2| + |y_1 - y_2|^2)^{1/2} \\ &= |x_1 - x_2| + |y_1 - y_2|. \end{aligned}$$

Se $\theta \geq 1$, então

$$\begin{aligned} d((x_1, y_1), (x_2, y_2)) &\leq |x_1 - x_2| + \theta |y_1 - y_2| \\ &= \tilde{d}((x_1, y_1), (x_2, y_2)), \end{aligned}$$

caso contrário,

$$\begin{aligned} d((x_1, y_1), (x_2, y_2)) &\leq 1/\theta (\theta |x_1 - x_2| + \theta |y_1 - y_2|) \\ &\leq 1/\theta (|x_1 - x_2| + \theta |y_1 - y_2|) \\ &= 1/\theta \tilde{d}((x_1, y_1), (x_2, y_2)), \end{aligned}$$

ou seja,

$$d((x_1, y_1), (x_2, y_2)) \leq C \tilde{d}((x_1, y_1), (x_2, y_2)), \quad (5)$$

onde $C = \max\{1, 1/\theta\}$. Logo, por (4) e (5),

$$1/C d((x_1, y_1), (x_2, y_2)) \leq \tilde{d}((x_1, y_1), (x_2, y_2)) \leq c \sqrt{2} d((x_1, y_1), (x_2, y_2)),$$

o que demonstra a equivalência das métricas. \square

Teorema II.3: *Considere o IFS, associado a Λ , $\{(\mathbb{R}^2, \tilde{d}); w_n, n \in \mathbb{N}\}$, com w_n definidas em (II.4) e (II.8) (ou (II.9)). Suponha que todos os fatores de escala vertical tenham módulo menor que um. Denote por G o atrator do IFS. Então G é o gráfico de uma função contínua $f_\Lambda : [x_0, x_N] \rightarrow \mathbb{R}$ que interpola Λ , isto é*

$$G = \{ (x, f_\Lambda(x)) \mid x \in [x_0, x_N] \},$$

com

$$f_{\Lambda}(x_n) = y_n, \quad \text{para todo } (x_n, y_n) \in \Lambda.$$

Prova: Denotemos por \mathcal{F} o conjunto das funções contínuas $f : [x_0, x_N] \rightarrow \mathbb{R}$ tais que $f(x_0) = y_0$ e $f(x_N) = y_N$. Neste espaço, definimos d , a métrica do máximo, por

$$d(f, g) = \max\{|f(x) - g(x)| \mid x \in [x_0, x_N]\}, \quad \text{para quaisquer } f, g \in \mathcal{F}.$$

Então (\mathcal{F}, d) é um espaço métrico completo (veja, por exemplo, [10]). Sejam a_n, c_n, e_n e f_n como em (II.8) (ou (II.9)). Defina agora a transformação $T : \mathcal{F} \rightarrow \mathcal{F}$ por

$$(Tf)(x) = c_n l_n^{-1}(x) + s_n f(l_n^{-1}(x)) + f_n \quad \text{para } x \in [x_{n-1}, x_n], \quad n \in \mathbf{N}, \quad (6)$$

onde $l_n : [x_0, x_N] \rightarrow [x_{n-1}, x_n]$ é dada por

$$l_n(x) = a_n x + e_n.$$

Será que a imagem de T realmente está contida em \mathcal{F} ? Vejamos que Tf preserva os pontos extremos.

$$(Tf)(x_0) = c_1 l_1^{-1}(x_0) + s_1 f(l_1^{-1}(x_0)) + f_1.$$

Há dois casos, dependendo do coeficiente de orientação:

$$o_1 = 1 \xrightarrow{\text{(II.3)}} (Tf)(x_0) = c_1 x_0 + s_1 f(x_0) + f_1 = y_0,$$

$$o_1 = -1 \xrightarrow{\text{(II.6)}} (Tf)(x_0) = c_1 x_N + s_1 f(x_N) + f_1 = y_0.$$

Para o extremo direito, temos

$$(Tf)(x_N) = c_N l_N^{-1}(x_N) + s_N f(l_N^{-1}(x_N)) + f_N.$$

Analogamente, aqui também há dois casos:

$$o_N = 1 \xrightarrow{\text{(II.3)}} (Tf)(x_N) = c_N x_N + s_N f(x_N) + f_N = y_N,$$

$$o_N = -1 \xrightarrow{\text{(II.6)}} (Tf)(x_N) = c_N x_0 + s_N f(x_0) + f_N = y_N.$$

Logo os extremos são preservados por T .

Falta verificar se Tf é contínua. Em cada subintervalo (x_{n-1}, x_n) , isto é claro, já que a composição é apenas de funções contínuas. A dúvida sobre a continuidade de Tf só pode surgir nos pontos extremos dos subintervalos. Precisamos ter certeza se a definição de $(Tf)(x_n)$ não traz problemas pelo fato de x_n estar tanto em $[x_{n-1}, x_n]$ quanto em $[x_n, x_{n+1}]$. Vejamos.

$$(Tf)(x_n) = c_n l_n^{-1}(x_n) + s_n f(l_n^{-1}(x_n)) + f_n = \begin{cases} c_n x_N + s_n f(x_N) + f_n = y_n, & \text{se } o_n = 1 \\ c_n x_0 + s_n f(x_0) + f_n = y_n, & \text{se } o_n = -1 \end{cases}$$

e

$$\begin{aligned} (Tf)(x_n) &= c_{n+1} l_{n+1}^{-1}(x_n) + s_{n+1} f(l_{n+1}^{-1}(x_n)) + f_{n+1} = \\ &= \begin{cases} c_{n+1} x_0 + s_{n+1} f(x_0) + f_{n+1} = y_n, & \text{se } o_n = 1 \\ c_{n+1} x_N + s_{n+1} f(x_N) + f_{n+1} = y_n, & \text{se } o_n = -1 \end{cases} \end{aligned}$$

Logo, Tf realmente é contínua e, portanto, $Tf \in \mathcal{F}$.

Provaremos agora que T é uma contração em (\mathcal{F}, d) . Sejam $f, g \in \mathcal{F}$, $n \in \mathbf{N}$ e $x \in [x_{n-1}, x_n]$. Então

$$|(Tf)(x) - (Tg)(x)| = |s_n| |f(l_n^{-1}(x)) - g(l_n^{-1}(x))| \leq |s_n| d(f, g).$$

Por consequência,

$$d(Tf, Tg) \leq s d(f, g),$$

onde $s = \max\{|s_n| \mid n \in \mathbf{N}\} < 1$. Pelo Teorema do ponto fixo de Banach, T tem um único ponto fixo f_Λ em \mathcal{F} . Vejamos que f_Λ interpola Λ . Como $(Tf_\Lambda)(x) = f_\Lambda(x)$,

$$(Tf_\Lambda)(x_n) = c_n l_n^{-1}(x_n) + s_n f_\Lambda(l_n^{-1}(x_n)) + f_n = c_n x_N + s_n f_\Lambda(x_N) + f_n = f_\Lambda(x_n),$$

e assim, usando (II.8), temos que

$$\begin{aligned} f_\Lambda(x_n) - s_n f_\Lambda(x_N) &= (x_N - x_0)^{-1} \left((y_n - y_{n-1}) x_N - s_n (y_N - y_0) x_N + \right. \\ &\quad \left. + (x_N y_{n-1} - x_0 y_n) - s_n (x_N y_0 - x_0 y_N) \right) \\ &= (x_N - x_0)^{-1} \left((x_N - x_0) y_n - s_n y_N x_N + s_n x_0 y_N \right) \\ &= (x_N - x_0)^{-1} (x_N - x_0) (y_n - s_n y_N) \\ &= y_n - s_n y_N = y_n - s_n f_\Lambda(x_N) \end{aligned}$$

e portanto

$$f_\Lambda(x_n) = y_n.$$

Se tivéssemos usado (II.9), ao invés de (II.8), obteríamos o mesmo resultado. Logo f_Λ , ponto fixo de T , é uma função contínua que interpola Λ .

Para finalizar, mostremos que o gráfico \tilde{G} de f_Λ é o atrator do IFS definido. Reescrevendo (6), vem que

$$(Tf_\Lambda)(l_n(x)) = (Tf_\Lambda)(a_nx + e_n) = c_nx + s_nf_\Lambda(x) + f_n, \quad \text{para } x \in [x_{n-1}, x_n], \quad n \in \mathbf{N},$$

o que implica em

$$\tilde{G} = \bigcup_{n \in \mathbf{N}} w_n(\tilde{G}).$$

Mas $\tilde{G} \in \mathcal{H}(\mathbb{R}^2)$ e como o atrator do IFS é único, $\tilde{G} = G$. □

Demonstrações do Capítulo III

Teorema III.2: *Seja $A \in \mathcal{H}(M)$, onde (M, d) é um espaço métrico. Sejam $\epsilon_n = Cr^n$, para qualquer r que satisfaça $0 < r < 1$, e $C > 0$ com $n = 1, 2, 3, \dots$. Então*

$$D = \lim_{n \rightarrow \infty} \left(\frac{\ln \mathcal{N}(A, \epsilon_n)}{\ln(1/\epsilon_n)} \right)$$

é a dimensão fractal de A .

Prova: Sejam r e C números reais, tais que $0 < r < 1$ e $C > 0$, e $E = \{\epsilon_n \mid n = 1, 2, \dots\}$. Defina $f(\epsilon) = \max\{\epsilon_n \in E \mid \epsilon_n \leq \epsilon\}$. Assuma que $\epsilon < r$. Então é fácil ver que

$$f(\epsilon) \leq \epsilon \leq f(\epsilon)/r \quad \text{e} \quad \mathcal{N}(A, f(\epsilon)) \geq \mathcal{N}(A, \epsilon) \geq \mathcal{N}(A, f(\epsilon)/r).$$

Como a função logaritmo é crescente e positiva, quando seu argumento é maior que um, temos que

$$\frac{\ln \mathcal{N}(A, f(\epsilon)/r)}{\ln(1/f(\epsilon))} \leq \frac{\ln \mathcal{N}(A, \epsilon)}{\ln(1/\epsilon)} \leq \frac{\ln \mathcal{N}(A, f(\epsilon))}{\ln(r/f(\epsilon))} \quad (7)$$

Assumindo que $\mathcal{N}(A, \epsilon) \rightarrow \infty$ quando $\epsilon \rightarrow 0$ (caso contrário, o teorema já é válido, pois todos os termos vão a zero), para o lado direito de (7), temos que

$$\begin{aligned} \lim_{\epsilon \rightarrow 0} \frac{\ln \mathcal{N}(A, f(\epsilon))}{\ln(r/f(\epsilon))} &= \lim_{n \rightarrow \infty} \frac{\ln \mathcal{N}(A, \epsilon_n)}{\ln(r/\epsilon_n)} \\ &= \lim_{n \rightarrow \infty} \frac{\ln \mathcal{N}(A, \epsilon_n)}{(\ln r + \ln(1/\epsilon_n))} \\ &= \lim_{n \rightarrow \infty} \frac{\ln \mathcal{N}(A, \epsilon_n)}{\ln(1/\epsilon_n)}. \end{aligned}$$

Analogamente, para o lado esquerdo, temos

$$\begin{aligned} \lim_{\epsilon \rightarrow 0} \frac{\ln \mathcal{N}(A, f(\epsilon)/r)}{\ln(1/f(\epsilon))} &= \lim_{n \rightarrow \infty} \frac{\ln \mathcal{N}(A, \epsilon_{n-1})}{\ln(1/\epsilon_n)} \\ &= \lim_{n \rightarrow \infty} \frac{\ln \mathcal{N}(A, \epsilon_{n-1})}{(\ln 1/r + \ln(1/\epsilon_{n-1}))} \\ &= \lim_{n \rightarrow \infty} \frac{\ln \mathcal{N}(A, \epsilon_n)}{\ln(1/\epsilon_n)}. \end{aligned}$$

Como ambos os lados de (7) tende ao mesmo limite, concluímos que

$$D = \lim_{\epsilon \rightarrow 0} \frac{\ln \mathcal{N}(A, \epsilon)}{\ln(1/\epsilon)} = \lim_{n \rightarrow \infty} \left(\frac{\ln \mathcal{N}(A, \epsilon_n)}{\ln(1/\epsilon_n)} \right),$$

o que demonstra o teorema. \square

Teorema III.3[Box Counting]: *Seja $A \in \mathcal{H}(\mathbb{R}^m)$, com d a métrica euclidiana em \mathbb{R}^m . Cubra o \mathbb{R}^m por hipercubos fechados, que se interceptem apenas nas faces, de lados $(1/2)^n$. Seja $\mathcal{N}_n(A)$ o número de hipercubos que interceptam A . Então*

$$D = \lim_{n \rightarrow \infty} \left(\frac{\ln \mathcal{N}_n(A)}{\ln 2^n} \right) \quad (8)$$

é a dimensão fractal de A .

Prova: Primeiramente, notemos que para $m = 1, 2, 3, \dots$

$$2^{-m} \mathcal{N}_{n-1} \leq \mathcal{N}(A, 2^{-n}) \leq \mathcal{N}_{k(n)},$$

onde $k(n)$ é o menor inteiro k tal que $k \geq n - 1 + 1/2 \log_2 m$. A primeira desigualdade é verdadeira pois bolas de raio 2^{-n} podem interceptar no máximo 2^m células de aresta $2^{-(n-1)}$ da malha. A segunda desigualdade segue do fato de que cada célula de aresta s pode ser inscrita numa bola de raio r dado por $r^2 \geq (s/2)^2 + \dots + (s/2)^2 = m(s/2)^2$, pelo teorema de Pitágoras. Agora

$$\lim_{n \rightarrow \infty} \left(\frac{\ln \mathcal{N}_{k(n)}}{\ln 2^n} \right) = \lim_{n \rightarrow \infty} \left(\frac{\ln 2^{k(n)}}{\ln 2^n} \frac{\ln \mathcal{N}_{k(n)}}{\ln 2^{k(n)}} \right) = D,$$

já que $k(n)/n \rightarrow 1$. Como também podemos ver

$$\lim_{n \rightarrow \infty} \left(\frac{\ln 2^{-m} \mathcal{N}_{n-1}}{\ln 2^n} \right) = \lim_{n \rightarrow \infty} \left(\frac{\ln \mathcal{N}_{n-1}}{\ln 2^{n-1}} \right) = D.$$

Utilizando a Teorema III.2, com $r = 1/2$, completamos a demonstração. \square

Apêndice B

Listagem das rotinas

Neste apêndice apresentamos as rotinas desenvolvidas e implementadas no decorrer dessa tese. Todos os procedimentos foram implementados em MATLAB, versão 4.0. As rotinas que se mostraram lentas, devido a característica interpretativa do MATLAB, também foram implementadas em C. O compilador usado foi o BC++ 4.5.

ifsplot Rotina encarregada de gerar o atrator de um IFS em \mathbb{R}^2 , onde as contrações consideradas são transformações afins. Os parâmetros de entrada são: IFS, matriz com as contrações do IFS; nint, número de iterações; xini,yini, ponto inicial; iprev, quantidade de iterações a serem feitas antes que os pontos sejam plotados; plot_op, opções usais para o comando plot do MATLAB. A rotina retorna os vetores X eY contendo os pontos computados.

```
function [X,Y]=ifsplot(IFS,nint,xini,yini,iprev,plot_op)

npar = nargin;

if npar == 2
    xini = 0;
    yini = 0;
    iprev = 10;
    plot_op = 'y';
elseif npar == 4
    iprev == 15;
    plot_op = 'y';
elseif npar == 5
```

```

    plot_op = 'y';
elseif npar == 6
else disp(' ')
    disp('Entrada invalida')
    disp(' ')
    disp('Entradas possivies:')
    disp('>> ifsplot(ifs_file,nint)')
    disp('>> ifsplot(ifs_file,nint,xini,yini)')
    disp('>> ifsplot(ifs_file,nint,xini,yini,iprev)')
    disp('>> ifsplot(ifs_file,nint,xini,yini,iprev,plot_op)')
return
end

% numero de contracoes
[n,m]=size(IFS);

% Gera vetor da acumulada
F=zeros(n,1); F(1)=IFS(1,7); for i=2:n-1 F(i)=F(i-1)+IFS(i,7); end F(n)=1;

for i=1:iprev
    a=rand(1,1);
    j= 1;
    while a > F(j,1), j=j+1; end
    nc=j;

    w    = IFS(nc,1) * xini + IFS(nc,2) * yini + IFS(nc,5);
    yini = IFS(nc,3) * xini + IFS(nc,4) * yini + IFS(nc,6);
    xini = w;
end

X(1) = xini;
Y(1) = yini;

for k=1:nint
    a=rand(1,1);
    j= 1;
    while a > F(j,1), j=j+1; end
    nc=j;

    w    = IFS(nc,1) * xini + IFS(nc,2) * yini + IFS(nc,5);
    yini = IFS(nc,3) * xini + IFS(nc,4) * yini + IFS(nc,6);
    xini =w;

    X(k+1) = xini;
    Y(k+1) = yini;
end

h = plot(X,Y,[plot_op '.']);

```

```
set(h,'markersize',2);
```

mkfif Esta rotina controla as contrações do IFS associado ao conjunto de dados Λ (pontos de interpolação ou de corte). Os parâmetros de entrada são: x, y , vetores com os pontos de interpolação ou de corte; s , vetor com os fatores de escala vertical; o , vetor com os coeficientes de orientação. O único parâmetro de saída é FIF, matriz com as contrações associadas ao IFS.

```
function FIF = mkfif(x,y,s,o)

n=length(x);

FIF=zeros(n-1,6);

deltax= x(n)-x(1);
deltay= y(n)-y(1);

for i = 1:n-1
    if o(i) == 1,
        xa = x(i);
        xb = x(i+1);
        ya = y(i);
        yb = y(i+1);
    else
        xa = x(i+1);
        xb = x(i);
        ya = y(i+1);
        yb = y(i);
    end
    FIF(i,1) = ( xb-xa )/deltax;
    FIF(i,2) = ( yb-ya )/deltax - s(i)*deltay/deltax;
    FIF(i,3) = s(i);
    FIF(i,4) = ( x(n)*xa - x(1)*xb )/deltax;
    FIF(i,5) = ( x(n)*ya - x(1)*yb )/deltax - s(i)*(x(n)*y(1) - x(1)*y(n))/deltax;
    FIF(i,6) = 1/(n-1);
end
```

datafill Rotina encarregada de interpolar linearmente os pontos não atingidos, da malha, pela rotina `fifplot`. Os parâmetros de entrada são: x, y , vetores de pontos a serem completados; $f1$, vetor que indica quais pontos foram efetivamente calculados (se $f1(i)=0$ significa que o i -ésimo ponto não foi computado, caso contrário $f1(i)=1$). A rotina retorna os mesmos vetores x, y , porém, agora, completados.

```

function [x,y]=datafill(x,y,fl)

n = length(x);

% Assumindo que as pontas do vetor estao sempre ocupadas
% (condicao satisfeita pelos vetores gerados por fifplot)

i=1;
j=i+1;
while j < n
    while fl(j) == 0 & j < n, j=j+1; end
    if j <= n,
        m = (y(j) - y(i))/(x(j) - x(i));
        y(i+1:j-1) = y(i) + m*(x(i+1:j-1) - x(i));
        while fl(j) == 1 & j < n, j=j+1; end
        i=j-1;
        j = i+1;
    end
end
end

```

fifplot Constrói o gráfico de uma FIF. Os parâmetros de entrada são: FIF, matriz de contrações (criada por mkfif); x_k, y_k , vetores contendo os pontos de corte; npts, número de pontos na malha; imax, número de iterações; exhibe, variável que controla a plotagem do resultado. Os parâmetros de saída são: x, y , vetores de pontos computados; fl, vetor que marca quais pontos da malha foram atingidos; r, porcentagem de pontos atingidos.

```

function [x,y,flag,r] = fifplot(FIF,xk,yk,npts,imax,exibe)

% numero de contracoes
[n,m]=size(FIF);

xmin = xk(1);
xmax = xk(n+1);

% inicializa vetores y e flag
y = zeros(npts,1);
flag = y;

% Gera vetor da acumulada
F = zeros(n,1);
F(1,1)= FIF(1,6);

for i=2:n-1

```

```

    F(i,1)=F(i-1,1)+FIF(i,6);
end
F(n,1)=1;

% Semente inicial
x = xk(1);
k = 1;

for i=1:imax

    a=rand(i,1);
    j= 1;
    while a > F(j,1), j=j+1; end
    nc=j;

    w          = FIF(nc,1) * x  + FIF(nc,4);
    knew       = round((w-xmin)*(npts-1)/(xmax-xmin) + 1);
    y(knew)    = FIF(nc,2) * x  + FIF(nc,3) * y(k) + FIF(nc,5);
    flag(knew) = 1;
    x=w;k=knew;
end

nn = length(y);
x = (linspace(xmin,xmax,nn))';

% Calcula o vetor de indices dos nos
for k=1:n+1
    ik(k) = convert(xmin,xmax,npts,xk(k));
end

% Forca que os nos estajam presentes
y(ik) = yk;
flag(ik) = ones(n+1,1);

if nargout == 2,
    [x,y] = datafill(x,y,flag);
else
    r = norm(flag,1)/nn;
end

if existe ~=0,
    if nargout == 4,
        for i=1:nn
            if flag(i) == 0, y(i) = NaN; end
        end
        plot(x,y,'.', 'markersize',1)
    elseif nargout == 2,
        plot(x,y)
    end
end

```

```

    end
end

```

s1 Estimador dos fatores de escala vertical. Os parâmetros de entrada são: x, y , gráfico em questão; xk , vetor contendo as abscissas dos pontos de corte. A rotina retorna um vetor s , com os fatores de escala vertical estimados.

```

function s = s1(x,y,xk)

n = length(xk);
nx = length(x);

% Define particao coerente com a malha
ik = convert(x(1),x(nx),nx,xk);
xk = x(ik);
yk = y(ik);

h = height(x,y);
[a,j] = max(abs(h));
smax = h(j);

for i=1:n-1
    I = ik(i):ik(i+1);

    h = height(x(I),y(I));
    [a,j] = max(abs(h));
    s(i) = h(j);
end

s = s(:) / smax;

```

oest Estimador dos coeficientes de orientação. Os parâmetros de entrada são: x, y , gráfico em questão; xk , vetor contendo as abscissas dos pontos de corte. A rotina retorna um vetor o , com os coeficientes de orientação estimados.

```

function o = oest(x,y,xk)

n = length(x);
nk = length(xk);

ik = convert(x(1),x(n),n,xk);

h = height(x,y);

```

```

[a,jmax] = max(abs(h));

if jmax > round(n/2) , dir = 1;
    else dir = -1;
end

for i = 1:nk-1
    h = height(x(ik(i):ik(i+1)),y(ik(i):ik(i+1)));
    [a,jlocal] = max(abs(h));
    if jlocal > round((ik(i+1)-ik(i)+1)/2), o(i) = dir;
        else o(i) = - dir;
    end
end
end

```

convert Calcula os índices de cada elemento de um vetor, numa malha discreta. Os parâmetros de entrada são: a,b , extremos do intervalo discretizado; n , número de pontos da malha; x , pontos em $[a,b]$ a serem discretizados. O único parâmetro de saída é o vetor k de índices dos elementos de x .

```

function k = convet(a,b,n,x)

k = round((x-a)*(n-1)/(b-a) + 1);

```

height Calcula a diferença entre o gráfico de uma função e a reta que une seus extremos. Os parâmetros de entrada são: x,y , gráfico em questão. O único parâmetro de saída é o vetor h contendo a diferença entre o gráfico e a reta que une seus extremos.

```

function h = height(x,y)

n = length(x);

m = (y(n)-y(1))/(x(n)-x(1));

h = y - y(1) - m *(x-x(1));

```

dfif Rotina encarregada de calcular a dimensão fractal de uma função de interpolação fractal, resolvendo a equação (III.2). Para a solução desta equação, usamos o método de Newton. Os parâmetros de entrada são: fif , matriz das contrações que definem o IFS (gerada por `mkfif`); $d0$, chute inicial para a dimensão; $pres$, precisão requerida. Os parâmetros de saída são: d , a dimensão computada; r , o resíduo.


```

function [d,r] = dfif(fif,d0,pres)

if nargin < 2,
    d0 = 1.2;
    pres = 10*eps;
elseif nargin == 2,
    pres = 10*eps;
end

a = abs(fif(:,1));
s = abs(fif(:,3));
cn = ones(length(a),1);

if sum(s) <= 1,
    d = 1;
    if nargin > 1,
        r = abs(s' * a.^(d-1) - 1);
    end
    return
end

d = d0; fun = s' * a.^(d-1) - 1;

while abs(fun) > pres,
    fun = s' * a.^(d-1) - 1;
    der = ((s' * a.^(d+eps-1) - 1) - (s' * a.^(d-1) - 1))/eps;
    d = d - fun/der ;
end

if nargin > 1,
    r = abs(s' * a.^(d-1) - 1);
end

```

dest2 Rotina encarregada de estimar a dimensão fractal de um gráfico, através do algoritmo *Box Counting*. Os parâmetros de entrada são: x,y , gráfico sob interesse; L , nível mais grosseiro da malha (2^{-L}); U , nível mais fino da malha (2^{-U}); v , variável que controla a exibição da reta ajustada. A rotina retorna a dimensão fractal estimada D .

```

function D = dest2(x,y,L,U,v)

if nargin < 5, v = 1;
end

n = length(x);

```

```

% Mapeia o grafico na caixa [0,1]x[0,1]
x = (x-x(1))/(x(n)-x(1)) ;

ymin = min(y);
ymax = max(y);
y = (y-ymin)/(ymax-ymin);

ik=[1 n];
for i=1:L-1
    nk = length(ik);
    for j=1:nk-1
        ikk(2*j-1) = ik(j);
        ikk(2*j ) = round( (ik(j+1)+ik(j)) /2 );
    end
    ik = [ikk n];
end

for k =L:U
    nk = length(ik);
    for j=1:nk-1
        ikk(2*j-1) = ik(j);
        ikk(2*j ) = round( (ik(j+1)+ik(j)) /2 );
    end
    ik = [ikk n];
    nk = length(ik);

    box = 0;
    for j=1:nk-1
        ymin = min(y(ik(j):ik(j+1)));
        ymax = max(y(ik(j):ik(j+1)));

        if ymin == 1
            ni =2;
        else
            ni = floor(ymin*2^k) + 1;
        end
        if ymax == 1
            nf =2;
        else
            nf = floor(ymax*2^k) + 1;
        end

        box = box + (nf-ni+1);
    end

    d(k-L+1,1) = k;

```

```

    d(k-L+1,2) = box;
end

d(:,2) = log(d(:,2))/log(2);
on      = ones(U-L+1,1);

A = [(on'*on) (d(:,1)')*on) ; (d(:,1)')*on) (d(:,1)')*d(:,1)];
b = [(d(:,2)')*on) ; (d(:,2)') * d(:,1))];

b = A\b;
D = b(2);

if v == 0, return
end

xi = d(1,1):.01:d(U-L+1,1);
yi = b(2)*xi+b(1);
plot(xi,yi,d(:,1),d(:,2),'y*')
title(sprintf('D = %6.4f',D))
xlabel('n'),ylabel('ln N(n) / ln 2')

```

dimen Rotina encarregar de calcular as estimativas da dimensão variando n_i e n_f (parâmetros que definem os tamanhos inicial e final da célula no *Box Counting*). Os parâmetros de entrada são: x, y , gráfico de interesse; L , potência que define $n_i = 2^{-L}$; U , potência que define $n_f = 2^{-U}$; op , opções padrões do plot. Os parâmetros de saída são: $dmax$, dimensão máxima estimada; $dmin$, dimensão mínima estimada; $imax$, índice onde $dmax$ foi atingido; $imin$, índice onde $dmin$ foi atingido; D , matriz com todas as estimativas calculadas.

```

function [dmax,dmin,imax,imin,D] = dimen(x,y,L,U,op)

n = length(x);
if nargin < 5, op = 'y';
end

% Mapeia o grafico na caixa [0,1]x[0,1]

x = (x-x(1))/(x(n)-x(1)) ;

ymin = min(y);
ymax = max(y);
y = (y-ymin)/(ymax-ymin);

ik=[1 n];

```

```

for i=1:L-1
    nk = length(ik);
    for j=1:nk-1
        ikk(2*j-1) = ik(j);
        ikk(2*j ) = round( (ik(j+1)+ik(j)) /2 );
    end
    ik = [ikk n];
end

for k =L:U
    nk = length(ik);
    for j=1:nk-1
        ikk(2*j-1) = ik(j);
        ikk(2*j ) = round( (ik(j+1)+ik(j)) /2 );
    end
    ik = [ikk n];
    nk = length(ik);

    box = 0;
    for j=1:nk-1
        ymin = min(y(ik(j):ik(j+1)));
        ymax = max(y(ik(j):ik(j+1)));

        if ymin == 1
            ni =2;
        else
            ni = floor(ymin*2^k) + 1;
        end
        if ymax == 1
            nf =2;
        else
            nf = floor(ymax*2^k) + 1;
        end

        box = box + (nf-ni+1);
    end

    d(k-L+1,1) = 2^k;
    d(k-L+1,2) = box;

    if k > L
        clear dd
        dd(:,1) = log(d(:,1))/log(2);
        dd(:,2) = log(d(:,2))/log(2);
        on      = ones(k-L+1,1);

        A = [(on'*on) (dd(:,1)')*on) ; (dd(:,1)')*on (dd(:,1)')*dd(:,1)];
    end
end

```

```

        b = [(dd(:,2))*on) ; (dd(:,2)' * dd(:,1))];

        b = A\b;
        D(k-L) = b(2);
    end
end

[dmax,imax] = max(D);
[dmin,imin] = min(D);

imax = imax + L;
imin = imin + L;

if op == 'i', return, end

plot(L+1:U,D,op,L+1:U,D,'*g',imax,dmax,'*c',imin,dmin,'*c');
b = (dmax-dmin)/10;
axis([L U+1 dmin-b dmax+b]);

set(gca,'ytick',sort(D),'xtick',L+1:U);
title(sprintf('Dmax = %6.4f, Dmin = %6.4f (%5.2f%%)',...
    dmax,dmin,100*(dmax-dmin)/dmax));
xlabel('Upper limit for box reduction') ylabel('Estimated dimension')

```

xinv Mapeia um vetor de pontos de um intervalo em outro. Os parâmetros de entrada são: x_0, x_n , extremos do intervalo *destino*; x_a, x_b , extremos do intervalo *origem*; x , pontos a serem mapeados. Os parâmetros de saída são: X , mapeamento de x .

```

function X = xinv(x0,xn,xa,xb,x)

X = x0 + (xn - x0)/(xb-xa) * (x-xa);

```

fknots Implementação de um passo do algoritmo de seleção de pontos para o problema de aproximação fractal. Os parâmetros de entrada são: x, y , gráfico em questão; s_{max} , limite superior para o módulo do fator de escala vertical; k_0 , índice inicial para a busca do próximo ponto; n_f , índice final para busca do próximo ponto. Os parâmetros de saída são: k_{otimo} , próximo ponto de corte; o , coeficiente de orientação associado; $tempo$, tempo gasto no processo.

```

function [kotimo,o,tempo] = fknots(x,y,smax,k0,nf)

```

```

n = length(x);

% Comprimento mínimo que um subintervalo pode ter
lmin = 5;

tries1 = zeros(nf-k0-lmin+1,3);

t0 = clock;

normy2 = norm(y,2);

for i=k0+lmin:nf

    iaux = i - k0 - lmin + 1;

    %---> Calcula fator de escala vertical apropriado
    %---> para o subintervalo [x(k0),x(i)]

    if k0 == 1, xk = [x(k0) x(i) x(n)];
                yk = [y(k0) y(i) y(n)];
                s = s1(x,y,xk);
                s = s(1);

        else xk = [x(1) x(k0) x(i) x(n)];
                yk = [y(1) y(k0) y(i) y(n)];
                s = s1(x,y,xk);
                s = s(2);
    end %(if k0 == 1)

    tries(iaux,3) = s;

if abs(s) < smax,

    xsub = x(k0:i);
    xsub = xsub(:)';
    ysub = y(k0:i);
    ysub = ysub(:)';

    for o = -1:2:1

        yy = 0*y;
        yy(1) = y(1);
        yy(n) = y(n);

        clear fl
        fl(1) = 1;

```

```

fl(n) = 1;

if o == 1,
    %---> Imagem inversa dos x em [x(k0),x(i)] no intervalo [x(1),x(n)]
    xint = xinv(x(1),x(n),x(k0),x(i),xsub);

    xa = x(k0);
    xb = x(i);
    ya = y(k0);
    yb = y(i);
else % (if o == 1)
    %---> Imagem inversa dos x em [x(k0),x(i)] no intervalo [x(1),x(n)]
    xint = xinv(x(1),x(n),x(i),x(k0),xsub);

    xa = x(i);
    xb = x(k0);
    ya = y(i);
    yb = y(k0);
end % (if o == 1)

d = 1/(x(n)-x(1));
a = d * (xb-xa);
e = d * (x(n) * xa - x(1) * xb);
c = d * ((yb - ya) - s * (y(n) - y(1)));
f = d * ((x(n)*ya - x(1)*yb) - s * (x(n)*y(1) - x(1)*y(n)));

% Montagem da transformacao que mapeia [x(k0),x(i)] -> [x(1),x(n)]
% Ak0 = [a 0 ; c s];
% bk0 = [e ; f];

%---> Inversa da matriz A (Ai = inv(Ak0));
Ai = [ 1/a 0 ; -c/(a*s) 1/s];

p = Ai * ( [xsub-bk0(1); ysub-bk0(2)] );

kint = convert(x(1),x(n),n,xint);
yy(kint) = p(2,:);
fl(kint) = ones(1,length(kint));

[x,yy]=datafill(x,yy,fl);

if o == 1, tries(iaux,1) = norm(y-yy,2)/normy2;
else tries(iaux,2) = norm(y-yy,2)/normy2;
end % (if o == 1)
end % (for o = 1:-2:-1)

```

```

    else % (if abs(s) < smax)
        tries(iaux,1) = 1;
        tries(iaux,2) = 1;
    end % (if abs(s) < smax)

    fprintf(['x(%4i) = %6.3f , s = %6.3f. o = 1 => erro = %6.4f,'...
           ' o = -1 => erro = %6.4f \n'],...
           i,x(i),s,tries(iaux,1),tries(iaux,2));

end % ( for i=k0+lmin:nf)

[min1,iot1] = min(tries(:,1));
[min2,iot2] = min(tries(:,2));

if min1 < min2, iotimo = iot1; o = 1; erro = min1;
else          iotimo = iot2; o =-1; erro = min2;
end % (if min1 < min2)

kotimo = iotimo + k0 + lmin -1;

fprintf(['\n x(%4i) = %6.3f \n s = %6.4f \n o = %2i'...
        '\n norma do erro = %7.5f'],...
        kotimo,x(kotimo),tries(iotimo,3),o,erro);

tempo = etime(clock,t0);
fprintf('\n Tempo gasto em segundos: %5.2f \n',tempo);

```

param Implementação do algoritmo de seleção de pontos. Basicamente, executa sucessivas vezes a rotina `fknots`. Os parâmetros de entrada são: `x,y`, gráfico em questão; `smax`, limitante superior para o módulo dos fatores de escala vertical; `kinicial`, ponto de corte inicial; `ray`, tamanho máximo do subintervalo aceitável. Os parâmetros de saída são: `xk,yk`, vetores com os pontos de corte; `ik`, vetor dos índices dos pontos de corte; `s`, vetor com os fatores de escala vertical; `o`, vetor com os coeficientes de orientação; `tempo`, tempo gasto no processo.

```

function [xk,yk,ik,s,o,tempo]=param(x,y,smax,kinicial,ray)

n=length(x);

k0 = kinicial;
ik(1) = kinicial;
tempo =0;

lmin = 5;

```



```

while k0 < n-lmin;
  if k0 == 1, nf = round(n/1.2);
  else nf = min([(k0+ray) n]);
  end

  [k1,oo,t] = fknots(x,y,smax,k0,nf);
  tempo = tempo +t;
  ik(length(ik)+1)=k1
  o(length(ik)-1) = oo;

  k0 = k1;
end

if k0 < n, ik(length(ik)+1) = n
          o(length(ik)-1) = 1;
end

xk=x(ik);
yk=y(ik);
s=s1(x,y,xk);

```

calcs Calcula o fator de escala vertical de um determinado subintervalo em função da dimensão fractal. Os parâmetros de entrada são: **xk**, vetor com as abscissas dos pontos de corte; **s**, vetor com os fatores de escala vertical; **K**, índice do subintervalo em questão; **D**, vetor de dimensão fractal. Os parâmetros de saída são: **ss**, vetor com os fatores de escala vertical em função de **D**; **Dmin**, mínima dimensão factível; **Dmax**, máxima dimensão factível.

```

function [ss,Dmin,Dmax] = calcs(xk,s,K,D)

N = length(D);

s = s(:);
xk = xk(:);
n = length(xk);
I = [1:K-1 K+1:n-1];

for k = 1:N
  a = ((xk(2:n)-xk(1:n-1))/(xk(n)-xk(1))).^(D(K)-1);
  ss(k) = (1 - abs(s(I))' * a(I))/a(K);
end

saux = abs(ss);

```

```
[saux,imin]=min(abs(ss));
Dmin = D(imin);

[saux,imax]=min(abs(ss-1));
Dmax = D(imax);
```

aprdata Gera o arquivo de entrada para o programa em C que encontra os pontos de corte para o problema de aproximação fractal. Os parâmetros de entrada são: x, y , gráfico em questão; $smax$, limitantes para o módulo dos fatores de escala vertical. A rotina grava os dados no arquivo texto `data.txt`.

```
function aprdata(x,y,smax)

x=x(:);
y=y(:);
n = length(x);

[mm,nn]=size(smax);

A = [mm smax(:,1)'+1 n x'
     mm smax(:,2)' n y']';
format long e
save data.txt A -ascii
format short
```

recdata Gera o arquivo de entrada para o programa em C que encontra os pontos de corte para o problema de reconstrução fractal. Os parâmetros de entrada são: x, y , gráfico em questão; $smax$, limitantes para o módulo dos fatores de escala vertical; a, b , extremos do subintervalo ausente. A rotina grava os dados no arquivo texto `data.txt`.

```
function recdata(x,y,smax,a,b)

x=x(:);
y=y(:);
n = length(x);
[mm,nn]= size(smax);

A = [mm (smax(:,1)'+1) (a-1) n x'
     mm (smax(:,2)') (b-1) n y']';
```

```
format long e
save data.txt A -ascii
format short
```

getparam Lê o arquivo de saída dos programas em C `spapr` e `sprec` (`ikso.txt`). Os parâmetros de saída são: `ik`, vetor com os índices dos pontos de corte; `s`, vetor com os fatores de escala vertical; `o`, vetor com os coeficientes de orientação.

```
function [ik,s,o]=getparam

load -ascii ikso.txt

[n,m]=size(ikso);

ik = ikso(:,1);
s = ikso(2:n,2);
o = ikso(2:n,3);
clear ikso
```

fdsdata Gera o arquivo `posdata.txt` de entrada para o programa em C, `finds`, que encontra o melhor fator de escala vertical para o subintervalo ausente. Os parâmetros de entrada são: `x`, `y`, gráfico em questão; `ik`, índices dos pontos de corte; `s`, vetor com os fatores de escala vertical; `o`, vetor com os coeficientes de orientação; `sot`, fatores de escala vertical a serem analisados.

```
function fdsdata(x,y,ik,s,o,K,sot)

x = x(:);
y = y(:);
nx = length(x);

ik = ik(:);
nk = length(ik)-1;
s = s(:);
o = o(:);

sot = sot(:);
ns = length(sot);

A = [nx x' nk ik' s' ns sot'
     00 y' K zeros(1,nk+1) o' 00 zeros(1,ns) ]';
```

```
format long e
save posdata.txt A -ascii
format short
```

spapr Implementação em C do algoritmo de seleção de pontos para o problema de aproximação fractal de sinais.

```
#include <time.h>
#include <math.h>
#include <stdio.h>

/*****
  Calcula a diferenca entre a funcao e a reta que une os extremos
  *****/
float height(float *x, float *y, int n)
{
    float m = (y[n-1]-y[0])/(x[n-1]-x[0]);

    float h=0;

    for (int i=0; i < n ; i++)
    {
        float aux = y[i]-y[0] - m*(x[i]-x[0]);
        if (fabs(aux)>fabs(h))
            h = aux;
    }

    return h;
}

/*****
  Estimador dos fatores de escala vertical
  *****/
void s1(float *x, float *y, int n, int *ik, int nk, float *s)
{
    float smax = height(x,y,n);

    for (int i =0; i < nk-1; i++)
        s[i] = height(x+ik[i],y+ik[i],ik[i+1]-ik[i]+1) /smax;
}

/*****
  Inverte um conjunto de indices do intervalo [i0,i1] para o grande
  *****/
void ikinv(int n, int i0, int i1, int d, int *ik)
```

```

{

for (int i =0 ; i <= abs(i1-i0); i++)
  if (d==1)
    ik[i] = (int) floor( (n-1) * float(i) /float(abs(i1-i0)));
  else
    ik[i0-i1-i] =(int) floor( (n-1) * float(i) /float(abs(i1-i0)));

}

/*****
  Interpola linearmente os pontos de x,y segundo fl
*****/
void datafill(float *x, float *y, int *fl, int n)
{
  int i = 0;
  int j = i + 1;
  int k;
  float m;

  while (j < n-1)
  {
    while (fl[j] == 0 && j < n-1) j++;
    if (j <= n-1)
      m = (y[j] - y[i])/(x[j]-x[i]);

    for (k=i+1 ; k < j ; k++)
      y[k] = y[i] + m * (x[k] - x[i]);

    while (fl[j] == 1 && j < n-1) j++;
    i = j -1;
    j = i +1;
  }
}

/*****
  Calcula sinal
*****/
int sgn(float x)
{
  if (x<0) return -1;
  else return 1;
}

/*****
  Calcula norma 2
*****/

```

```

float norm(float *x, int n)
{
    float norma = 0;
    for (int j = 1 ; j < n ; j++)
        norma += x[j]*x[j];

    return sqrt(norma);
}

/*****
Indice da posicao de minimo
*****/
int indexmin(float *x, int n)
{
    int i = 0;
    for (int j=i ; j < n ; j++)
        if (x[j] < x[i])
            i = j;
    return i;
}

/*****
Encontra o ponto de corte otimo a direita de k0
*****/
void fknots(float *x, float *y, int n,
           float smax, int lmin, int k0, int nf,
           float& sot, int& oot, int& kot, float& erro)
{
    float *yy, *direct, *reverse, *scale;
    float s, xa, xb, ya, yb;
    float a,d,c,e,f;
    int *fl,*is;
    int *ik;

    direct = new float [nf - k0 -lmin +1];
    reverse= new float [nf - k0 -lmin +1];
    scale = new float [nf - k0 -lmin +1];

    is = new int [2];
    is[0] = k0;

    for (int i = k0+lmin; i <= nf ; i++)
    {
        // Definie o extremo direito do subintervalo
        // para estimar fator de escala vertical
        is[1] = i;
        s1(x,y,n,is,2,&s);
    }
}

```

```

scale[i-k0-lmin] = s;

// Se fator de escala vertical for aceito...
if (fabs(s) < smax)
{
    // Faz a transformacao para os doi coeficientes de orientacao
    for (int o = -1; o <= 1; o+=2)
    {

        // Cria os vetores da imagen transformada
        yy      = new float [n];
        yy[0]   = y[0];
        yy[n-1] = y[n-1];

        fl      = new int [n];
        fl[0]   = 1;
        fl[n-1] = 1;

        // Zera posicoes ociosas
        for (int j=1 ; j< n-1 ; j++)
            fl[j] = 0;

        ik = new int [i-k0+1];

        if (o == 1)
        {
            ikinv(n,k0,i,1,ik);

            xa = x[k0];
            xb = x[i];
            ya = y[k0];
            yb = y[i];
        }
        else // if o == -1
        {
            ikinv(n,i,k0,-1,ik);

            xa = x[i];
            xb = x[k0];
            ya = y[i];
            yb = y[k0];
        }
        if (fabs(s) < 1e-6)
            s = sgn(s) * 10-6;

        d = 1/(x[n-1] - x[0]);
        a = d *(xb - xa);
        e = d *(x[n-1]*xa - x[0]*xb);
    }
}

```

```

c = d * ((yb-ya) - s*(y[n-1]-y[0]));
f = d * ((x[n-1]*ya - x[0]*yb) - s*(x[n-1]*y[0]-x[0]*y[n-1]));

for (j = k0; j <= i ; j++)
{
    yy[ik[j-k0]] = -c/(a*s) * (x[j]-e) + 1/s * (y[j]-f);
    fl[ik[j-k0]] = 1;
}
delete ik;

// Preenche o vetor com interpolacao linear
datafill(x,yy,fl,n);
delete fl;

// armazena em yy <- yy-y
for (int ii =0 ; ii < n ; ii++)
    yy[ii]-=y[ii];

if (o == 1)
{
    if (norm(y,n) > 0) direct[i-k0-lmin] = norm(yy,n)/norm(y,n);
    else direct[i-k0-lmin] = 1;
}
else
{
    if (norm(y,n) > 0) reverse[i-k0-lmin] = norm(yy,n)/norm(y,n);
    else reverse[i-k0-lmin] = 1;
}

    delete yy;
} // for o=-1:2:1
}
else // if abs(s) >= smax
{
    direct[i-k0-lmin] = 1;
    reverse[i-k0-lmin] = 1;
}
}

int i1 = indexmin(direct,nf-k0-lmin+1);
int i2 = indexmin(reverse,nf-k0-lmin+1);

if (direct[i1] < reverse[i2])
{
    oot = 1;
    kot = i1+k0+lmin;
    erro = direct[i1];
}

```



```
        sot = scale[i1];
    }
    else
    {
        oot = -1;
        kot = i2+k0+lmin;
        erro = reverse[i2];
        sot = scale[i2];
    }

    delete direct;
    delete reverse;
    delete is;
    delete scale;
}

// Programa principal

void main()
{
    float *x, *y, *smax;
    float nn, mm;
    int *sinterv;
    int n, ns;

    //Armazena o tempo atual
    time_t t1 = time(NULL);

    printf("Algoritmo de Selecao de Pontos para o problema de\n");
    printf("Aproximacao Fractal de Funes\n\n");

    FILE *pontos;
    pontos = fopen("data.txt", "rt");

    // Le os vetores que definem os fatores de escala
    // vertical maximos por intervalos

    fscanf(pontos, "%f %f", &nn, &mm);
    ns = int(nn);

    sinterv = new int [ns];
    smax     = new float [ns];

    for (int i=0; i < ns; i++)
    {
        fscanf(pontos, "%f %f", &nn, &smax[i]);
        sinterv[i] = int(nn);
    }
}
```

```

}

// Le a dimensao dos vetores de pontos
fscanf(pontos, "%f %f", &nn, &mm);

n = int(nn);

printf("Dimensao dos vetores de pontos: %i\n\n",n);
printf("Fatores de escala vertical maximos por faixa:\n");

for (i=0; i < ns; i++)
    printf(" -> %5i, |s| < %5.3f\n",sinterv[i]+1,smax[i]);

// Cria os vetores x e y
x = new float [n];
y = new float [n];

// Le os pontos do arquivo
for (i=0; i < n ; i++)
    fscanf(pontos, "%f %f", &x[i], &y[i]);
fclose(pontos);

FILE *nos = fopen("ikso.txt","wt");

int o,nf;
int k0 = 0;
int k1 = 0;
int kaux = 0;
int lmin = 5;
float s;
float erro;

// Grava o extremo esquerdo
fprintf(nos,"%i 0 0\n",k1+1);

while (k1 < n-1-lmin)
{
    if (k0 == 0)
    {
        nf = int (floor(0.8 * float (n-1)));
        printf("\nLimite para busca na iteracao inicial: %i\n ", nf);
    }
    else
        nf = n-1;

    // Descobre o extremo direito do primeiro subintervalo
    fknots(x,y,n,smax[kaux],lmin,k0,nf,s,o,k1,erro);
}

```

```

printf("\n k = %4i | x[k] = %8.3f | ",k1+1,x[k1]);
printf("o = %2i | s = %6.3f (%5.3f) | e = %6.3e",o,s,smax[kaux],erro);

while (k1 > sinterv[kaux])
    kaux++;

if (k1 == sinterv[kaux])
    kaux++;

fprintf(nos,"%i %f %i\n",k1+1,s,o);

k0 = k1;
}

if (k1 < n-1)
    fprintf(nos,"%i 0 1",n);

time_t t2 = time(NULL);
printf("\n\nTempo total gasto: %3.1f min\n", float(t2-t1)/60);

fclose(nos);

// Libera memoria
delete x;
delete y;
delete smax;
delete sinterv;

}

```

sprec Implementação em C do algoritmo de seleção de pontos para o problema de reconstrução fractal de sinais.

```

#include <time.h>
#include <math.h>
#include <stdio.h>

/*****
Calcula a diferenca entre a funcao e a reta que une os extremos
*****/
float height(float *x, float *y, int n)
{
    float m = (y[n-1]-y[0])/(x[n-1]-x[0]);

```

```

float h=0;

for (int i=0; i < n ; i++)
{
float aux = y[i]-y[0] - m*(x[i]-x[0]);
if (fabs(aux)>fabs(h))
h = aux;
}

return h;
}

/*****
Estimador dos fatores de escala vertical
*****/
void s1(float *x, float *y, int n, int *ik, int nk, float *s)
{
float smax = height(x,y,n);

for (int i =0; i < nk-1; i++)
s[i] = height(x+ik[i],y+ik[i],ik[i+1]-ik[i]+1) /smax;
}

/*****
Inverte um conjunto de indices do intervalo [i0,i1] para o grande
*****/
void ikinv(int n, int i0, int i1, int d, int *ik)
{
for (int i =0 ; i <= abs(i1-i0); i++)
if (d==1)
ik[i] = int (floor( (n-1) * float(i) /float(abs(i1-i0))));
else
ik[i0-i1-i] = int (floor( (n-1) * float(i) /float(abs(i1-i0))));
}

/*****
Interpola linearmente os pontos de x,y segundo fl
*****/
void datafill(float *x, float *y, int *fl, int n)
{
int i = 0;
int j = i + 1;
int k;
float m;

while (j < n-1)

```

```

    {
        while (fl[j] == 0 && j < n-1) j++;
        if (j <= n-1)
            m = (y[j] - y[i])/(x[j]-x[i]);

        for (k=i+1 ; k < j ; k++)
            y[k] = y[i] + m * (x[k] - x[i]);

        while (fl[j] == 1 && j < n-1) j++;
        i = j -1;
        j = i +1;
    }
}

/*****
Calcula sinal
*****/
int sgn(float x)
{
    if (x<0) return -1;
    else return 1;
}

/*****
Calcula norma 2
*****/
float norm(float *x, int n)
{
    float norma = 0;
    for (int j = 1 ; j < n ; j++)
        norma += x[j]*x[j];

    return sqrt(norma);
}

/*****
Indice da posicao de minimo
*****/
int indexmin(float *x, int n)
{
    int i = 0;
    for (int j=1 ; j < n ; j++)
        if (x[j] < x[i])
            i = j;
    return i;
}

/*****

```

```

Encontra o ponto de corte otimo a direita de k0
*****/
void fknots(float *x, float *y, int n,
           float smax, int lmin, int k0, int ia, int ib, int nf,
           float& sot,int& oot, int& kot, float& erro)
{
    float *yy, *direct, *reverse, *scale;
    float s, xa, xb, ya, yb;
    float a,d,c,e,f;
    int *fl,*is, *ik;

    direct = new float [nf - k0 -lmin +1];
    reverse= new float [nf - k0 -lmin +1];
    scale  = new float [nf - k0 -lmin +1];

    is     = new int [2];
    is[0] = k0;

    // Define ate onde a busca pelo extremo direito pode ser feita
    int nfad;
    if (k0 <= ia)
        nfad = ia;
    else
        nfad = nf;

    // Inicia busca pelo extremo direito
    for (int i = k0+lmin; i <= nfad ; i++)
    {
        // Definie o extremo direito do subintervalo
        // para estimar fator de escala vertical
        is[1] = i;
        s1(x,y,n,is,2,&s);
        scale[i-k0-lmin] = s;

        // Se fator de escala vertical for aceito...
        if (fabs(s) < smax)
        {
            // Faz a transformacao para os doi coeficientes de orientacao
            for (int o = -1; o <= 1; o+=2)
            {

                // Cria os vetores da imagen transformada
                yy     = new float [n];
                yy[0]  = y[0];
                yy[n-1] = y[n-1];
            }
        }
    }
}

```

```

fl      = new int [n];
fl[0]   = 1;
fl[n-1] = 1;

// Zera posicoes ociosas
for (int j=1 ; j< n-1 ; j++)
    fl[j] = 0;

ik = new int [i-k0+1];

if (o == 1)
{
    ikinv(n,k0,i,1,ik);

    xa = x[k0];
    xb = x[i];
    ya = y[k0];
    yb = y[i];
}
else // if o == -1
{
    ikinv(n,i,k0,-1,ik);

    xa = x[i];
    xb = x[k0];
    ya = y[i];
    yb = y[k0];
}

if (fabs(s)<1e-6)
{
    printf("\nWarning: scale factor less than 1e-6 ( %e )",s);
    s= sgn(s) * 1e-6;
}

d = 1/(x[n-1] - x[0]);
a = d *(xb - xa);
e = d *(x[n-1]*xa - x[0]*xb);
c = d *((yb-ya) - s*(y[n-1]-y[0]));
f = d *((x[n-1]*ya - x[0]*yb) - s*(x[n-1]*y[0]-x[0]*y[n-1]));

for (int q = k0; q <= i ; q++)
{
    yy[ik[q-k0]] = -c/(a*s) * (x[q]-e) + 1/s * (y[q]-f);
    fl[ik[q-k0]] = 1;
}
delete ik;

```

```

// Preenche o vetor com interpolacao linear
datafill(x,yy,fl,n);

delete fl;

// armazena em yy <- yy-y para os pontos fora do intervalo ausente
for (int i1 =0 ; i1 <= ia ; i1++)
    yy[i1]-=y[i1];
for (int i2 =(ia+1) ; i2 <= (ib-1) ; i2++)
    yy[i2]=0;
for (int i3 =ib ; i3 < n ; i3++)
    yy[i3]-=y[i3];

if (o == 1)
    direct[i-k0-lmin] = norm(yy,n)/norm(y,n);
else
    reverse[i-k0-lmin] = norm(yy,n)/norm(y,n);
delete yy;
} // for o=-1:2:1
}
else // if abs(s) >= smax
{
    direct[i-k0-lmin] = 1;
    reverse[i-k0-lmin] = 1;
}
}

int i1 = indexmin(direct,nf-k0-lmin+1);
int i2 = indexmin(reverse,nf-k0-lmin+1);

if (direct[i1] < reverse[i2])
{
    oot = 1;
    kot = i1+k0+lmin;
    erro = direct[i1];
    sot = scale[i1];
}
else
{
    oot = -1;
    kot = i2+k0+lmin;
    erro = reverse[i2];
    sot = scale[i2];
}

delete direct;
delete reverse;
delete is;

```



```
        delete scale;
    }

// Programa principal

void main()
{
    float *x, *y, *smax;
    float nn,mm;
    int n,ns,a,b;
    int *sinterv;

    printf("Algoritmo de Selecao de Pontos para o problema de\n");
    printf("Reconstrucao Fractal de Sinais\n\n");

    FILE *pontos;
    pontos = fopen("data.txt", "rt");

    // Le os vetores que definem os fatores de escala
    // vertical maximos por intervalos

    fscanf(pontos,"%f %f",&nn,&mm);
    ns = int(nn);

    sinterv = new int [ns];
    smax     = new float [ns];

    for (int i=0; i < ns; i++)
    {
        fscanf(pontos,"%f %f",&nn,&smax[i]);
        sinterv[i] = int(nn);
    }

    // Le os indices dos extremos do intervalo ausente
    fscanf(pontos, "%f %f", &nn, &mm);
    a = int(nn);
    b = int(mm);

    // Le a dimensao dos vetores de pontos
    fscanf(pontos, "%f %f", &nn, &mm);
    n = int(nn);

    printf("Dimensao dos vetores de pontos: %i\n\n",n);
    printf("Fatores de escala vertical maximos por faixa:\n");

    for (i=0; i < ns; i++)
```

```

    printf(" -> %5i, |s| < %5.3f\n",sinterv[i]+1,smax[i]);

// Cria os vetores x e y
x = new float [n];
y = new float [n];

// Le os pontos do arquivo
for (i=0; i < n ; i++)
    fscanf(pontos, "%f %f", &x[i], &y[i]);
fclose(pontos);

printf("\nIntervalo ausente -> [ x(%i) , x(%i) ]=[ %8.3f , %8.3f ]\n",
        a+1,b+1,x[a],x[b]);

FILE *nos = fopen("ikso.txt","wt");

// Define o numero minimo de elementos num subintervalo
int lmin = 5;

int o,nf;
int k0 = 0;
int k1 = 0;
int kaux = 0;
float s;
float erro;

// Grava o extremo esquerdo
fprintf(nos,"%i 0 0\n",k1+1);

//Armazena o tempo atual
time_t t1 = time(NULL);

nf = a;
printf("\nLimite para busca na iteracao inicial: %i\n ", nf+1);

while (k1 < a-1-lmin)
{
    // Descobre o extremo direito do primeiro subintervalo
    fknots(x,y,n,smax[kaux],lmin,k0,a,b,nf,s,o,k1,erro);

    while (k1 > sinterv[kaux])
        kaux++;

    if (k1 == sinterv[kaux])
        kaux++;

    printf("\n k = %4i | x[k] = %8.3f | ",k1+1,x[k1]);
    printf("o = %2i | s = %6.3f (%5.3f) | e = %6.3e",o,s,smax[kaux],erro);
}

```

```

    fprintf(nos,"%i %f %i\n",k1+1,s,o);

    k0 = k1;
}

if (k1 < a)
{
    int *is = new int[2];
    is[0]=k1;
    is[1]=a;
    float s;
    s1(x,y,n,is,2,&s);
    k1 = a;

    printf("\n-----");
    printf("-----");
    printf("\n k = %4i | x[k] = %8.3f | ",k1+1,x[k1]);
    printf("o = %2i | s = %6.3f (%5.3f) | e = %6.3e",o,s,smax[kaux],erro);

    fprintf(nos,"%i %f 1\n",k1+1,s);
}

// Recomeca a busca pelo extremo direito do intervalo ausente
k0 = b;
k1 = k0;
while (k1 > sinterv[kaux])
    kaux++;

if (k1 == sinterv[kaux])
    kaux++;

nf = n-1;

// Grava o extremo esquerdo do intervalo ausente
fprintf(nos,"%i 0 0\n",k1+1);

printf("\n-----");
printf("-----");
printf("\n k = %4i | x[k] = %8.3f | ",k1+1,x[k1]);
printf("o = ? | s = ????? (%5.3f) | e = %6.3e",smax[kaux],erro);
printf("\n-----");
printf("-----");

while (k1 < n-1-lmin)
{
    // Descobre o extremo direito do primeiro subintervalo
    fknots(x,y,n,smax[kaux],lmin,k0,a,b,nf,s,o,k1,erro);
}

```

```

printf("\n k = %4i | x[k] = %8.3f | ",k1+1,x[k1]);
printf("o = %2i | s = %6.3f (%5.3f) | e = %6.3e",o,s,smax[kaux],erro);

while (k1 > sinterv[kaux])
    kaux++;

if (k1 == sinterv[kaux])
    kaux++;

fprintf(nos,"%i %f %i\n",k1+1,s,o);

k0 = k1;
}

if (k1 < n-1)
{
    int *is = new int[2];
    is[0]=k1;
    is[1]=n-1;
    float s;
    s1(x,y,n,is,2,&s);
    fprintf(nos,"%i %f 1",n,s);
}

time_t t2 = time(NULL);
printf("\n\nTempo total gasto: %3.1f min\n", float(t2-t1)/60);

fclose(nos);

// Libera memoria
delete x;
delete y;
delete smax;
delete sinterv;
}

```

finds Programa em C, encarregado encontrar o melhor fator de escala vertical para o subintervalo ausente, através de busca.

```

#include <time.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

```

```

/*****
Arredonda um numero real
*****/
int round(float x)

{
    int xi;
    if ((x - floor(x)) >= 0.5)
        xi = ceil(x);
    else
        xi = floor(x);

    return xi;
}

/*****
Gera matriz de contracoes da FIF
*****/
void mkfif(float *xk, float *yk, float *s, int *o, int nk,
          float **FIF )
{

    float deltax = xk[nk] - xk[0];
    float deltay = yk[nk] - yk[0];
    float xa, xb, ya, yb;

    for (int i = 0; i < nk ; i++)
    {
        if (o[i] == 1)
        {
            xa = xk[i];
            xb = xk[i+1];
            ya = yk[i];
            yb = yk[i+1];
        }
        else
        {
            xa = xk[i+1];
            xb = xk[i];
            ya = yk[i+1];
            yb = yk[i];
        }
        FIF[i][0] = ( xb-xa )/deltax;
        FIF[i][1] = ( yb-ya )/deltax - s[i]*deltay/deltax;
        FIF[i][2] = s[i];
        FIF[i][3] = ( xk[nk]*xa - xk[0]*xb )/deltax;
        FIF[i][4] = ( xk[nk]*ya - xk[0]*yb )/deltax -
            s[i]*( xk[nk]*yk[0] - xk[0]*yk[nk] )/deltax;
    }
}

```

```

        FIF[i][5] = 1.0 / nk;
    }

}

/*****
  Interpola linearmente os pontos de x,y segundo fl
  *****/
void datafill(float *x, float *y, int *fl, int n)
{
    int i = 0;
    int j = i + 1;
    int k;
    float m;

    while (j < n-1)
    {
        while (fl[j] == 0 && j < n-1) j++;
        m = (y[j] - y[i]) / (x[j] - x[i]);

        for (k=i+1 ; k < j ; k++)
            y[k] = y[i] + m * (x[k] - x[i]);

        while (fl[j] == 1 && j < n-1) j++;
        i = j - 1;
        j = i + 1;
    }
}

/*****
  Calcula norma um
  *****/
float norm(float *x, int n, int ia, int ib)
{
    float sum = 0;

    for (int i = 0 ; i <= ia ; i++)
        sum += fabs(x[i]);

    for (i = ib ; i < n ; i++)
        sum += fabs(x[i]);

    return sum;
}

```

```

/*****
  Compara o grafico de uma FIF com (x,y)
*****/

float cpfif(float *x, float *y, int nx, int *ik,
            float *xk, float *yk, float **FIF, int nk,
            int ia, int ib)

{

  const int factor = 5;
  float a, w, xx;
  int j, nc, k, knew;

  const float xmin = xk[0];
  const float xmax = xk[nk];

  // Aloca e inicializa os vetores yy e fl
  float *yy = new float [nx];
  int *fl = new int [nx];

  for (int i = 0; i < nx; i++)
  {
    yy[i] = 0;
    fl[i] = 0;
  }

  // Gera vetor da distribuicao acumulada
  float *F = new float [nk];

  F[0] = FIF[0][5];

  for (i = 1; i < nk; i++)
    F[i] = F[i-1] + FIF[i][5];

  F[nk-1]=1;

  // Semente inicial
  xx = xk[0];
  k = 0;

  for (i = 1; i < nx*factor; i++)
  {
    a = float(random(10000)) / 10000.0;

    j = 0;
    while (a > F[j])

```

```

        j=j+1;

nc=j;

w      = FIF[nc][0] * xx + FIF[nc][3];
knew   = round((w-xmin)*(nx-1)/(xmax-xmin));

yy[knew] = FIF[nc][1] * xx + FIF[nc][2] * yy[k] + FIF[nc][4];
fl[knew] = 1;
xx = w;
k = knew;
}

// Forca que os extremos estejam presentes

for (i = 0; i <= nk; i++)
{
    yy[ik[i]] = yk[i];
    fl[ik[i]] = 1;
}

// Completa o grafico
datafill(x, yy, fl, nx);

for (i = 0; i < nx; i++)
    yy[i] = yy[i] - y[i];

float erro = norm(yy,nx,ia,ib);

delete yy;
delete fl;
delete F;

return erro;
}

// Programa principal

void main()
{
    //Armazena o tempo atual
    time_t t1 = time(NULL);

    // Dados de entrada
    float *x, *y, *xk, *yk, *s, *sot;
    int *ik, *o, K, nx, nk, ns;

```



```

// Variaveis auxiliares
float nn, mm;

printf("Reconstrucao Fractal de Sinais");
printf("\nRotina para estimar parametros otimos");

FILE *pontos;
pontos = fopen("posdata.txt", "rt");

/*****
          LE OS VETORES DE PONTOS
*****/

// Le dimensao dos vetores de pontos
fscanf(pontos, "%f %f", &nn, &mm);
nx = int(nn);

// Aloca memoria para o pontos
x = new float [nx];
y = new float [nx];

// Le os pontos
for (int i=0; i < nx ; i++)
    fscanf(pontos, "%f %f", &x[i], &y[i]);

// Le quantidade de subintervalos e posicao do
// subintervalo ausente

fscanf(pontos, "%f %f", &nn, &mm);
nk = int(nn);
K = int(mm);

// Aloca memoria para xk, yk, s e o
ik = new int [nk+1];
xk = new float [nk+1];
yk = new float [nk+1];
s = new float [nk];
o = new int [nk];

// Le o vetor ik de indices dos pontos de corte
for (i=0; i <= nk ; i++)
{
    fscanf(pontos, "%f %f", &nn, &mm);
    ik[i] = (int (nn))-1;
    xk[i] = x[ik[i]];
    yk[i] = y[ik[i]];
}

```

```

int ia = ik[K-1];
int ib = ik[K];

// Le fatores de escala vertical e coeficientes de orientacao
for (i=0; i < nk ; i++)
{
    fscanf(pontos, "%f %f", &s[i], &nn);
    o[i] = int(nn);
}

printf("\n\nDimensao dos vetores de pontos: %5i",nx);
printf("\nQuantidade de subintervalos:      %3i",nk);
printf("\nSubintervalo a reconstruir:      %3i",K);

// Le quantidade de ss a serem testados
fscanf(pontos, "%f %f", &nn, &mm);
ns = int(nn);
printf("\nTotal de teste a fazer:          %2i",ns);

// Aloca memoria e le vetor sot
sot = new float [ns];
for (i=0; i < ns ; i++)
    fscanf(pontos, "%f %f", &sot[i], &mm);

// Fecha arquivo de entrada de dados
fclose(pontos);

/*****
                LEITURA DOS DADOS CONCLUIDA
*****/

// Aloca memoria para a FIF
float **FIF;

FIF = new float *[nk];
for (i = 0; i < nk ; i++)
    FIF[i] = new float [6];

// Aloca memoria para a matrix de erros
float **erros;

erros = new float *[ns];
for (i = 0; i < ns ; i++)
{
    erros[i] = new float [4];
    for (int j = 0; j < 4; j++)

```

```

    erros[i][j] = 0;
}

const int nloops = 3;

printf("\n\nIntervalo ausente: [x(%i),x(%i)] = [%7.4f,%7.4f]",
       ia+1,ib+1,x[ia],x[ib]);

printf("\n\n          +/+   |   -/+   |   +/-   |   -/-");

randomize();

// Testa todos os valores de sot
for (i = 0 ; i < ns ; i++)
{
    for (int loop = 0; loop < nloops ; loop++)
    {
        // Configuracao 1: +/+
        s[K-1] = sot[i];
        o[K-1] = 1;
        mkfif(xk,yk,s,o,nk,FIF);

        erros[i][0] += cpfif(x,y,nx,ik,xk,yk,FIF,nk,ia,ib);

        // Configuracao 2: -/+
        s[K-1] = -sot[i];
        o[K-1] = 1;

        mkfif(xk,yk,s,o,nk,FIF);
        erros[i][1] += cpfif(x,y,nx,ik,xk,yk,FIF,nk,ia,ib);

        // Configuracao 3: +/-
        s[K-1] = sot[i];
        o[K-1] = -1;

        mkfif(xk,yk,s,o,nk,FIF);
        erros[i][2] += cpfif(x,y,nx,ik,xk,yk,FIF,nk,ia,ib);

        // Configuracao 4: -/-
        s[K-1] = -sot[i];
        o[K-1] = -1;

        mkfif(xk,yk,s,o,nk,FIF);
        erros[i][3] += cpfif(x,y,nx,ik,xk,yk,FIF,nk,ia,ib);
    }
    for (int j = 0 ; j < 4; j++)
        erros[i][j] = erros[i][j] / nloops;
    printf("\n %6.4f -> %9.4f | %9.4f | %9.4f | %9.4f",sot[i],

```

```
        erros[i][0],erros[i][1],erros[i][2],erros[i][3]);
    }

    int ii = 0;
    int jj = 0;

    for (i = 0; i < ns; i++)
        for (int j = 0; j < 4; j++)
            if (erros[i][j] < erros[ii][jj])
                {
                    ii = i;
                    jj = j;
                }

    printf("\n\nns = %7.4f e configuracao = %1i",sot[ii],jj+1);

    time_t t2 = time(NULL);
    printf("\n\nTempo total gasto: %3.1f min\n", float(t2-t1)/60);

    // Libera memoria
    delete x;
    delete y;
    delete ik;
    delete xk;
    delete yk;
    delete s;
    delete o;
    delete sot;
    delete FIF;
    delete erros;

}
```

Bibliografia

- [1] M. F. BARNESLEY, *Fractal Functions and Interpolation*, Constructive Approximation, 2 (1986), 303–329.
- [2] M. F. BARNESLEY, *Fractals Everywhere*, Academy Press, 1988.
- [3] M. F. BARNESLEY AND S. G. DEMKO, *Iterated Function Systems and the Global Construction of Fractals*, Proceedings of the Royal Society of London, A399 (1985), 243–275.
- [4] M. F. BARNESLEY, V. ERVIN, D. HARDIN AND J. LANCASTER, *Solution of an Inverse Problem for Fractals and Other Sets*, Proc. Natl. Acad. Sci. USA, 83 (Mathematics) (1986), 1975–1977.
- [5] M. A. BERGER, *Random Affine Iterated Function Systems: Curves Generation and Wavelets*, SIAM Review, 34 (1992), 361–385.
- [6] R. C. A. BILOTI E L. T. SANTOS, *Estudo de Fractais Gerados por Iterações de Ponto Fixo*, Relatório Técnico IMECC–Unicamp, RP 34/94, julho, 1994.
- [7] S. D. CONTE & C. DE BOOR, *Elementary Numerical Analysis: An Algorithmic Approach*, Mathematical and Statistics Series, McGraw-Hill, 3rd ed., 1987.
- [8] G. A. EDGAR, *Measure, Topology and Fractal Geometry*, Springer-Verlag, 1990.
- [9] J. HUTCHINSON, *Fractals and Self-Similarity*, Indiana University Journal of Mathematics, 30 (1981), 731–747.
- [10] W. RUDIN, *Real and Complex Analysis*, McGraw-Hill, 1966.