

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

APROXIMAÇÃO DE NUVENS DE PONTOS DE
DADOS POR MEIO DE SUPERFÍCIES DE
BÉZIER

Alexsandro Santos Soares

Uberlândia
fevereiro de 2007

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

APROXIMAÇÃO DE NUVENS DE PONTOS DE
DADOS POR MEIO DE SUPERFÍCIES DE
BÉZIER

Alexsandro Santos Soares

Tese apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Doutor em Ciências.

Área de concentração: Processamento da Informação.

Linha de pesquisa: Inteligência Artificial.

Orientador: Prof. Dr. Antonio Eduardo Costa Pereira

Uberlândia
fevereiro de 2007

Dados Internacionais de Catalogação na Publicação (CIP)

S676a Soares, Alexsandro Santos.

Aproximação de nuvens de pontos de dados por meio de superfícies de Bézier : uma aplicação em geofísica / Alexsandro Santos Soares. - 2007.

240 p. : il.

Orientador: Antonio Eduardo Costa Pereira.

Tese (doutorado) – Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica.

Inclui bibliografia.

1. Processamento eletrônico de dados - Teses. 2. Geofísica - Teses. I. Pereira, Antonio Eduardo Costa. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU: 681.3.02

Alexsandro Santos Soares

Aproximação de nuvens de pontos de dados por meio de superfícies de Bézier

Tese apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Doutor em Ciências.

Área de concentração: Processamento da Informação.

Linha de pesquisa: Inteligência Artificial.

Uberlândia, 23 de fevereiro de 2007

Composição da Banca Examinadora:

Prof. Dr.	Antonio Eduardo Costa Pereira	Orientador	-	UFU
Prof. Dr.	Jonas Rodrigues de Souza	Membro Externo	-	INPE
Prof. Dr.	Severino Luiz Guimarães Dutra	Membro Externo	-	INPE
Prof. Dr.	Keiji Yamanaka	Membro	-	UFU
Profa. Dra.	Rita Maria da Silva Julia	Membro	-	UFU

A minha esposa Elzilaine,
a minha filha Amanda,
pela motivação,
carinho e compreensão.

Agradecimentos

Ao professor Antonio Eduardo Costa Pereira, pela amizade, incentivo e entusiasmo durante os vários anos de jornada e pela confiança depositada em mim para a realização deste trabalho.

Ao professor Bela Fejer, pelo incentivo, entusiasmo e confiança.

Aos professores Keiji Yamanaka e Luciano Vieira Lima, pela amizade e pelas longas conversas no curso deste doutoramento.

Aos amigos Alexandre Grings e Reny Cury, pelo companheirismo.

À Marli Junqueira Buzzi, pela simpatia e presteza costumeira na secretaria da pós-graduação.

Veritatis simplex oratio est
(*A verdade dispensa enfeites*) — SÊNECA

Resumo

A modelagem computacional empírica de fenômenos geofísicos, tais como a derivas zonais de plasma na região F, é uma ferramenta fundamental para a compreensão e previsão de efeitos que exercem grande impacto sócio-econômico nas atividades humanas. Este trabalho propõe, desenvolve e implementa ferramentas numéricas, baseadas em técnicas da modelagem geométrica auxiliada por computador, para as derivas zonais de plasma na região F sobre Jicamarca, no Peru. Dentre os modelos desenvolvidos, o de maior sucesso utiliza as aproximações de curvas de Bézier com o método dos mínimos quadrados. Outros modelos usados, como a teoria de ondaletas e os filtros de Kalman e minimax, também foram tentados, mas sem o mesmo sucesso. A fonte principal dos dados foi o radar de espalhamento incoerente instalado no Rádio-Observatório de Jicamarca. Estes dados sofreram pré-processamento, objetivando o tratamento de erros e uma melhoria de sua qualidade estatística. Depois de tratados, os dados formaram a base para a construção dos modelos matemáticos de tempo calmo e perturbado das derivas zonais. Os modelos foram implementados na linguagem de programação funcional Objective Caml, que se mostrou eficiente, concisa e, assim, adequada para a área de computação numérica.

Palavras-Chave: Geofísica Espacial, modelagem auxiliada por computador, curvas de Bézier, polinômios de Bernstein, ondaletas, filtro de Kalman

Abstract

The computational and empirical modelling of geophysical phenomena, such as F region zonal plasma drifts, is a basic tool for the understanding and forecasting of effects that have great social-economic impact on the human activities. This work proposes, develops and implements a model, based on Computer Aided Geometric Design techniques, to F region zonal plasma drifts over Jicamarca, Peru. Amongst the developed models, that which shows itself as having the greatest success uses Bézier curve fitting with least-square methods. Other models used, such as wavelet theory, minimax and Kalman filters, had also been attempted, but without the same success. The main data source was the incoherent scatter radar measurements installed in the Jicamarca Radio Observatory. This data was preprocessed in order to carry out error handling and to improve its statistical quality. The preprocessed data formed the basis to the quiet and disturbed time zonal drifts models construction. The models had been implemented in the functional programming language Objective Caml, an efficient, concise and well tailored programming language in numerical calculus.

Keywords: Spatial Geophysics, computer aided geometric design, Bézier curves, Bernstein polynomials, wavelets, Kalman filter

Sumário

LISTA DE FIGURAS	xix
LISTA DE ABREVIATURAS E SIGLAS	xxiii
LISTA DE SÍMBOLOS	xxv
1 INTRODUÇÃO	27
1.1 Objetivos	28
1.2 Motivação	28
1.3 Organização do trabalho	29
2 CONCEITOS DE GEOFÍSICA ESPACIAL	31
2.1 Plasma	31
2.2 Física solar	34
2.2.1 A estrutura do Sol	34
2.2.2 Ciclo de atividade solar	36
2.2.3 Explosões solares	36
2.2.4 Ejeções de massa coronal	38
2.2.5 Vento solar	39
2.2.6 Índice de fluxo solar $F_{10,7}$ cm	39

2.3	O campo magnético da Terra e os efeitos da magnetosfera e da ionosfera	40
2.3.1	Magnetômetros	43
2.3.1.1	Magnetômetro de prótons	44
2.3.1.2	Inclinômetro de indução	44
2.3.1.3	Magnetômetro de <i>fluxgate</i>	44
2.3.2	Magnetosfera	45
2.3.3	Ionosfera	47
2.3.4	Variação diurna	49
2.3.5	Índices geomagnéticos	50
2.3.6	Tempestades magnéticas	51
2.4	Deriva equatorial do plasma ionosférico	51
2.4.1	Dínamo da região F	52
2.4.2	Dínamo da região E	53
2.4.3	Radar de Espalhamento Incoerente	54
3	FILTROS DE KALMAN E MINIMAX	55
3.1	Filtro de Kalman	55
3.1.1	Propriedades	65
3.2	Filtro Minimax	66
3.2.1	Filtragem H_∞	66
3.3	Considerações finais	69
4	ONDALETAS	71
4.1	A transformada de Fourier	71
4.2	Transformada de Fourier de curta duração	73
4.3	Análise de multirresolução	75

4.4	Transformada de ondaletas contínua	76
4.5	Transformada de ondaletas discreta	80
4.5.1	Exemplo de cálculo da transformada de ondaletas discreta de Haar	81
4.5.2	Ondaletas e os bancos de filtros	82
4.5.3	Bancos de filtros ortogonais e bases de ondaletas	87
4.6	Combinando filtro de Kalman com ondaletas	89
4.7	O esquema de lifting para ondaletas	98
5	APROXIMAÇÃO DE CURVAS E SUPERFÍCIES	103
5.1	Polinômios de Bernstein	104
5.1.1	Propriedades dos polinômios de Bernstein	105
5.2	Curvas de Bézier	106
5.2.1	Parametrização de curvas	107
5.2.2	Aproximação global de curvas	108
5.2.2.1	Resolução pelas equações normais	109
5.2.2.2	Suavização da curva de ajuste	111
5.3	Superfícies de Bézier	114
5.3.1	Superfícies definidas pelo produto tensorial de polinômios de Bernstein	115
5.3.2	Parâmetros para superfícies	116
5.3.3	Aproximação global de superfícies	118
6	MODELAGEM DA DERIVA ZONAL DO PLASMA NA REGIÃO F	123
6.1	Medições e dados	124
6.2	Formulação matemática	125
6.2.1	Derivação dos modelos	129
7	RESULTADOS E DISCUSSÃO	131

7.1	Preparação dos dados	131
7.2	Filtragem com ondaletas	134
7.2.1	Aproximação com curvas de Bézier e mínimos quadrados	138
7.2.1.1	Modelo de derivas zonais em épocas geomagneticamente calmas	138
7.2.1.2	Modelo de derivas zonais em épocas geomagneticamente perturbadas	139
7.3	Detalhes de implementação	144
8	CONCLUSÃO	147
8.1	Trabalhos futuros	149
	REFERÊNCIAS BIBLIOGRÁFICAS	151
	APÊNDICE A – CLIMATOLOGY OF F REGION ZONAL PLASMA DRIFTS OVER JICAMARCA	155
	APÊNDICE B – CÓDIGO PARA A CURVA DE BÉZIER COM MÍNIMOS QUADRADOS	167
	APÊNDICE C – CÓDIGO PARA A SUPERFÍCIE DE BÉZIER	179
C.1	Bernstein3D.ml	179
	APÊNDICE D – CÓDIGO PARA AS ONDALETAS	189
D.1	Util.mli	190
D.2	Util.ml	190
D.3	Lu.mli	191
D.4	Lu.ml	191
D.5	Neville.mli	195
D.6	Neville.ml	195

D.7	Filter.mli	196
D.8	Filter.ml	196
D.9	Lift.mli	198
D.10	Lift.ml	198
D.11	Mallat.mli	205
D.12	Mallat.ml	206
D.13	Flwt.mli	211
D.14	Flwt.ml	211
D.15	Ondaleta.ml	232

Lista de Figuras

FIGURA 2.1 – Matéria no estado gasoso e no estado de plasma.	32
FIGURA 2.2 – Movimento de um elétron e um íon em um campo magnético.	33
FIGURA 2.3 – Ilustração da estrutura do Sol	34
FIGURA 2.4 – Camadas internas do Sol.	36
FIGURA 2.5 – Comparação das dimensões de uma CME com a Terra.	38
FIGURA 2.6 – O vento solar e sua interação com a magnetosfera terrestre.	39
FIGURA 2.7 – Camadas distintas do interior da Terra.	41
FIGURA 2.8 – Componentes do campo geomagnético.	42
FIGURA 2.9 – Esquema ilustrativo do magnetômetro de <i>fluxgate</i>	45
FIGURA 2.10 – Estrutura da magnetosfera.	46
FIGURA 2.11 – Cinturão de Van Allen.	47
FIGURA 2.12 – Localização das camadas ionosféricas.	48
FIGURA 2.13 – Tempestade Magnética típica	52
FIGURA 4.1 – A implementação por banco de filtros da transformada discreta de ondaleta.	84
FIGURA 4.2 – Estrutura da transformada de ondaleta: recursivamente dividida em médias e diferenças	100
FIGURA 4.3 – Estrutura da transformada inversa de ondaleta: recursivamente reúne as médias e diferenças	100

FIGURA 5.1 – Aproximação de dados usando curvas de Bézier com suavização pelo método das diferenças segundas	114
FIGURA 5.2 – Disposição dos pontos de controle em uma grade.	116
FIGURA 5.3 – Disposição dos pontos de dados em uma grade.	117
FIGURA 5.4 – Algoritmo para o cálculo dos parâmetros de uma superfície de Bézier.	118
FIGURA 6.1 – Matriz de projeto para o ajuste por mínimos quadrados.	127
FIGURA 7.1 – Fragmento de um arquivo contendo dados de velocidades de deriva vertical, medidas pelo radar de espalhamento incoerente para o dia 12/12/2001	132
FIGURA 7.2 – Fragmento de um arquivo contendo erros de medições, em m/s, da deriva vertical feitas pelo radar de espalhamento incoerente do Rádio-Observatório de Jicamarca para o dia 12/12/2001.	133
FIGURA 7.3 – Fragmento de um arquivo contendo dados de velocidades de deriva zonal, em m/s, medidas pelo radar de espalhamento incoerente do Rádio-Observatório de Jicamarca para o dia 12/12/2001.	133
FIGURA 7.4 – Ondaleta de Haar aplicada à variação vertical da velocidade de deriva zonal.	135
FIGURA 7.5 – Ondaleta CDF(2,2) aplicada à variação vertical da velocidade de deriva zonal.	136
FIGURA 7.6 – Ondaleta CDF(4,2) aplicada à variação vertical da velocidade de deriva zonal.	137
FIGURA 7.7 – Dependência sazonal e do ciclo solar das derivas zonais médias em tempo calmo da região F sobre Jicamarca.	139
FIGURA 7.8 – Comparações das derivas zonais do modelo de tempo calmo em Jicamarca para condições de fluxos solares baixo e moderadamente alto.	140
FIGURA 7.9 – Desvios-padrões da deriva zonal obtidos pelo agrupamento de dados e calculados usando o modelo empírico-analítico.	141

FIGURA 7.10 –Dependência do ciclo solar e da sazonalidade das derivas zonais mé-	
dias em tempos calmo e perturbado sobre Jicamarca.	142
FIGURA 7.11 –Comparações entre as derivas perturbadas modeladas com aquelas	
grupadas.	143

Lista de Abreviaturas e Siglas

AMR	análise multirresolução
CME	coronal mass ejection
$F_{10,7}$	índice de fluxo solar
Dst	disturbance storm time
TF	transformada de Fourier
TFCD	transformada de Fourier de curta duração

Lista de Símbolos

B_i^w	i -ésimo polinômio de Bernstein de grau w
$\mathbf{M}^T$	matriz transposta de $\mathbf{M}$
$\mathbf{P}$	matriz de covariância
$E\{x\}$	valor esperado de x
j	unidade imaginária $\sqrt{-1}$
ψ^*	complexo conjugado de ψ
σ	desvio-padrão
$\min_{x,y,z,\dots} f$	valores das variáveis $x, y, z, \dots$ para os quais f possui o menor valor
$\max_{x,y,z,\dots} f$	valores das variáveis $x, y, z, \dots$ para os quais f possui o maior valor
$\bigcup_j V_j$	união de todos os conjuntos V_j , $j \in \mathbb{Z}$
$\bigcap_j V_j$	intersecção de todos os conjuntos V_j , $j \in \mathbb{Z}$
$ \mathbf{x} - \mathbf{y} $	distância euclideana entre os vetores $\mathbf{x}$ e $\mathbf{y}$

1 Introdução

Pesquisadores de determinada área, por necessidade de alta especialização, são, por vezes, surpreendidos pelo fato de que alguns dos seus problemas técnicos já possuem solução total ou parcial em outras áreas do saber. As técnicas de tratamentos de dados provenientes de outros campos científicos podem ser (e muitas vezes são) de grande aplicabilidade em domínios de conhecimento diferentes daqueles para os quais elas foram desenvolvidas.

A área de modelagem geométrica auxiliada por computador, área da Ciência da Computação, que, normalmente, encontra uso em computação gráfica ou no projeto aerodinâmico de superfícies para veículos, envolve a representação, especificação, projeto, manipulação, exibição e análise de curvas e superfícies de forma livre. Suas técnicas e princípios vêm da análise numérica, da teoria da aproximação, da computação gráfica, do desenho geométrico e mecânico e de outras áreas (BARKSKY, 1981).

Neste trabalho, fez-se uma aplicação das técnicas de modelagem geométrica auxiliada por computador a um problema geofísico, a modelagem das derivas zonais do plasma, que, até então, não havia sido feito. Esta é a principal contribuição desta tese.

1.1 Objetivos

O principal objetivo deste trabalho é o desenvolvimento de ferramentas numéricas para a construção de modelos empíricos voltados para os estudos em clima espacial. Um caso de uma aplicação com sucesso, na modelagem da deriva zonal do plasma ionosférico usando aproximações de curvas de Bézier e métodos dos mínimos quadrados, é apresentado.

Um objetivo secundário deste trabalho é mostrar a utilidade prática das linguagens declarativas, particularmente, as linguagens funcionais, na modelagem numérica de fenômenos físicos.

1.2 Motivação

O estudo dos fenômenos associados à atividade solar está intimamente relacionado com alguns efeitos que ocorrem na Terra.

As ondas de choque produzidas pelas ejeções de massa coronal são capazes de gerar efeitos que vão desde prejuízos nas telecomunicações via satélite ou por ondas de rádio, passando por danos causados em equipamentos eletrônicos, até a produção de blecautes em várias partes do globo. Como as atividades humanas estão cada vez mais dependentes da alta tecnologia, tecnologia esta que sofre os efeitos da atividade solar, torna-se cada vez mais necessária a investigação científica das origens dos fenômenos solares e de sua influência na Terra e no espaço (MILONE *et al.*, 2003; YAMASHITA, 1999).

O estudo dos efeitos das tempestades magnéticas sobre a ionosfera, na região equatorial e de baixa latitude magnética, é de grande interesse para o nosso país, posto que quase todo o território brasileiro fica entre essas faixas de latitudes magnéticas.

As instabilidades de plasma ionosférico, na região F , fazem surgir irregularidades na densidade eletrônica, as quais podem interferir nas ondas transionosféricas, utilizadas em telecomunicações via satélite ou pelos sistemas de posicionamento global (Global Positioning System - GPS), usados em sistemas de navegação aérea, terrestre e marítimo-fluvial. Assim, estudar o comportamento do clima espacial na região F pode tornar possível a previsão da ocorrência de fenômenos, tais como as “bolhas ionosféricas” ou os traços F espalhados, que são eventos noturnos causadores de perda de sinal de rádio com conseqüente interrupção das comunicações (BERTONI, 2004).

1.3 Organização do trabalho

O Capítulo 2 contém uma introdução geral aos conceitos de Geofísica Espacial, necessários para a compreensão da modelagem aqui realizada. Vários fenômenos da interação Sol-Terra são caracterizados juntamente com índices que os medem.

O Capítulo 3 apresenta as definições dos filtros de Kalman e minimax. Estes filtros são úteis na estimação de variáveis de estado dos sistemas lineares e, com alguma perda, dos sistemas não-lineares.

O Capítulo 4 discorre, de forma breve, sobre a teoria das ondaletas. As ondaletas podem ser utilizadas, entre outras aplicações, em aproximações funcionais e também na filtragem. Uma possível integração entre as ondaletas e o filtro de Kalman é vista.

O Capítulo 5 mostra como os polinômios de Bernstein e as curvas e superfícies de Bézier podem ser usadas na aproximação funcional. Este é o capítulo que fundamenta o modelo proposto nesta tese.

O Capítulo 6 descreve como foi construído o modelo empírico para derivas zonais,

usando as idéias expostas no capítulo 5.

No Capítulo 7, são exibidos os resultados da aplicação do modelo construído nos dois capítulos anteriores aos dados do radar de espalhamento incoerente de Jicamarca. Neste capítulo, também são discutidos detalhes da implementação computacional deste modelo.

No Capítulo 8, são apresentadas as conclusões deste trabalho.

Quatro apêndices fazem parte deste trabalho. O Apêndice A é constituído pelo principal artigo gerado por esta investigação. No Apêndice B, é mostrado o código desenvolvido na linguagem Objective Caml para o modelo baseado em curvas de Bézier. No Apêndice C está o código para a modelagem baseada em superfícies de Bézier. No Apêndice D, estão os módulos com os códigos necessários para a compilação das ondaletas.

2 Conceitos de Geofísica Espacial

Este capítulo foi escrito tendo em vista um leitor não familiarizado com os conceitos de geofísica espacial. Ele visa fornecer uma introdução geral aos conceitos necessários para a compreensão do trabalho realizado, sem a pretensão de esgotar o assunto.

2.1 Plasma

A palavra *plasma* é originária do grego $\pi\lambda\alpha\sigma\mu\alpha$, que significa material ou obra moldável. A palavra continua a ser usada com essa acepção para designar materiais moldáveis ou que fluem, como, por exemplo, em “plasma sanguíneo”.

Uma substância submetida a um acréscimo constante de temperatura passa do estado sólido para o líquido, deste para o gasoso e, finalmente, para o estado de plasma (FIG. 2.1). Um plasma é composto por um grande número de elétrons livres, íons e átomos neutros, em proporções variadas, o que o torna um meio condutor de eletricidade (ELIEZER; ELIEZER, 2001).

Os processos de ionização que geram o plasma podem ocorrer devido à passagem de radiação através da matéria ou pelo aquecimento da matéria a altas temperaturas. Ao serem aquecidos, os átomos ganham energia, que é absorvida pelos elétrons e faz com que

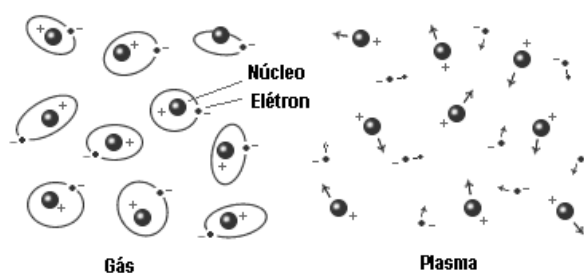


FIGURA 2.1 – Matéria no estado gasoso e no estado de plasma.

Fonte: <http://www.fis.unb.br/plasmas/plasma.htm>.

estes se desprendam dele, criando íons positivos. No plasma, existirão, sobretudo, íons positivos, mas os íons negativos podem também ser formados, se os elétrons liberados ligarem-se a outros átomos ou moléculas.

Um plasma é um sistema quase-neutro. A igualdade aproximada de concentração entre elétrons e íons deve-se ao fato de que forças elétricas atraem cargas opostas. Assim, se, por alguma razão, um grupo de elétrons acumular-se em um ponto particular do plasma, este grupo será, imediatamente, atraído pelos íons positivos, tal que a quase-neutralidade média é satisfeita.

Em circunstâncias apropriadas, as partículas carregadas eletricamente dentro de um plasma podem oscilar coletivamente de maneira organizada. Por exemplo, os elétrons movimentam-se em um comportamento coletivo descrito por ondas, normalmente, chamadas ondas de plasma. As ondas de plasma formam um fenômeno coletivo. A massa de um elétron é muito menor que a massa de um íon. Logo, os elétrons se moverão mais rápidos e com maior facilidade do que os íons.

Em um estado de plasma de equilíbrio, é importante saber o que acontece se um dos parâmetros do plasma é suavemente perturbado. Se a perturbação aumenta, o plasma é *instável*. Por outro lado, se a perturbação diminui suavemente e, então, desaparece, diz-se que o plasma está em *equilíbrio estável* (ELIEZER; ELIEZER, 2001).

Um *fóton* é uma partícula não carregada, de massa zero e que se move à velocidade da luz. Um fóton é um quantum de energia radiante.

Os fótons e elétrons livres, ao colidirem dentro do plasma com os elétrons ligados, fazem com que estes saltem para níveis de maior energia. Quando esses elétrons excitados retornam ao seu nível normal, liberam mais fótons no plasma. Estes fótons são parte da radiação eletromagnética dentro do plasma. Ocasionalmente, alguns desses fótons escapam do plasma, causando perda de energia.

O movimento de partículas carregadas, dentro do plasma, cria um campo eletromagnético. Cada elétron ou íon move-se em torno de uma linha de campo magnético em trajetórias espirais e está confinado à vizinhança desta linha até que seja perturbado (Figura 2.2).

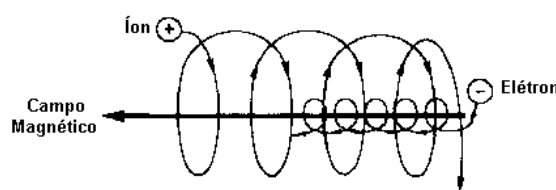


FIGURA 2.2 – Movimento de um elétron e um íon em um campo magnético.

Fonte: Adaptado de (ELIEZER; ELIEZER, 2001).

O plasma é descrito macroscopicamente por sua temperatura e densidade. Alterações no plasma são calculadas por meio das equações de conservação, tais como a conservação de energia, momento e massa. Na escala microscópica, o plasma é descrito estatisticamente, usando-se probabilidades para o cálculo das posições e velocidades de todas as suas partículas. Em decorrência de suas colisões mútuas, as partículas carregadas do plasma emitem radiação.

Do exposto, pode-se concluir que plasma é um sistema quase-neutro, formado por um grande número de partículas carregadas eletricamente, que exibem comportamento

coletivo.

2.2 Física solar

Esta seção faz uma descrição sucinta dos fenômenos envolvendo o Sol, e toda a informação aqui presente, salvo especificação em contrário, advém da referência (MILONE *et al.*, 2003), que o leitor poderá consultar para uma visão pormenorizada sobre o assunto aqui tratado.

2.2.1 A estrutura do Sol

Conceitualmente, pode-se caracterizar o Sol em duas grandes partes (ver Figura 2.3):

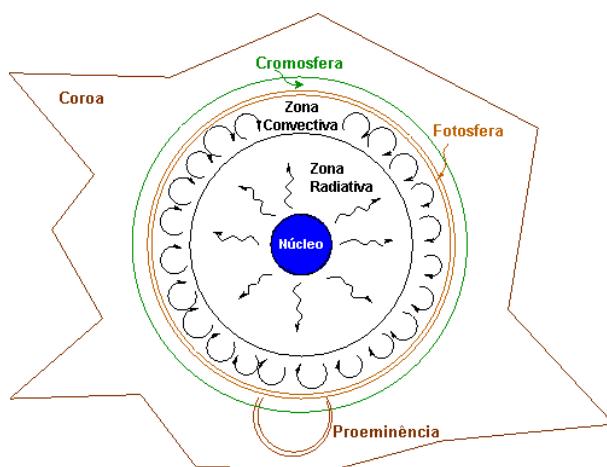


FIGURA 2.3 – Ilustração da estrutura do Sol. A proeminência é criada pela presença de campos magnéticos intensos com pólos na fotosfera.
Fonte: Adaptado de <http://astro.if.ufrgs.br/esol/>

- o *interior solar*, composto de um núcleo, zona radiativa e um envelope convectivo, que pode chegar até a superfície (Figura 2.4). O Sol é uma gigantesca massa de gás ionizado (plasma), cujo núcleo é mantido a pressões e temperaturas altíssimas. Parte da energia produzida pela fusão dos elementos químicos em seu núcleo é

consumida para manter as condições de temperatura e pressão internas, e a outra parte é emitida sob forma de radiação, que escapa pela superfície. O interior do Sol é muito opaco à radiação e, devido ao múltiplo espalhamento que ela sofre, demora cerca de 10^7 anos para chegar à superfície.

- a *atmosfera solar*, composta pela fotosfera, cromosfera e a coroa, em que:
 - a *fotosfera* é uma camada com, aproximadamente, 500 km de espessura e de onde provém a maior parte da luz visível. Nesta camada, podem ser observadas manchas escuras, que surgem e desaparecem em períodos regulares de cerca de 11 anos. Essas manchas solares, que, normalmente, aparecem em grupos, são decorrentes da existência de campos magnéticos intensos na atmosfera solar.
 - a *cromosfera* estende-se até cerca de 10.000 km acima da fotosfera. A fronteira com a coroa é conhecida como região de transição e possui algumas centenas de quilômetros.
 - a *coroa* é a parte mais externa da atmosfera solar e estende-se por milhões de quilômetros a partir do Sol. A sua densidade é de 2 a 3 ordens de grandeza mais baixa que aquela da cromosfera, e sua temperatura está entre 1 a 2 milhões de graus, o que faz com que ela emita grande quantidade de raios-X. Como os gases que a formam se encontram na forma de plasma, produzem-se íons e elétrons que formarão o vento solar. As regiões ativas dessa camada possuem temperatura e densidade elevadas, com campos magnéticos intensos, distribuídos em formato de arcos com os pólos magnéticos situados na fotosfera. Nestas regiões, ocorrem as explosões (*flares*) solares cuja frequência é maior nos períodos de máximo de atividade do ciclo solar.

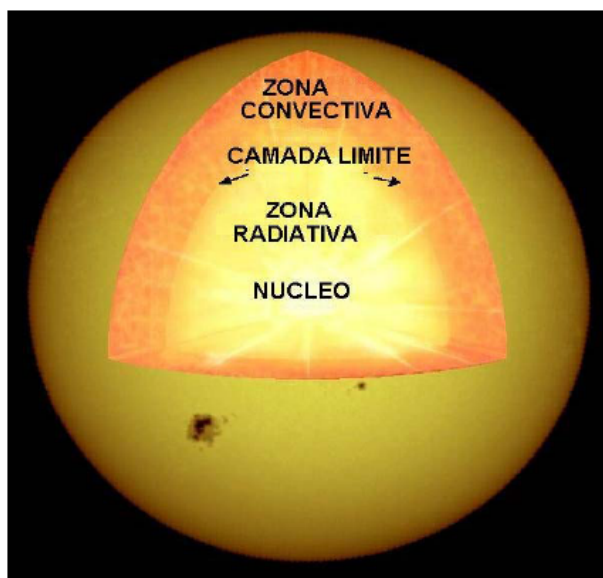


FIGURA 2.4 – Camadas internas do Sol: núcleo, zona radiativa e zona convectiva. O núcleo encontra-se em equilíbrio radioativo.

Fonte: (MILONE *et al.*, 2003)

2.2.2 Ciclo de atividade solar

A atividade solar possui um ciclo médio de 11 anos. No início deste ciclo, surge um reduzido número de pequenas manchas em latitudes solares mais elevadas e à medida que o ciclo se dirige ao seu máximo, as manchas aumentam em tamanho e número, surgindo cada vez mais próximas ao equador solar. Depois, há uma redução gradativa até o fim do ciclo.

A atividade solar possui efeitos causadores de prejuízos em diversas áreas, como nos sistemas de comunicação, navegação e fornecimento de energia, em sistemas biológicos, etc.

2.2.3 Explosões solares

As *explosões solares* caracterizam-se pela emissão de grandes quantidades de energia, em intervalos de tempo, que variam de alguns segundos até poucas horas. A energia

liberada provém da energia magnética armazenada na atmosfera solar dentro das regiões ativas e provoca o aquecimento e a aceleração dos elétrons, prótons e íons presentes nas proximidades do local de liberação. A interação das partículas carregadas com o meio ambiente provoca a emissão de mais energia, sob a forma de radiação, e também pode causar a emissão de partículas energéticas na forma de íons ou elétrons.

As explosões solares podem estar associadas à expulsão de grandes quantidades de matéria a velocidades, que variam entre centenas a poucos milhares de quilômetros por segundo, fenômeno este conhecido como ejeção de massa coronal, que também pode ocorrer dissociado de qualquer explosão solar.

Algumas explosões solares também estão associadas à ocorrência de tempestades magnéticas na Terra, que, dependendo da intensidade, podem causar grandes perturbações na nossa magnetosfera, levando até mesmo a blecautes. Estes blecautes ocorrem quando o campo magnético da Terra captura partículas ionizadas trazidas pelo vento solar, criando correntes elétricas geomagneticamente induzidas que fluem através dos sistemas de potência, entrando e saindo dos muitos pontos de aterramento em uma rede de transmissão. Estas correntes são produzidas quando choques resultantes das tempestades magnéticas repentinas e severas, sujeitam porções da superfície da Terra à flutuações do campo magnético do planeta. Estas flutuações induzem campos elétricos que geram diferenças de potencial elétrico entre os pontos de aterramento, o que faz com que as correntes induzidas fluam através dos transformadores, linhas de potência e os pontos de aterramento. Somente um poucos ampères são necessários para perturbar a operação normal de um transformador, mas valores de até 100 ampères já foram medidos em conexões de aterramento dos transformadores em áreas afetadas (KAPPENMAN; ZANETTI; RADASKY, 1997).

2.2.4 Ejeções de massa coronal

As *ejeções de massa coronal* (CMEs - do inglês, *Coronal Mass Ejections*) são grandes quantidades de matéria (10^{15} a 10^{16} g), entremeadas de linhas de campo magnético, expul-sas do Sol durante um período de várias horas, com velocidades variando entre centenas a poucos milhares de km/s. A frequência de ocorrência das CMEs muda com o ciclo de atividade solar, indo de um evento por semana, no mínimo do ciclo, a dois ou três eventos diários, no máximo. A maioria das CMEs produz ondas de choque que podem atingir a Terra em cerca de dois dias. A Figura 2.5 compara as dimensões de uma CME com a Terra.

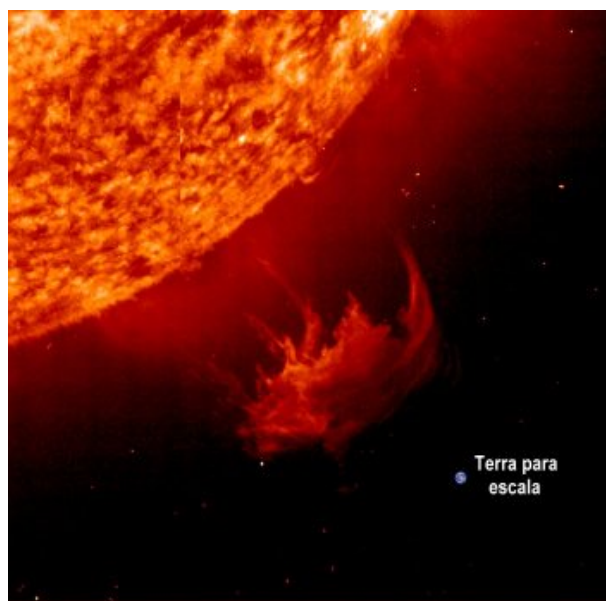


FIGURA 2.5 – Comparação das dimensões de uma CME com a Terra.

Fonte: <http://astro.if.ufrgs.br/esol/>

As CMEs podem alterar o fluxo do vento solar e produzir perturbações na Terra, que trazem danos às atividades humanas. Este fenômeno, em conjunto com as explosões solares, afeta diretamente o chamado “clima espacial”.

2.2.5 Vento solar

Denomina-se *vento solar* o fluxo de elétrons e íons positivos expulsos da coroa solar em alta velocidade (cerca de 450 km/s em pontos afastados de 1 AU do Sol) e que se propaga no meio interplanetário. Esse vento, durante período perturbado, ao se aproximar da Terra, interage com o campo geomagnético formando a magnetosfera. Isto produz deformações no campo geomagnético na direção do eixo Sol-Terra, tanto no espaço situado entre o Sol e a Terra quanto na direção do espaço exterior, como exemplificado na Figura 2.6. O controle exercido por esse campo na região da magnetosfera faz as partículas eletricamente carregadas interagirem com os gases da alta atmosfera, levando, por exemplo, aos fenômenos conhecidos como aurora boreal (no hemisfério norte) e aurora austral (no hemisfério sul).

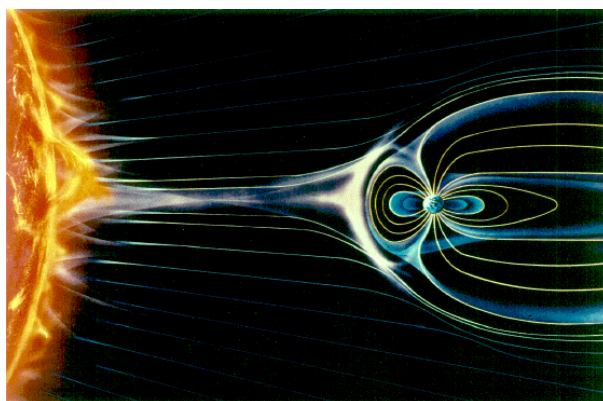


FIGURA 2.6 – O vento solar e sua interação com a magnetosfera terrestre. A ação do vento comprime a magnetosfera em direção ao espaço exterior

Fonte: (MILONE *et al.*, 2003)

2.2.6 Índice de fluxo solar $F_{10,7}$ cm

O fluxo solar é a quantidade de energia proveniente do Sol. Ele é proporcional e flutua de acordo com o número de manchas solares, variando conforme a hora do dia, a época do ano e a posição da Terra em relação ao Sol. O índice de fluxo solar $F_{10,7}$ cm é medido

diariamente na frequência de 2800 MHz (10,7 cm de comprimento de onda) do sinal de rádio emitido pelo Sol. O valor deste índice varia de 60, onde há a ausência de manchas solares, até 300, onde há a presença de muitas manchas solares (WIKIPÉDIA, 2006).

2.3 O campo magnético da Terra e os efeitos da magnetosfera e da ionosfera

A teoria científica mais aceita para explicar a formação do campo geomagnético é a teoria do dínamo auto-sustentável. Por esta teoria, o dínamo interno da Terra funcionaria como um gerador elétrico, que cria campos elétricos e magnéticos a partir da energia cinética de suas partes móveis. Na Terra o movimento ocorre dentro de um fluido eletricamente condutor, na vasta região de material fundido que circula em seu núcleo externo (ver Figura 2.7).

São três as condições básicas para a geração de qualquer campo magnético planetário (GLATZMAIER; OLSON, 2005):

1. Um grande volume de fluido eletricamente condutor, como o núcleo externo da Terra, rico em ferro líquido.
2. Um suprimento de energia para mover o fluido. A energia que alimenta o geodínamo é, em parte, térmica e, em parte, química. O material é mais quente no fundo que no topo. Assim, aquele material mais quente e menos denso tende a subir. Ao alcançar o topo, o líquido perde parte do seu calor para o manto, resfriando-se. O líquido frio torna-se mais denso que o meio circundante e afunda. Esse processo de transferência de calor, do fundo para o topo por ascensão e afundamento de fluidos,

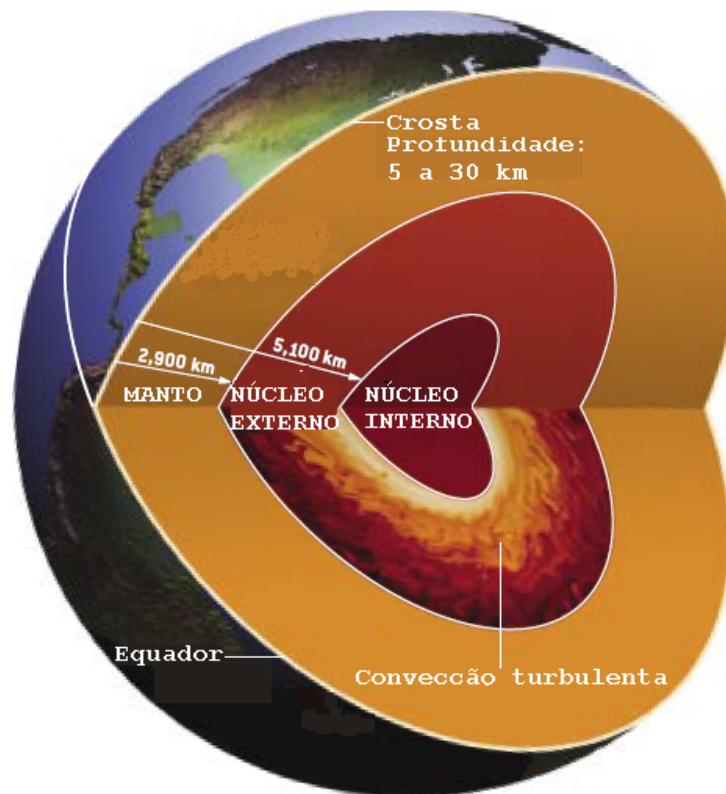


FIGURA 2.7 – As camadas distintas do interior da Terra incluem um núcleo externo líquido, onde complexos padrões de circulação da convecção turbulenta geram o campo geomagnético.

Fonte: Adaptado de (GLATZMAIER; OLSON, 2005)

é chamado de convecção térmica.

3. A rotação. No núcleo, as forças de Coriolis desviam fluidos ascendentes para rotas helicoidais.

Assim, a Terra, que possui um núcleo externo líquido, energia suficiente para conduzir a convecção e uma força de Coriolis para torcer o fluido convectivo, reúne todas as condições para que o geodínamo se autosustente por bilhões de anos.

O campo magnético da Terra é vetorial e, como tal, pode ser descrito por uma amplitude e por dois ângulos - declinação e inclinação - ou pela descrição de seus três componentes num referencial conhecido. É comum a adoção dos eixos geográficos como um referencial cartesiano em que o eixo x é positivo para o norte, o eixo y é positivo para o

leste e o eixo vertical z é positivo para baixo (ver Figura 2.8). O norte magnético local é dado pela direção da componente horizontal.

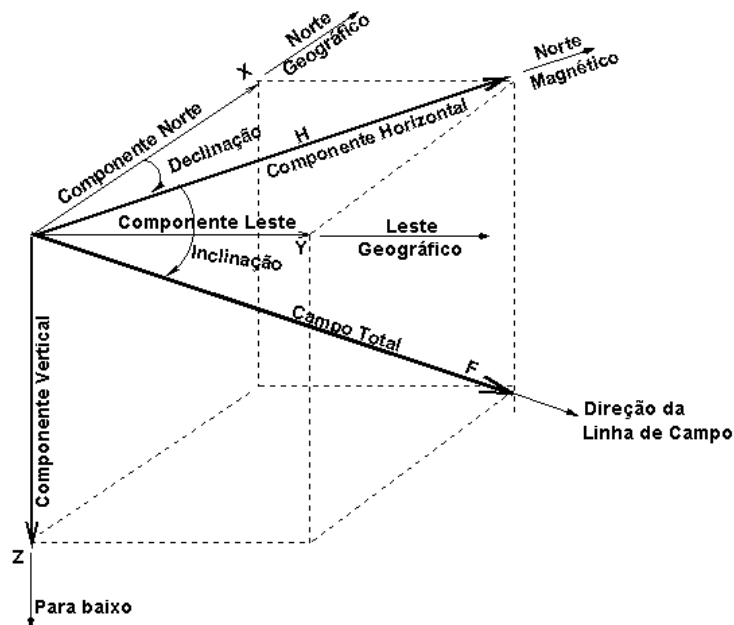


FIGURA 2.8 – Componentes do campo geomagnético.
Fonte: Adaptado de (CAMPBELL, 2003)

A *declinação magnética* é a diferença entre as direções do norte geográfico e do norte magnético. Esse ângulo varia com a localização geográfica, com a altitude (muito pouco) e com o tempo (GOMES, 2004).

A *inclinação magnética* é definida como o ângulo formado entre as linhas de força do campo magnético com o plano horizontal no local de observação. O *equador magnético* é definido como a linha imaginária sobre a qual a inclinação é nula.

As componentes X e Y são usadas nas definições da componente horizontal H e da componente vertical Z . A relação entre estas grandezas e os ângulos de declinação (D) e inclinação (I) pode ser expressa matematicamente por (MIRANDA, 2002):

$$\begin{aligned}H &= \sqrt{X^2 + Y^2} \\D &= \arctg \frac{Y}{X} \\I &= \arctg \frac{Z}{H}\end{aligned}\tag{2.1}$$

A observação contínua do campo geomagnético mostra que as médias anuais de seus componentes apresentam uma variação lenta, mas significativa, espelhando a escala temporal dos fenômenos físicos que ocorrem no núcleo líquido da Terra. Este fenômeno é conhecido como *variação secular* e possui como uma de suas conseqüências a variação da declinação magnética ao longo dos anos (MIRANDA, 2002).

2.3.1 Magnetômetros

O campo geomagnético é medido com o emprego de magnetômetros, classificados como (MIRANDA, 2002):

absolutos se medem a declinação e a inclinação e/ou a intensidade do campo magnético por meio de medidas de massa, comprimento, tempo, intensidade de corrente elétrica ou que recorrem a fenômenos como a ressonância magnética nuclear.

relativos se precisam ser regularmente calibrados por comparação aos instrumentos absolutos. Um exemplo é o QHM (*Quartz Horizontal Magnetometer*), usado na medição da componente horizontal do campo geomagnético.

variógrafos se medem apenas a variação temporal do campo geomagnético.

2.3.1.1 Magnetômetro de prótons

O magnetômetro de prótons é um instrumento absoluto, que mede a intensidade do campo geomagnético. Seu funcionamento baseia-se na ressonância magnética dos núcleos dos átomos de hidrogênio ou de césio, quando submetidos a um campo magnético ambiente.

2.3.1.2 Inclinômetro de indução

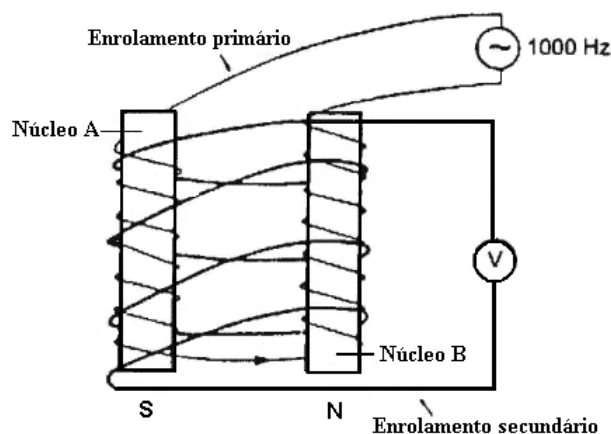
O inclinômetro de indução é um equipamento absoluto, que mede, simultaneamente, a declinação e a inclinação. Ele se baseia no fato de que a corrente induzida em uma bobina, que gira transversalmente em um eixo colinear (localmente) com o campo geomagnético, é nula.

2.3.1.3 Magnetômetro de *fluxgate*

O magnetômetro de *fluxgate* é um magnetômetro vetorial, que, se adequadamente disposto, pode medir de forma independente as três componentes do campo geomagnético.

O sensor desse instrumento é constituído em muitos equipamentos por dois núcleos paralelos de um material com permeabilidade magnética elevada, em torno dos quais dois enrolamentos, primário e secundário, são trançados em sentidos contrários (ver Figura 2.9).

Quando uma corrente alternada é aplicada ao enrolamento primário, induz-se um campo magnético, que leva cada núcleo à saturação. No enrolamento secundário, também é induzido um campo magnético com igual direção, mas com sentido contrário ao primeiro. Na ausência de um campo externo, a corrente aplicada é escolhida de forma que os núcleos

FIGURA 2.9 – Esquema ilustrativo do magnetômetro de *fluxgate*.Fonte: Adaptado de <http://pkukmweb.ukm.my/~rahim>

saturem a cada meio ciclo de corrente, tal que o sinal de tensão observado no enrolamento secundário é nulo.

Quando um campo magnético exterior é aplicado em direção não perpendicular ao campo do núcleo, haverá um reforço desse campo. Neste caso, um dos núcleos satura antes do outro, gerando um sinal de tensão, cuja amplitude é proporcional à componente do campo exterior, que é colinear com o núcleo do sensor.

2.3.2 Magnetosfera

A região mais externa do campo geomagnético é constantemente afetada pelo vento solar. Isto faz com que as linhas de força do campo sejam comprimidas no lado voltado para o Sol e espichadas no lado oposto, formando uma cavidade onde a penetração do vento solar é muito reduzida. A esta cavidade dá-se o nome de *magnetosfera* (MIRANDA, 2002). A magnetopausa é a fronteira entre o vento solar e o campo magnético da Terra e apresenta uma forma alongada, semelhante à cauda de um cometa (Figura 2.10).

Em sua fronteira voltada para o Sol, a magnetosfera recebe o vento solar, em que são geradas ondas de choque devidas às diferenças entre a velocidade do vento solar

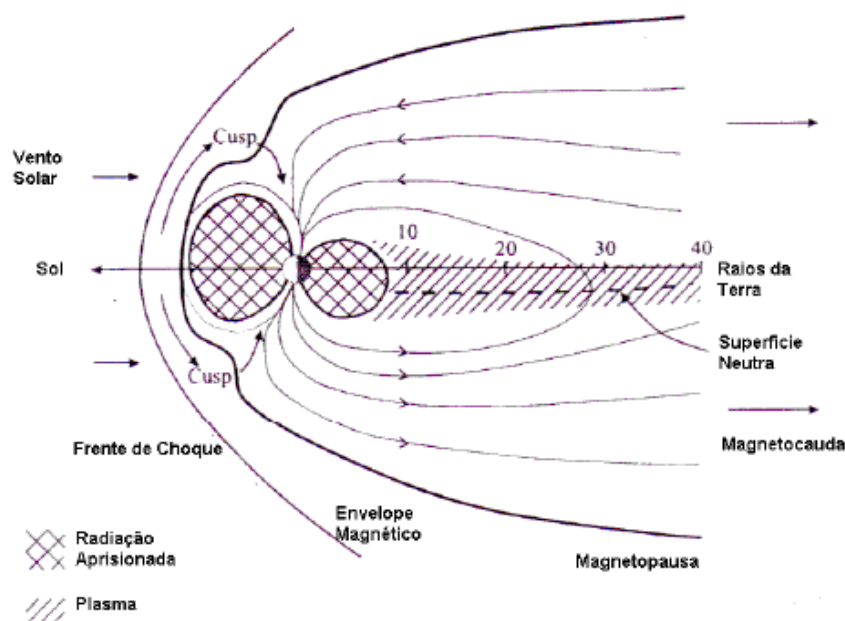


FIGURA 2.10 – Estrutura da magnetosfera.

Fonte: (MIRANDA, 2002)

e a velocidade do som para o plasma interplanetário. Mais próximo da Terra, (1) os movimentos em espiral de partículas carregadas eletricamente em torno das linhas do campo geomagnético, (2) entre os pontos conjugados nos hemisférios norte e sul e (3) de deriva longitudinal no sentido leste para oeste, contribuem para a formação dos *cinturões de radiação Van Allen* (Figura 2.11). A circulação equatorial no cinturão dá origem a uma corrente elétrica, denominada **anel de corrente**, que origina uma parte do campo magnético medido à superfície (MIRANDA, 2002).

Na região compreendida entre a frente de choque e a magnetopausa, as partículas do vento solar desaceleram, até atingir velocidades sub-magnetosônicas. Próximo aos polos magnéticos, verifica-se a penetração do vento solar que pode atingir a ionosfera. O estiramento das linhas de força do campo geomagnético, na magnetocauda, gera o aparecimento de uma superfície *neutra*, do lado escuro da Terra, onde o campo é praticamente nulo (MIRANDA, 2002).

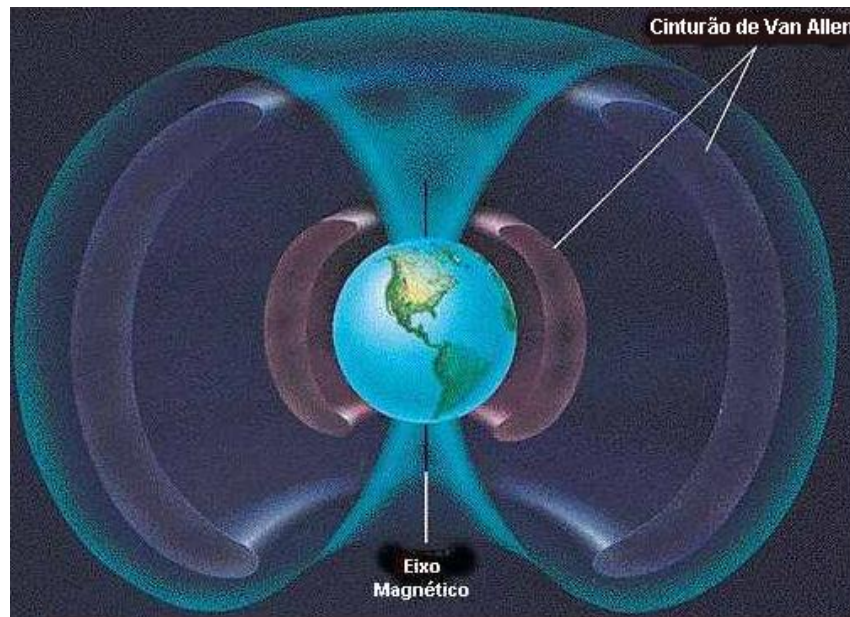


FIGURA 2.11 – O cinturão de Van Allen. A faixa interior se situa entre 600 e 15.000 quilômetros e a exterior, entre 13.000 e 51.000 quilômetros da superfície da Terra.

Fonte: <http://br.geocities.com/saladefisica5/leituras/vanallen.htm>

2.3.3 Ionosfera

A ionosfera é a parte da atmosfera onde existe alta concentração de elétrons e íons positivos livres. Essa camada está situada, aproximadamente, entre 60 km e 1000 km de altura. Ela se divide em três regiões, que correspondem a três faixas de altura, nas quais ocorrem máximos do perfil de densidade eletrônica (YAMASHITA, 1999):

- A região D, localizada entre 60 km e 95 km de altura, é a menos densa das regiões ionosféricas, deixando de existir durante a noite. Essa região é responsável pela absorção das ondas de rádio em alta frequência, que são refletidas nas camadas superiores.
- A região E, localizada entre 95 km e 180 km de altura, subdivide-se em: camada E, normalmente, encontrada; camada E2, mais densa que a camada E; e camada Es (esporádica), pouco densa e instável. A região E é importante pela presença de

correntes elétricas que nela fluem e por sua interação com o campo magnético.

- A região F, localizada acima de 180 km de altura, é onde se concentram as camadas refletoras mais importantes: a camada F1, encontrada esporadicamente; a camada F2, normalmente, encontrada; e a camada F3 na região equatorial.

A nomenclatura das regiões ionosféricas, em função da densidade eletrônica, está indicada na Figura 2.12. As alturas e as densidades numéricas são apenas aproximações e os valores numéricos reais podem variar por diversos fatores, entre eles: a localização, estação do ano, tempo local, atividade magnética etc.

A concentração eletrônica se torna muito pequena abaixo de 70 km e também acima de 1000 km, onde começam a predominar os íons de hidrogênio (prótons).

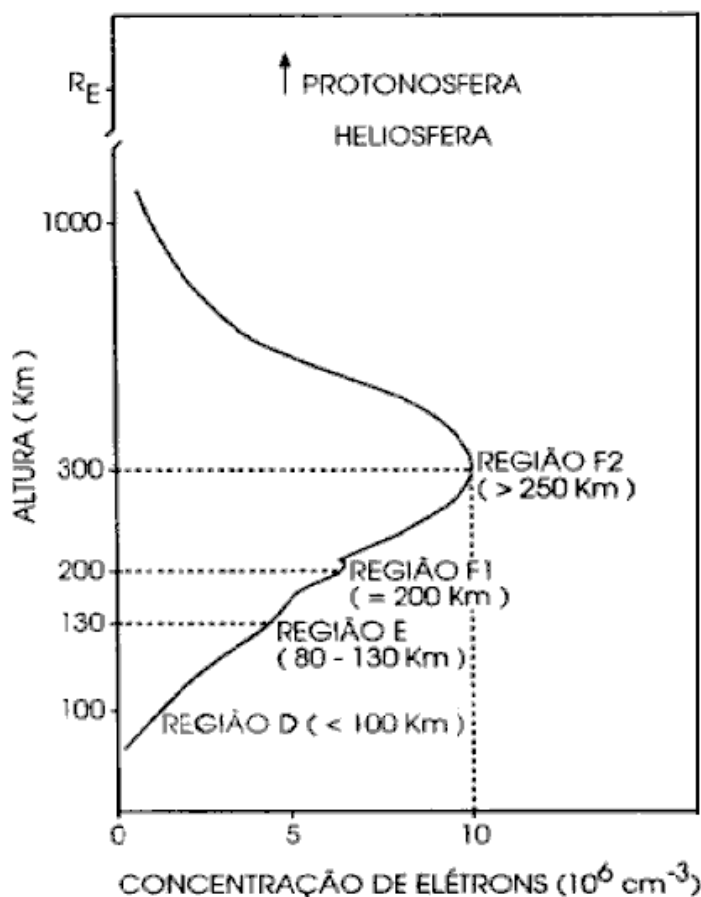


FIGURA 2.12 – Localização das camadas ionosféricas.
Fonte: (YAMASHITA, 1999)

2.3.4 Variação diurna

Na atmosfera, existe um efeito de maré gerado pelo fato de o eixo do dipolo geomagnético estar inclinado em relação à direção do vento produzido por aquecimento solar. Este vento e essa geometria do campo produzem correntes elétricas na região E ionosférica, conseqüentemente, causa variação no próprio campo geomagnético. Tal variação é chamada de *variação diurna*.

A variação diurna pode ser conceitualmente decomposta em uma componente solar **S**, com um período de 24 horas; em uma componente lunar **L**, com período próximo das 25 horas; e em uma componente perturbadora **D**. Ao se considerar apenas os dias magneticamente calmos, obtém-se a variação devida apenas à variabilidade solar S_q . Tanto S_q quanto **L** originam-se em fenômenos de maré. A maré atmosférica solar é essencialmente térmica, enquanto a lunar é essencialmente gravitacional (MIRANDA, 2002).

Em uma banda de cerca de 2° de largura em torno do equador magnético, ocorre uma duplicação de S_q . Tal fato se deve à existência de uma corrente, denominada de *eletrojato equatorial*, que flui de leste para oeste no hemisfério diurno.

Pode-se definir o campo perturbado **D** pela diferença:

$$D = \Delta F - S_q - L \quad (2.2)$$

em que ΔF é a diferença entre o valor medido do campo total em cada instante e o valor médio mensal.

2.3.5 Índices geomagnéticos

Os índices geomagnéticos servem para resumir aspectos interessantes do comportamento do campo magnético da Terra. Os índices de maior interesse para este trabalho são os seguintes (YAMASHITA, 1999):

AE : índice do eletrojato auroral, este índice é obtido por meio de uma cadeia de magnetômetros dispostos globalmente na zona do eletrojato auroral e consiste na diferença entre os invólucros superior (*AU*) e inferior (*AL*) dos registros magnetométricos. Sua intensidade é proporcional ao depósito de energia cinética das partículas que se precipitam na região auroral.

K_p : o índice *K_p* é um número proporcional ao grau de perturbação global do campo magnético terrestre, obtido periodicamente por alguns observatórios espaciais internacionais. Esse índice é uma média de valores obtidos no espaço de tempo de três horas, começando à zero hora. A soma dos oito valores diários constitui o que se denomina ΣK_p .

Dst : Disturbance Storm Time, este índice representa o invólucro das curvas magnetométricas obtidas por uma cadeia de magnetômetros localizados na região equatorial, ao longo do globo terrestre. Esse índice representa a intensidade das correntes de anel nos períodos perturbados e, com ele, define-se uma tempestade magnética. A curva do índice *Dst* representa a variação média do campo geomagnético, na região equatorial, durante as tempestades magnéticas, ocasião em que ocorre a injeção de partículas no cinturão de radiação de Van Allen.

2.3.6 Tempestades magnéticas

As *tempestades magnéticas* são caracterizadas por oscilações muito intensas do campo geomagnético, medidas pelos índices Dst e K_p , podendo durar alguns dias (Figura 2.13). Elas se iniciam por uma elevação brusca da componente X , seguida, minutos depois, por uma descida repentina desta componente para um valor inferior ao valor médio antes da tempestade. Em seguida, assiste-se a uma fase de recuperação da tempestade, correspondendo à modulação em amplitude do anel de corrente, que, por sua vez, corresponde à deriva de íons positivos e negativos, em sentidos contrários, ao longo de uma trajetória fechada, tal como ocorre nos cinturões de Van Allen (MIRANDA, 2002).

Durante as tempestades magnéticas, o índice K_p aumenta, adquirindo elevados valores em dias de forte tempestade. Em contrapartida, nos dias magneticamente calmos, o valor de K_p é relativamente baixo. Essas tempestades causam variação na ionização ionosférica em relação aos períodos calmos e podem fazer a ionosfera subir ou descer (YAMASHITA, 1999).

2.4 Deriva equatorial do plasma ionosférico

Os principais agentes de transporte do plasma da ionosfera equatorial são os campos elétricos ($\mathbf{E}$) na presença do campo geomagnético ($\mathbf{B}$). Esses campos causam derivas zonais e verticais do plasma, as quais podem ser determinadas por $\mathbf{E} \times \mathbf{B}/B^2$. Para períodos geomagneticamente calmos, a origem dos campos elétricos ionosféricos pode ser explicada por meio das teorias dos dínamos das regiões E e F (KELLEY, 1989; CARRASCO, 2005).

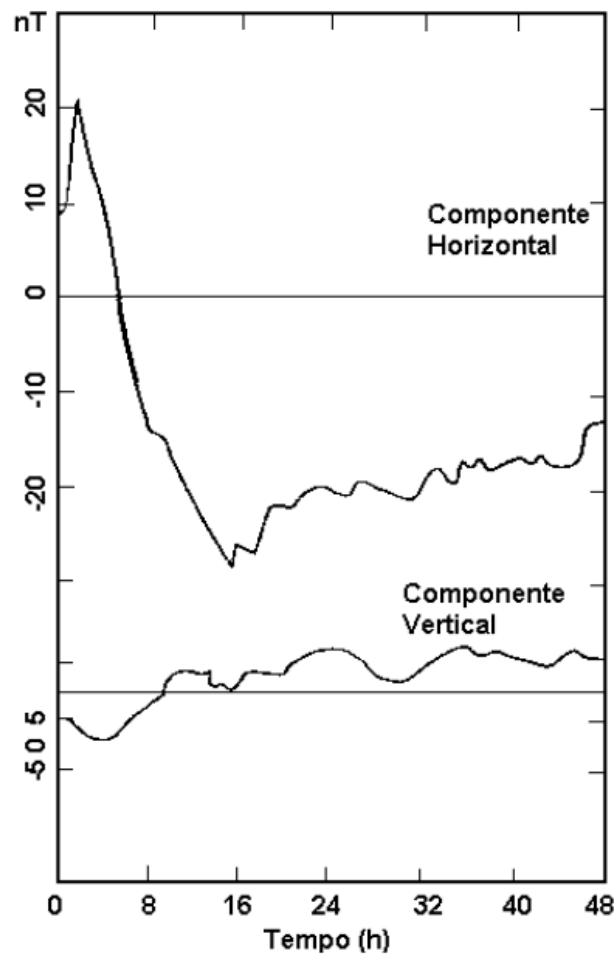


FIGURA 2.13 – Tempestade Magnética típica. Valores médios para a latitude 40N. No hemisfério Sul, a variação da componente vertical seria invertida.
Fonte: (MIRANDA, 2002)

2.4.1 Dínamo da região F

Os ventos neutros induzem movimentos às partículas carregadas da região F da ionosfera. Então, surge a deriva dos íons e elétrons ao longo das linhas de campo geomagnético com intensidade igual à componente do vento na direção do campo. Por outro lado, existe um movimento bem menos intenso na direção perpendicular tanto ao vento como ao campo, cuja velocidade é dada por:

$$\mathbf{V} = \frac{v\omega}{v^2 + \omega^2} \frac{\mathbf{U} \times \mathbf{B}}{B}$$

em que:

$\mathbf{V}$ é a velocidade das partículas carregadas (íons ou elétrons),

$\mathbf{U}$ é a velocidade do vento neutro,

$\mathbf{B}$ é o campo magnético da Terra,

ν é a frequência de colisão entre partículas neutras-carregadas,

$\omega = qB/m$ é a girofrequência das partículas - q e m são carga e massa das partículas, respectivamente.

Devido à dependência com a carga q (positiva ou negativa), é fácil perceber que os íons se movem em um sentido e os elétrons se movem em sentido contrário, criando assim uma corrente elétrica ($\mathbf{J}$). Conseqüentemente, e devido à variação vertical da condutividade, é criado um campo elétrico de polarização vertical que produz deriva zonal.

2.4.2 Dínamo da região E

O campo elétrico da região E equatorial é resultante da superposição de duas contribuições. A primeira componente é gerada pelos ventos de marés, os quais arrastam os íons e deixam os elétrons para trás. Na verdade, os elétrons não são afetados pelos ventos, pois a frequência de colisões com as partículas neutras são muito menores do que a girofrequência. A segunda componente é uma consequência por existir a presença da primeira, ou seja, o campo elétrico gerado pelos ventos de marés, por efeito Hall e devido às variações verticais da condutividade, produz mais um campo elétrico. Os campos elétricos da região E, conforme mencionados acima, possuem suas direções Leste-Oeste, portanto, causam derivas verticais.

É importante destacar que, devido as linhas do campo geomagnético serem conside-

radar linhas equipotenciais, os campos elétricos da região E/F são diretamente mapeados para a região F/E sem atenuações.

2.4.3 Radar de Espalhamento Incoerente

Os radares de espalhamentos incoerentes utilizam o princípio de espalhamento Thomson, o qual consiste na reflexão de parte da energia eletromagnética incidente sobre elétrons livres. Esses instrumentos operam em frequências superiores à frequência crítica da região F, a qual, geralmente, não ultrapassa valores maiores que 20 MHz (BERTONI, 2004).

Os pulsos de ondas eletromagnéticas de frequência bem mais alta que a frequência crítica da região F, ao se deslocarem através do plasma ionosférico, sofrem espalhamento pelos elétrons, produzindo pequenos ecos. Estes ecos possuem fases aleatórias, e daí vem o nome espalhamento incoerente. Utilizando-se um mesmo conjunto de pulsos e sincronizando a recepção dos ecos em diferentes intervalos de tempo, podem-se obter informações da parte inferior ao pico de densidade eletrônica da região F ou de sua parte superior (BERTONI, 2004).

Os sinais recebidos pelos radares de espalhamentos incoerentes criam um espectro de potência. A partir deste espectro, é possível determinar a deriva vertical e zonal, o campo elétrico, o tensor condutividade, as correntes elétricas, a velocidade do vento neutro etc.

3 Filtros de Kalman e Minimax

Este capítulo apresentará os conceitos básicos sobre os filtros de Kalman e o H_∞ . Pretende-se, aqui, descrever os algoritmos destes filtros, sem se alongar demasiadamente em suas fundamentações matemáticas. Para isto, o leitor pode consultar o livro de Simon (SIMON, 2006).

3.1 Filtro de Kalman

O filtro de Kalman foi detalhado, em 1960, por Rudolph Emil Kalman, em seu famoso artigo (KALMAN, 1960), embora as bases de seu algoritmo remontem ao método dos mínimos quadrados de Gauss (SORENSEN, 1970). O algoritmo é um procedimento recursivo, que permite estimar os estados de um sistema linear, geralmente, não diretamente observável, por meio de um conjunto de variáveis observáveis.

Para a utilização do filtro de Kalman, deve existir um modelo linear do processo de interesse. O filtro, então, ajustará os parâmetros do modelo com o objetivo de melhorar o casamento entre o modelo e os dados experimentais que o sustentam.

Um sistema linear de um processo pode ser descrito pelas seguintes equações (SIMON, 2001b):

Equação de estado: $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k$

Equação de saída: $\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{z}_k$

em que:

- $\mathbf{A}$, $\mathbf{B}$ e $\mathbf{C}$ são matrizes.
- k é um índice de tempo.
- $\mathbf{x}$ é o estado do sistema. Este vetor contém toda a informação sobre o estado atual do sistema, mas não pode ser diretamente observável.
- $\mathbf{u}$ é a entrada observável do sistema.
- $\mathbf{y}$ é saída medida. Este vetor é uma função de $\mathbf{x}$, mas está corrompido pelo ruído $\mathbf{z}$.
- $\mathbf{w}$ é o ruído do processo.
- $\mathbf{z}$ é o ruído da medição.

Para apresentar as equações que compõem o filtro de Kalman, considere-se o problema de estimar a altitude e velocidade vertical de um avião. As variáveis de estado são

- $x_1(k)$ é a altitude no tempo t_k .
- $x_2(k)$ é a velocidade no tempo t_k .

A aceleração é dada por

$$a_m = \frac{dx_2(k)}{dt} - g + w \quad (3.1)$$

em que

- g é a aceleração da gravidade e

$$\frac{dx_2(k)}{dt} = \gamma = \frac{dv(t_k)}{dt}$$

é a aceleração do avião.

- w é o erro de medição associado ao processo. Se w é um ruído branco, então, ele possui média zero e desvio-padrão σ_w constante. Para este exemplo, considere-se $\sigma_w = 0.1$.

Os instrumentos a bordo do avião (altímetro e acelerômetro) produzirão medidas a cada intervalo de 0.5 s, tal que $t_0 = 0, t_1 = 0.5, t_2 = 1.0, t_3 = 1.5, \dots$. Se a aceleração for constante nesses pequenos intervalos, a velocidade e a posição serão dadas pelas fórmulas:

$$v(t_k + \Delta t) = v(t_k) + \gamma \Delta t$$

$$s(t_k + \Delta t) = s(t_k) + v(t_k) \Delta t + \frac{1}{2} \gamma \Delta t^2$$

Dado que $\Delta t = 0.5$, $s(t_k) = x_1(k)$, $v(t_k) = x_2(k)$, $s(t_k + \Delta t) = x_1(k+1)$ e $v(t_k + \Delta t) = x_2(k+1)$, pode-se usar as seguintes fórmulas de predição para as duas variáveis de estado:

$$\begin{aligned} x_1(k+1) &= x_1(k) + \frac{1}{2} x_2(k) + \frac{1}{8} (a_m + g - w) \\ x_2(k+1) &= x_2(k) + \frac{1}{2} (a_m + g - w) \end{aligned} \quad (3.2)$$

Em forma matricial, estas equações tornam-se:

$$\mathbf{x} = \Phi \mathbf{x} + \mathbf{h} + \mathbf{u} \quad (3.3)$$

$$\Phi = \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & 1 \end{bmatrix} \quad (3.4)$$

$$\mathbf{h} = \begin{bmatrix} \frac{1}{8}(g + a_m) \\ \frac{1}{2}(g + a_m) \end{bmatrix} \quad (3.5)$$

$$\mathbf{u} = \begin{bmatrix} -\frac{1}{8}w \\ -\frac{1}{2}w \end{bmatrix} \quad (3.6)$$

O filtro de Kalman é um algoritmo usado para estimar as variáveis de estado $\mathbf{x}(k)$ por meio das medições $\mathbf{y}(k)$. Essas estimativas serão chamadas $\hat{\mathbf{x}}(k)$. O algoritmo inicia com valores iniciais para as variáveis de estado em $\mathbf{x}$. Este valor inicial é denominado de estimativa a priori e é denotado por $\hat{\mathbf{x}}(0)$. Pode-se fornecer a matriz de covariância inicial do erro $\mathbf{P}(0)$, que expressa a confiança que se possui nos valores iniciais das variáveis de estado. A parte central do filtro de Kalman é o cálculo da matriz de covariância do erro na estimação de $\mathbf{x}$. A matriz de covariância é dada por:

$$\mathbf{P} = E\{(\mathbf{x}(t_0) - \hat{\mathbf{x}}(t_0))(\mathbf{x}(t_0) - \hat{\mathbf{x}}(t_0))^T\}.$$

Cada elemento da matriz é dado por:

$$\mathbf{P}_{ij}(k) = E\{x_i(k)x_j(k)\}$$

Os elementos da diagonal principal são as variâncias de x_i . Os outros elementos são as

covariâncias e estão relacionados às correlações entre x_i e x_j . De fato, as correlações são as covariâncias divididas pelos produtos dos desvios-padrões:

$$\rho = \frac{\mathbf{P}_{ij}}{\sigma_i \sigma_j}$$

O altímetro produz uma medida de y a cada 0.5 s:

$$y = x_1 + z$$

em que z é o erro na medição. Supondo que z é um ruído branco com desvio-padrão de 100 pés, a variância de z será 10^4 , isto é, 10,000. Em forma matricial,

$$\mathbf{y} = \mathbf{M}\mathbf{x} + \mathbf{z}$$

em que

$$\mathbf{M} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Tanto $\mathbf{y}$ quanto $\mathbf{z}$ possuem apenas um elemento neste exemplo. Tendo iniciado com uma estimativa a priori $\hat{\mathbf{x}}(0)$, o próximo passo é estimar $\mathbf{x}(1)$. Kalman mostra que se pode extrapolar a partir de $\hat{\mathbf{x}}(0)$, usando a seguinte equação:

$$\bar{\mathbf{x}}(1) = \Phi \hat{\mathbf{x}}(0) + \mathbf{h}(0)$$

Em que a barra é usada para indicar que isto é uma extrapolação a partir de $\hat{\mathbf{x}}(0)$ e que não se incluiu qualquer nova medição. Retirar-se-á os argumentos das variáveis com o intuito de simplificar a notação.

Equação de Kalman 1

$$\bar{\mathbf{x}} = \Phi \hat{\mathbf{x}} + \mathbf{h}$$

Para calcular a matriz de covariância do erro do novo $\bar{\mathbf{x}}$, tem-se:

$$\bar{\mathbf{P}}(1) = \Phi \mathbf{P}(0) \Phi^T + \mathbf{Q}$$

em que $\mathbf{Q}$ é a matriz de covariância de $\mathbf{u}(k)$. Novamente, retiram-se os argumentos para obter:

Equação de Kalman 2

$$\bar{\mathbf{P}} = \Phi \mathbf{P} \Phi^T + \mathbf{Q}$$

Para este exemplo, a matriz de covariância $\mathbf{Q}$ é dada por

$$\begin{aligned} \mathbf{Q} &= E\{\mathbf{u}\mathbf{u}^T\} \\ &= E\left\{ \begin{bmatrix} -\frac{1}{8}w \\ -\frac{1}{2}w \end{bmatrix} \begin{bmatrix} -\frac{1}{8}w & -\frac{1}{2}w \end{bmatrix} \right\} \\ &= E\left\{ \begin{bmatrix} (\frac{1}{8}w)^2 & \frac{1}{2}\frac{1}{8}w^2 \\ \frac{1}{2}\frac{1}{8}w^2 & (\frac{1}{2}w)^2 \end{bmatrix} \right\} \\ &= \begin{bmatrix} E\{(\frac{1}{8}w)^2\} & E\{\frac{1}{2}\frac{1}{8}w^2\} \\ E\{\frac{1}{2}\frac{1}{8}w^2\} & E\{(\frac{1}{2}w)^2\} \end{bmatrix} \\ &= E\{w^2\} \begin{bmatrix} (\frac{1}{8})^2 & \frac{1}{2} \times \frac{1}{8} \\ \frac{1}{2} \times \frac{1}{8} & (\frac{1}{2})^2 \end{bmatrix} \end{aligned}$$

Desde que $E\{w^2\} = 0.01$, a matriz final será:

$$\mathbf{Q} = \begin{bmatrix} 1.5625 \times 10^{-4} & 6.25 \times 10^{-4} \\ 6.25 \times 10^{-4} & 2.5 \times 10^{-3} \end{bmatrix}$$

A matriz de covariância $\mathbf{R}$ do ruído branco z , um vetor unário, é:

$$\mathbf{R} = [10^4]$$

Quando o piloto ativa o sistema durante o vôo, ele obtém a altitude inicial de 10000 pés e uma velocidade inicial nula. Logo,

$$\hat{\mathbf{x}} = \begin{bmatrix} 10000 \\ 0 \end{bmatrix}$$

O piloto também lê que o erro na estimativa da altitude possui um desvio-padrão de 1000 pés e que o erro na estimativa da velocidade possui um desvio-padrão de 5 pés/s. Logo,

$$\mathbf{P} = \begin{bmatrix} 10^6 & 0 \\ 0 & 25 \end{bmatrix}$$

Agora, está-se pronto para usar o filtro de Kalman. Neste ponto, o piloto lê a primeira medida do acelerômetro:

$$a_m = 33$$

Se se adotar

$$g = -32$$

a equação de Kalman 1 produz

$$\bar{\mathbf{x}} = \begin{bmatrix} 10000.125 \\ 0.5 \end{bmatrix}$$

A equação de Kalman 2 produz

$$\bar{\mathbf{P}} = \begin{bmatrix} 1,000,006.25 & 12.500625 \\ 12.500625 & 25.0025 \end{bmatrix}$$

Agora, tome-se a primeira medida $\mathbf{y}(1)$ e, de acordo com Kalman, a nova estimativa ótima é dada pela

Equação de Kalman 3

$$\hat{\mathbf{x}} = \bar{\mathbf{x}} + \mathbf{K}(\mathbf{y} - \mathbf{M}\bar{\mathbf{x}})$$

em que $\mathbf{K}$ é o ganho do filtro de Kalman. O ganho ótimo pode ser calculado por meio da seguinte equação

Equação de Kalman 4

$$\mathbf{K} = \bar{\mathbf{P}}\mathbf{M}^T(\mathbf{M}\bar{\mathbf{P}}\mathbf{M}^T + \mathbf{R})^{-1}$$

Para este exemplo, a equação de Kalman 4 produz

$$\mathbf{K} = \begin{bmatrix} 0.990099 \\ 1.2376 \times 10^{-5} \end{bmatrix}$$

Supondo-se que a primeira medida do altímetro é

$$\mathbf{y}(1) = 10000 \quad \text{pés}$$

Neste caso, a equação de Kalman 3 produz:

$$\hat{\mathbf{x}} = \begin{bmatrix} 10990 \\ 0.512 \end{bmatrix}$$

O novo $\hat{\mathbf{x}}$ tem uma matriz de covariância de erro, que pode ser calculada usando a equação de Kalman 5:

Equação de Kalman 5

$$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{M})\bar{\mathbf{P}}(\mathbf{I} - \mathbf{K}\mathbf{M})^T + \mathbf{K}\mathbf{R}\mathbf{K}^T$$

em que $\mathbf{I}$ é a matriz identidade com as mesmas dimensões de $\mathbf{P}$. Usando a equação 5, a nova matriz de covariância será:

$$\mathbf{P} = \begin{bmatrix} 9901 & 0.12377 \\ 0.12377 & 25.0023 \end{bmatrix}$$

O filtro de Kalman consiste no repetido uso das equações 1 a 5 para cada nova medição.

Assim, retornando às equações 1 e 2, o resultado da equação 2 é

$$\bar{\mathbf{P}} = \begin{bmatrix} 9907 & 12.625 \\ 12.625 & 25.0048 \end{bmatrix}$$

Então, usando as equações 4, 3 e 5 novamente, a equação 5 produz:

$$\mathbf{P} = \begin{bmatrix} 4977 & 6.342 \\ 6.342 & 24.996 \end{bmatrix}$$

Colocando todas as equações que formam o filtro de Kalman juntas, tem-se:

$$\bar{\mathbf{x}} = \Phi \hat{\mathbf{x}} + \mathbf{h} \quad (3.7)$$

$$\bar{\mathbf{P}} = \Phi \mathbf{P} \Phi^T + \mathbf{Q} \quad (3.8)$$

$$\hat{\mathbf{x}} = \bar{\mathbf{x}} + \mathbf{K}(\mathbf{y} - \mathbf{M}\bar{\mathbf{x}}) \quad (3.9)$$

$$\mathbf{K} = \bar{\mathbf{P}}\mathbf{M}^T(\mathbf{M}\bar{\mathbf{P}}\mathbf{M}^T + \mathbf{R})^{-1} \quad (3.10)$$

$$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{M})\bar{\mathbf{P}}(\mathbf{I} - \mathbf{K}\mathbf{M})^T + \mathbf{K}\mathbf{R}\mathbf{K}^T \quad (3.11)$$

3.1.1 Propriedades

O filtro de Kalman é um estimador que satisfaz aos seguintes requisitos (SIMON, 2001b):

- o valor médio da estimativa do estado é igual ao valor médio do estado verdadeiro.

Dito de outra forma, o valor esperado da estimativa é igual ao valor esperado do estado.

- A estimativa do estado varia minimamente em relação ao estado verdadeiro. Em outros termos, o estimador apresenta a menor variância de erro possível.

A solução do filtro de Kalman somente pode ser aplicada se as condições abaixo sobre o ruído que afeta o sistema são satisfeitas (SIMON, 2001b):

1. Tanto o ruído do processo $\mathbf{w}$ quanto o ruído de medição $\mathbf{z}$ possuem valores médios iguais a zero;
2. Não existe correlação entre $\mathbf{w}$ e $\mathbf{z}$. Ou seja, em qualquer tempo k , $\mathbf{w}_k$ e $\mathbf{z}_k$ são variáveis aleatórias independentes

Dadas as restrições acima, as matrizes de covariância do ruído do processo, $\mathbf{S}_w$, e de ruído de medição $\mathbf{S}_z$ são definidas por:

$$\mathbf{S}_w = E\{\mathbf{w}_k \mathbf{w}_k^T\}$$

$$\mathbf{S}_z = E\{\mathbf{z}_k \mathbf{z}_k^T\}$$

3.2 Filtro Minimax

Como visto, o filtro de Kalman funciona bem sob certas condições:

- os ruídos envolvidos no processo precisam ter médias zero. Esta propriedade de média zero deve manter-se durante o tempo total de duração do processo e também a cada instante de tempo. Ou seja, o valor esperado de $\mathbf{w}$ e $\mathbf{z}$, em cada instante, deve ser igual a zero.
- É necessário conhecer o desvio-padrão dos ruídos envolvidos no processo. O filtro de Kalman usa as matrizes $\mathbf{S}_w$ e $\mathbf{S}_z$ como parâmetros de projeto. Assim, se essas matrizes não forem conhecidas, não se consegue projetar um filtro de Kalman apropriado.

Se as condições anteriores não forem atendidas, uma opção é usar o filtro de Kalman mesmo assim, e esperar que ele funcione razoavelmente bem. Uma outra opção é usar um filtro H_∞ , também conhecido por *filtro minimax*. O filtro H_∞ não impõe qualquer condição sobre o ruído e minimiza o erro de estimativa no pior caso.

3.2.1 Filtragem H_∞

Suponha-se que se, para um dado sistema dinâmico linear, deseje encontrar um estimador para os seus estados, satisfazendo a algum critério J de qualidade. O problema que o filtro H_∞ tenta resolver é:

$$\min_{\hat{\mathbf{x}}} \max_{\mathbf{w}, \mathbf{z}} J$$

Pode-se pensar nos ruídos $\mathbf{w}$ e $\mathbf{z}$ como adversários que tentam piorar nossa estimativa. Assim, dados os piores valores para esses ruídos, deseja-se encontrar uma estimativa de

estado que minimize o pior efeito possível desses ruídos no erro de estimação. Devido a este fato, esse filtro é também chamado de filtro minimax: ele tenta minimizar o erro de estimação máximo (SIMON, 2001a). A função J pode ser definida como¹:

$$J = \frac{\text{média } \|\mathbf{x}_k - \hat{\mathbf{x}}_k\|_{\mathbf{Q}}}{\text{média } \|\mathbf{w}_k\|_{\mathbf{W}} + \text{média } \|\mathbf{z}_k\|_{\mathbf{Z}}} \quad (3.12)$$

As médias acima são tomadas sobre todos os intervalos k . As matrizes $\mathbf{Q}$, $\mathbf{W}$ e $\mathbf{Z}$, usadas nas normas ponderadas em J , são escolhidas pelo projetista, visando obter a melhor relação custo-benefício. Por exemplo, se se souber que o ruído $\mathbf{w}$ será menor que o ruído $\mathbf{z}$, então, os elementos da matriz $\mathbf{W}$ serão menores que aqueles da matriz $\mathbf{Z}$. Isto enfatizará a importância do ruído $\mathbf{z}$ em relação ao ruído $\mathbf{w}$. Se o que interessa é a precisão da estimativa em elementos específicos do vetor de estado $\mathbf{x}$, ou se os elementos do vetor de estados estiverem escalonados tal que difiram por uma ordem de magnitude ou mais, então, definir-se-á uma matriz $\mathbf{Q}$ que leve em conta tais fatos (SIMON, 2001a).

O problema do filtro minimax é difícil de resolver analiticamente. Entretanto, pode-se resolver um problema relacionado:

$$J < \frac{1}{\gamma}$$

em que γ é uma constante numérica determinada pelo projetista.

O problema original se reduz a encontrar uma estimativa de estado tal que o valor máximo de J seja sempre menor que a quantidade $\frac{1}{\gamma}$, independentemente dos valores dos

¹A norma vetorial ponderada de um vetor $\mathbf{x}$ é definida como:

$$\|\mathbf{x}\|_{\mathbf{Q}}^2 = \mathbf{x}^T \mathbf{Q} \mathbf{x}$$

onde $\mathbf{Q}$ é qualquer matriz com dimensões compatíveis.

ruídos $\mathbf{w}$ e $\mathbf{z}$. A estimativa de estado que força $J < \frac{1}{\gamma}$ é dada por:

$$\mathbf{L}_k = (\mathbf{I} - \gamma \mathbf{Q} \mathbf{P}_k + \mathbf{C}^T \mathbf{Z}^{-1} \mathbf{C} \mathbf{P}_k)^{-1} \quad (3.13)$$

$$\mathbf{K}_k = \mathbf{A} \mathbf{P}_k \mathbf{L}_k \mathbf{C}^T \mathbf{Z}^{-1} \quad (3.14)$$

$$\hat{\mathbf{x}}_{k+1} = \mathbf{A} \hat{\mathbf{x}}_k + \mathbf{B} \mathbf{u}_k + \mathbf{K}_k (\mathbf{y}_k - \mathbf{C} \hat{\mathbf{x}}_k) \quad (3.15)$$

$$\mathbf{P}_{k+1} = \mathbf{A} \mathbf{P}_k \mathbf{L}_k \mathbf{A}^T + \mathbf{W} \quad (3.16)$$

Estas são as equações do filtro H_∞ , em que $\mathbf{I}$ é a matriz identidade e $\mathbf{K}_k$ é a matriz de ganho. A estimativa inicial $\hat{\mathbf{x}}_0$ deve ser inicializada com o melhor palpite sobre $\mathbf{x}_0$, e o valor inicial de $\mathbf{P}_0$ deve ser ajustado para se obter um desempenho aceitável do filtro. Em geral, os valores dos elementos de $\mathbf{P}_0$ deveriam ser pequenos, se se estiver altamente confiante na estimativa inicial do estado $\hat{\mathbf{x}}_0$.

A derivação matemática das equações do H_∞ é válida somente se γ for escolhida tal que todos os autovalores da matriz $\mathbf{P}$ tenham magnitudes menores que um. Se γ possuir um valor elevado, não existirá uma solução para o problema de filtragem H_∞ (SIMON, 2001a).

Como pode ser observado nas equações do filtro H_∞ , a inversão de matrizes é necessária em quase todos os passos do cálculo dos termos $\mathbf{L}_k$. Pode-se melhorar o comportamento em tempo real do filtro minimax, observando que os cálculos de $\mathbf{L}_k$, $\mathbf{P}_k$ e $\mathbf{K}_k$ podem ser realizados a priori. Isto é, não se precisa das medições para computar a matriz de ganho $\mathbf{K}_k$. É possível calcular cada $\mathbf{K}_k$ previamente e armazenar as matrizes resultantes.

3.3 Considerações finais

O filtro de Kalman é mais utilizado e conhecido que o filtro minimax e seu desempenho, geralmente, é melhor (SIMON, 2001a).

O filtro H_∞ exige maior quantidade de ajustes para funcionar bem. Seus parâmetros de ajuste incluem γ e as matrizes $\mathbf{P}_0$, $\mathbf{Z}$, $\mathbf{W}$ e $\mathbf{Q}$. Devido a esse grande número de parâmetros, é mais difícil ajustar um filtro H_∞ do que um filtro de Kalman.

4 Ondaletas

O objetivo deste capítulo é fazer uma introdução geral à teoria das ondaletas¹. Existem muitas formas de visualizar ondaletas. Aqui, elas serão consideradas como transformadas, que podem ser obtidas pela aplicação repetida de filtros. Isto abrirá oportunidade para a combinação de ondaletas com o filtro de Kalman.

Existem vários trabalhos que relacionam as ondaletas aos problemas geofísicos (PROTÁZIO, 2001; BOLZAN, 2004; DOMINGUES; MENDES; COSTA, 2005). O propósito, aqui, foi verificar a possibilidade de uso delas em climatologia espacial.

4.1 A transformada de Fourier

Em 1822, o matemático e físico francês Jean-Baptiste Joseph Fourier mostrou que qualquer função periódica pode ser expressa por uma soma infinita de funções exponenciais complexas periódicas. Posteriormente, a idéia de Fourier fora generalizada, primeiro, para funções não periódicas e, depois, para sinais discretos periódicos ou não.

¹Segue-se aqui a tradução de Pedro Morettin (MORETTIN, 1999) para os vocábulos *wavelet* (inglês) e *ondelette* (francês).

A *transformada de Fourier* (TF) é definida pelas seguintes equações

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-2j\pi ft} dt \quad (4.1)$$

$$x(t) = \int_{-\infty}^{\infty} X(f)e^{2j\pi ft} df \quad (4.2)$$

A equação (4.1) é chamada de transformada direta de Fourier da função $x(t)$ e a equação (4.2) é chamada de transformada inversa de Fourier de $X(f)$.

A TF decompõe um sinal em termos de funções exponenciais complexas de diferentes frequências. Isto pode ser visto na equação (4.1), em que o sinal $x(t)$ é multiplicado por um termo exponencial com uma determinada frequência f e, depois, é integrado sobre *todo o eixo do tempo*. Logo, a informação fornecida por essa integral corresponde a todos os instantes de tempo, de menos infinito a mais infinito. Desta forma, não importa onde, no tempo, a componente com frequência f apareça, pois ela afeta igualmente o resultado da integração. Do exposto, vê-se que a transformada de Fourier não é adequada para sinais cujo conteúdo frequencial varie com o tempo, ou seja, para sinais não-estacionários.

A transformada de Fourier informa se um determinado componente de frequência existe ou não. Esta informação é independente do instante no tempo em que esta componente aparece. Assim, é muito importante saber se um sinal é estacionário ou não, antes de processá-lo com a TF.

4.2 Transformada de Fourier de curta duração

Uma forma intuitiva de analisar um sinal não-estacionário é realizar uma análise espectral dependente do tempo. Se for possível dividir um sinal não-estacionário em uma sequência de segmentos temporais, nos quais o sinal seja, aproximadamente, estacionário, então, pode-se aplicar a TF a cada um desses segmentos locais do sinal. Esta abordagem leva a uma versão revisada da transformada de Fourier, chamada de *transformada de Fourier de curta duração* (TFCD), também conhecida como transformada janelada de Fourier.

A transformada de Fourier de curta duração é associada a uma janela de largura fixa, e foi o matemático e físico húngaro Dennis Gábor, em 1946, o primeiro a introduzi-la. Esta transformada é definida como

$$TFCD_x^w(\tau, f) = \int_{-\infty}^{+\infty} [x(t)w^*(t - \tau)]e^{-2j\pi ft} dt \quad (4.3)$$

A função de janelamento w (na equação w^* é o seu complexo conjugado) é escolhida tal que a largura de sua janela seja igual ao tamanho do segmento do sinal em que a condição de estacionaridade é válida.

Como visto na equação (4.3), a função de janelamento é deslocada ao longo do eixo temporal e centrada em τ . Esta janela deslizando permite a localização temporal do sinal $x(t)$.

A TFCD possui um problema relacionado com o Princípio da Incerteza de Heisenberg, que, quando aplicado à representação tempo-frequência do sinal, enuncia que não se pode saber com exatidão quais componentes espectrais existem em um determinado instante no

tempo. O máximo que se pode saber são os *intervalos* de tempo nos quais certas *bandas* de frequências existem.

A largura Δt da janela $w(t)$, no domínio do tempo, e a largura de banda $\Delta\omega$ da janela $W(\omega)$, a transformada de Fourier de $w(t)$, no domínio da frequência, são definidas como

$$\Delta t^2 = \frac{\int_{-\infty}^{+\infty} t^2 |w(t)|^2 dt}{\int_{-\infty}^{+\infty} |w(t)|^2 dt} \quad \Delta\omega^2 = \frac{\int_{-\infty}^{+\infty} \omega^2 |W(\omega)|^2 d\omega}{\int_{-\infty}^{+\infty} |W(\omega)|^2 d\omega} \quad (4.4)$$

Dois sinais senoidais podem ser discriminados somente se eles estão separados por pelo menos $\Delta\omega$. Assim, $\Delta\omega$ é a resolução no domínio da frequência da transformada de Fourier de curta duração. Similarmente, dois pulsos no domínio do tempo podem ser discriminados somente se estiverem separados por pelo menos Δt . Assim que uma janela tenha sido escolhida para a TFCD, as resoluções no tempo e na frequência, dadas por (4.4), estarão fixas em todo o plano tempo-frequência (POULARIKAS, 2000).

O Princípio da Incerteza de Heisenberg enuncia que o produto das resoluções de tempo e frequência é limitado por

$$\Delta t \Delta\omega \geq \frac{1}{2} \quad (4.5)$$

Um sinal não pode ser representado por um ponto no espaço tempo-frequência. Pode-se somente determinar sua posição neste espaço dentro de um retângulo de área $\Delta t \Delta\omega$. Assim, pode-se somente trocar a resolução no tempo pela resolução na frequência ou vice versa.

A largura da janela na TFCD é, então, limitada por esse princípio. Ao se usar uma janela com comprimento infinito, obtém-se a transformada de Fourier, que dá uma perfeita resolução em frequência, mas nenhuma informação sobre tempo. Se forem utilizadas

janelas cada vez mais estreitas, para obedecer à condição de estacionaridade, melhora-se a resolução no tempo, mas piora-se a resolução em frequência.

Gabor propôs a função Gaussiana como a função de janelamento, pois ela possui o menor produto entre largura de banda e o tempo, como determinado pelo Princípio da Incerteza (4.5). A transformada de Fourier da janela Gaussiana é ainda uma função Gaussiana, como visto abaixo

$$w(t) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{t^2}{2\sigma^2}\right) \quad \text{e} \quad W(\omega) = \exp\left(-\frac{\sigma^2\omega^2}{2}\right)$$

Nestas equações, σ é um parâmetro que controla a largura da janela. Um cálculo simples mostra que

$$\Delta t^2 = \frac{\sigma^2}{2} \quad \text{e} \quad \Delta\omega^2 = \frac{1}{2\sigma^2}$$

que satisfaz ao Princípio da Incerteza e alcança o menor produto entre a largura de banda e o tempo, pois $\Delta t \Delta\omega = 1/2$.

4.3 Análise de multirresolução

Os problemas de resolução no tempo e na frequência são resultantes de fenômenos físicos e existem independentemente da transformada usada. Entretanto, é possível analisar qualquer sinal empregando uma abordagem alternativa, chamada de análise de multirresolução (AMR). A AMR analisa o sinal em diferentes frequências, utilizando diferentes resoluções. A resolução para cada componente espectral não será igual, como no caso da TFCD.

A análise de multirresolução é projetada para dar uma boa resolução no tempo e uma

pobre resolução na frequência, quando se lida com os componentes de alta frequência do sinal e uma boa resolução na frequência e uma pobre resolução no tempo, quando se lida com componentes de baixa frequência. Essa abordagem faz sentido, especialmente, nos casos em que o sinal possui componentes de alta frequência de curta duração e componentes de baixa frequência de longa duração (POLIKAR, 2001).

Formalmente, uma AMR é uma sequência crescente de subespaços fechados $\{V_j, j \in \mathbb{Z}\}$, que aproximam $L^2(\mathbb{R})$, ou seja, ela apresenta as seguintes propriedades (MORETTIN, 1999):

1. $\dots \subset V_{-1} \subset V_0 \subset V_1 \subset \dots$
2. $L^2(\mathbb{R}) = \bigcup_j V_j$
3. $\bigcap_j V_j = \{0\}$
4. $f(t) \in V_j \Leftrightarrow f(2t) \in V_{j+1}, \quad \forall j$
5. Há uma base ortonormal $\{\phi_{j,k} \mid j, k \in \mathbb{Z}\}$ para cada subespaço V_j , ou seja, as projeções ortogonais de $f(t)$ sobre cada um dos subespaços V_j podem ser descritas como combinações lineares de $\phi_{j,k}$.

4.4 Transformada de ondaletas contínua

Ondaletas são funções obtidas a partir de uma função protótipo $\psi(t) \in L^2(\mathbb{R})^2$, conhecida como *ondaleta-mãe*, por meio de dilatações e translações. A idéia aqui é considerar

² $L^2(\mathbb{R})$ é o espaço de todas as funções mensuráveis de quadrado integrável sobre $\mathbb{R}$ para as quais

$$\|f\|^2 = \int_{-\infty}^{+\infty} |f(t)|^2 dt < \infty$$

dilatações (ou compressões) e translações de uma única função ψ de modo a cobrir $\mathbb{R}$. A forma geral de uma família de ondaletas é dada por (MORETTIN, 1999)

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}} \psi\left(\frac{t-b}{a}\right) \quad (4.6)$$

em que a é um fator de escalamento responsável pelas dilatações, e a variável b introduz as translações.

Alguns exemplos de ondaletas-mãe são (STARK, 2005):

- Ondaleta de Haar

$$\psi(t) = \begin{cases} 1 & \text{se } 0 \leq t < \frac{1}{2} \\ -1 & \text{se } \frac{1}{2} \leq t < 1 \\ 0 & \text{em qualquer outro caso} \end{cases}$$

- Chapéu mexicano

$$\psi(t) = \frac{2}{\sqrt{3}} \pi^{-\frac{1}{4}} (1 - t^2) e^{-\frac{t^2}{2}}$$

- Ondaleta de Morlet

$$\psi(t) = \pi^{-\frac{1}{4}} (e^{-j\alpha t} - e^{-\frac{\alpha^2}{2}}) e^{-\frac{t^2}{2}} \quad \text{com} \quad \alpha = \pi \sqrt{\frac{2}{\ln 2}}$$

Para que uma ondaleta-mãe $\psi(t) \in L^2(\mathbb{R})$ origine uma família de ondaletas, exige-se que ela (FARIA, 1997):

1. seja absolutamente integrável, isto é,

$$\int_{-\infty}^{+\infty} \psi(t) dt < \infty$$

2. possua energia finita

$$\int_{-\infty}^{+\infty} |\psi(t)|^2 dt < \infty$$

3. satisfaça a uma condição de admissibilidade

$$c_\psi = 2\pi \int_{-\infty}^{+\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega < \infty, \text{ onde } \hat{\psi}(\omega) \text{ é a transformada de Fourier de } \psi(t)$$

Uma vez que tais exigências sejam cumpridas, pode-se criar uma família de ondaletas duplamente indexadas pelo escalamento e pelas translações da ondaleta-mãe:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}} \psi\left(\frac{t-b}{a}\right) \quad \text{com } a, b \in \mathbb{R} \text{ e } a \neq 0 \quad (4.7)$$

Para $f \in L^2(\mathbb{R})$, a *transformada de ondaletas contínua*, com respeito a ψ , é definida por (MORETTIN, 1999):

$$(W_\psi f)(a, b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{+\infty} f(t) \psi^*\left(\frac{t-b}{a}\right) dt \quad (4.8)$$

em que ψ^* é o complexo conjugado de ψ . A função $f(t)$ pode ser recuperada de $(W_\psi f)(a, b)$ por meio da *transformada de ondaletas inversa*, dada por (FARIA, 1997):

$$f(t) = \frac{1}{c_\psi} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \frac{(W_\psi f)(a, b) \psi_{a,b}(t)}{a^2} da db \quad (4.9)$$

onde a constante c_ψ é a mesma da condição de admissibilidade.

Como visto na seção anterior, a transformada de Fourier de curta duração divide o plano tempo-frequência em segmentos retangulares de dimensões constantes, analisando tanto as altas frequências quanto as baixas com a mesma resolução. A transformada de ondaletas permite analisar o plano tempo-frequência com uma resolução variável: uma resolução mais fina no tempo para altas frequências e mais seletiva em frequência para baixas frequências. Este efeito pode ser explicado da seguinte forma: em frequências mais altas a transformada utiliza ondaletas mais estreitas, comprimidas e de curta duração, permitindo ver o sinal analisado em uma escala mais detalhada, localizada e com melhor resolução temporal. Em frequências mais baixas, ondaletas mais largas e dilatadas enquadram o sinal numa escala maior, com menos detalhes, focando nas características globais. Nesta faixa, a resolução em frequência é mais fina do que a temporal, facilitando a localização em frequência (FARIA, 1997).

A implementação da transformada de ondaletas contínua no computador envolve a simulação de processos analógicos, o que somente pode ser feito por meio de amostragem dessa transformada. Isto envolve, muitas vezes, uma grande carga computacional, que, além de introduzir erros e de dificultar uma implementação em tempo real, é altamente redundante no tocante à reconstrução do sinal. Uma solução para esses problemas é usar a transformada de ondaletas discreta, que fornece informação suficiente tanto para a análise quanto para a síntese do sinal original e reduz, significativamente, o tempo de computação (POLIKAR, 2001).

4.5 Transformada de ondaletas discreta

Para encontrar uma forma discreta da transformada de ondaletas, começa-se pela discretização dos parâmetros a (escala) e b (deslocamento) da equação (4.8). Isto pode ser feito tomando-se valores inteiros para a escala, que sejam potências de uma constante a_0 , ou seja,

$$a = a_0^{-j} \text{ com } a_0 > 1 \text{ e } j \in \mathbb{Z}$$

A discretização de b dependerá de j para que ondaletas estreitas, adequadas para os componentes de alta frequência, sejam deslocadas por passos pequenos, e ondaletas mais largas, adequadas para os componentes de baixa frequência, sejam deslocadas por passos maiores (FARIA, 1997). Uma maneira de alcançar essa discretização é fazer

$$b = ka_0^{-j} \quad \text{com } k \in \mathbb{Z}$$

Uma escolha bastante comum é fazer $a_0 = 2$. Com isto, a equação da ondaleta (4.7) se transforma em

$$\psi_{j,k}(t) = 2^{\frac{j}{2}} \psi(2^j t - k) \quad \text{com } j, k \in \mathbb{Z} \quad (4.10)$$

A transformada de ondaletas discreta e a sua inversa são definidas por

$$d_{j,k}^\psi = \sum_{t=-\infty}^{+\infty} f(t) \psi_{j,k}(t) \quad (4.11)$$

$$f(t) = \sum_{j=-\infty}^{+\infty} \sum_{k=-\infty}^{+\infty} d_{j,k}^\psi \psi_{j,k}(t) \quad (4.12)$$

Os $d_{j,k}^\psi$ vistos nessas equações são chamados de *coeficientes das ondaletas*.

4.5.1 Exemplo de cálculo da transformada de ondaletas discreta de Haar

A ondaleta de Haar é a mais simples das ondaletas e, por isto, ela é usada como exemplo para ilustrar a forma de cálculo da transformada de ondaletas discreta.

Supondo-se que o sinal de interesse seja uma função rampa:

$$f(x) = x$$

Aplica-se esta função a pontos discretos para obter a seqüência $f(1), f(2), f(3), \dots, f(32)$.

Ela deve possuir $n = 2^m$ valores. Neste caso, $m = 5$ e a seqüência é igual a

$$1, 2, 3, \dots, 32$$

A seqüência é indexada e os índices representam a variável temporal (ou espacial). A transformada de Haar, como outras transformadas, possui dois filtros:

Filtro passa-baixa: A média de dois valores adjacentes na seqüência.

$$l_i = \begin{cases} \frac{x_i + x_{i+1}}{2} & \text{para } i = 1, \dots, n-1 \\ \frac{x_n}{2} & \text{para } i = n \end{cases}$$

A média funciona como um filtro passa-baixa, pois elimina as altas frequências.

Filtro passa-alta: A diferença entre os mesmos dois valores usados no filtro passa-baixa.

$$h_i = \begin{cases} x_{i+1} - x_i & \text{para } i = 1, \dots, n-1 \\ x_n & \text{para } i = n \end{cases}$$

4.5.2 Ondaletas e os bancos de filtros

O cálculo da matriz $\mathbf{H}$, para o banco de filtro de síntese de Haar, assume uma entrada x consistindo de n amostras. O filtro passa-baixa de Haar H_0 simplesmente aplica a média aos valores adjacentes de sua entrada. Em forma matricial, tem-se:

$$\mathbf{H}_0^n = \frac{1}{2} \begin{bmatrix} 1 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}_{n \times n}$$

O filtro passa-alta de Haar H_1 calcula a metade das diferenças entre amostras sucessivas da entrada:

$$\mathbf{H}_1^n = \frac{1}{2} \begin{bmatrix} 1 & -1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & -1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & -1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}_{n \times n}$$

A matriz de decimação (ou *downsampling*) tomará a primeira, terceira, quinta, ..., entradas do sinal:

$$\mathbf{D}^n = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 \end{bmatrix}_{\frac{n}{2} \times n}$$

A ação combinada da filtragem seguida pela decimação é dada por:

$$\mathbf{L}^n = \mathbf{D}^n \times \mathbf{H}_0^n$$

para o filtro passa-baixa, e por

$$\mathbf{B}^n = \mathbf{D}^n \times \mathbf{H}_1^n$$

para o filtro passa-alta.

Considere-se um exemplo concreto no qual alguém deseje encontrar as ondaletas no nível 2, para $n = 4$. Neste caso, os filtros são:

$$\mathbf{H}_0^4 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

$$\mathbf{H}_1^4 = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

A matriz de decimação é:

$$\mathbf{D}^4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

O filtro passa-alta é dado por $\mathbf{B}^4 = \mathbf{D}^4 \times \mathbf{H}_1^4$, ou seja:

$$\mathbf{B}^4 = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} \end{pmatrix}$$

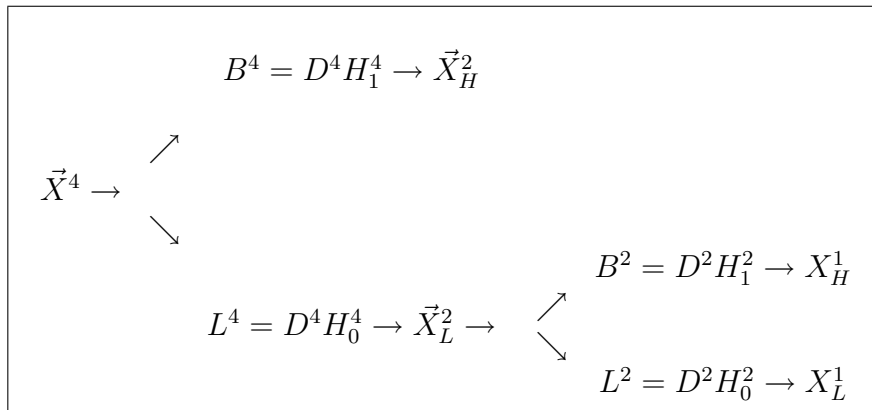


FIGURA 4.1 – A implementação por banco de filtros da transformada discreta de ondaleta.

Até o momento, foi feita a implementação do caminho

$$\mathbf{B}^4 = \mathbf{D}^4 \mathbf{H}_1^4$$

da Figura 4.1, que corresponde à parte superior do diagrama. Indo para o ramo inferior, existem dois caminhos: $\mathbf{L}^4\mathbf{B}^2$ e $\mathbf{L}^4\mathbf{L}^2$.

O próximo passo é calcular $\mathbf{L}^4$:

$$\mathbf{L}^4 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

$\mathbf{L}^4$ será multiplicada por $\mathbf{B}^2$ e por $\mathbf{L}^2$ para obter $\mathbf{X}_{\mathbf{H}}^1$ e $\mathbf{X}_{\mathbf{L}}^1$, respectivamente. A matriz $\mathbf{D}^2$ é dada por:

$$\mathbf{D}^2 = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

As matrizes $\mathbf{H}_0^2$ e $\mathbf{H}_1^2$ são dadas por:

$$\mathbf{H}_0^2 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} \end{pmatrix}$$

$$\mathbf{H}_1^2 = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{1}{2} \end{pmatrix}$$

Para obter $\mathbf{B}^2$, deve-se realizar a operação $\mathbf{D}^2 \times \mathbf{H}_1^2$. O resultado é:

$$\mathbf{B}^2 = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \end{pmatrix}$$

Para calcular $\mathbf{L}^2$, multiplica-se $\mathbf{D}^2$ por $\mathbf{H}_0^2$:

$$\mathbf{L}^2 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Finalmente, multiplica-se $\mathbf{B}^2$ por $\mathbf{L}^4$ para obter:

$$\mathbf{B}^2\mathbf{L}^4 = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \end{pmatrix}$$

E aqui está $\mathbf{L}^2\mathbf{L}^4$:

$$\mathbf{L}^2\mathbf{L}^4 = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

O passo final é montar a matriz da transformada de dois níveis:

$$\mathbf{T}^{i-2} = \begin{pmatrix} \mathbf{L}^2\mathbf{L}^4 \\ \mathbf{B}^2\mathbf{L}^4 \\ \mathbf{B}^4 \end{pmatrix}$$

que, para o exemplo, fica

$$\mathbf{T}^{i-2} = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} \end{pmatrix}$$

Entrando com o vetor

$$\mathbf{x}^4 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

Multiplicando este vetor por $\mathbf{T}^{i-2}$, obtém-se:

$$\mathbf{T}^{i-2}\mathbf{x}^4 = \begin{pmatrix} \frac{5}{2} \\ -1 \\ -\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}$$

O primeiro elemento do vetor transformado é a média do sinal, como esperado.

4.5.3 Bancos de filtros ortogonais e bases de ondaletas

Diz-se que um banco de filtro é ortogonal se sua matriz correspondente é ortogonal.

Uma matriz $\mathbf{H}$ é ortogonal se sua inversa é igual à sua transposta:

$$\mathbf{H}^T\mathbf{H} = \mathbf{H}\mathbf{H}^T = \mathbf{I}$$

em que $\mathbf{I}$ é a matriz identidade.

Com um escalamento adequado da entrada antes das filtragens passa-baixa e passa-alta, o banco de filtros de Haar torna-se ortogonal. Por exemplo, para $n = 4$, tem-se:

$$\mathbf{H}^n = \mathbf{H}^4 = \begin{bmatrix} \mathbf{L}^4 \\ \mathbf{B}^4 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

e

$$(\mathbf{H}^n)^T\mathbf{H}^n = \mathbf{H}^n(\mathbf{H}^n)^T = \frac{1}{2}I$$

Portando, a matriz $\sqrt{2}\mathbf{H}^n$ é ortogonal. Um banco de filtros de Haar que inclua um escalonamento por $\sqrt{2}$ antes das operações passa-baixa e passa-alta é ortogonal.

No caso do exemplo da última seção, deve-se multiplicar as matrizes $\mathbf{H}_0$ e $\mathbf{H}_1$ por $\sqrt{2}$.

Assim, obtém-se:

$$\mathbf{H}_0^n = \frac{\sqrt{2}}{2} \begin{bmatrix} 1 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}_{n \times n}$$

e

$$\mathbf{H}_1^n = \frac{\sqrt{2}}{2} \begin{bmatrix} 1 & -1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & -1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & -1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}_{n \times n}$$

Usando essa nova informação, pode-se recalcular a matriz de transformada de dois

níveis $\mathbf{T}^{i-2}$:

$$\mathbf{T}^{i-2} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{2} & \frac{-1}{2} \\ \frac{\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} & 0 & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} \end{pmatrix}$$

A idéia por trás do uso de uma matriz ortogonal é encontrar a matriz da transformada inversa. Tudo o que se necessita fazer é encontrar a matriz transposta da matriz da transformada direta, pois:

$$\mathbf{Y} = \mathbf{W} \times \mathbf{X}$$

se, e somente se

$$\mathbf{X} = \mathbf{W}^T \times \mathbf{Y}$$

em que $\mathbf{W}$ é a matriz ortogonal da transformada direta de ondaleta.

4.6 Combinando filtro de Kalman com ondaletas

Nesta seção, considera-se uma forma de combinar as ondaletas com o filtro de Kalman com o objetivo de melhorar a capacidade preditiva do filtro de Kalman. A referência básica para esta seção é o trabalho de Hong et al. (HONG; CHENG; CHUI, 1998).

Considere-se uma seqüência de sinais aleatórios $x(n, k)$ governados pelo seguinte processo estocástico:

$$x(n, k + 1) = A(n, k)x(n, k) + w(n, k) \quad (4.13)$$

Esta seqüência é tomada no mais alto nível de resolução (nível n) e medida como

$$y(n, k) = C(n, k)x(n, k) + z(n, k) \quad (4.14)$$

em que $w(n, k)$ e $z(n, k)$ representam as incertezas de modelagem e da medição, cujas médias e variâncias são dadas por

$$E\{w(n, k)\} = 0$$

$$E\{w^2(n, k)\} = Q(n, k)$$

$$E\{z(n, k)\} = 0$$

$$E\{z^2(n, k)\} = R(n, k)$$

Serão analisados somente dois níveis de estimação e decomposição, ou seja, do nível n aos níveis $\frac{n}{2}$ e $\frac{n}{4}$. Tome-se um bloco de dados com tamanho $m = 2^2 = 4$ até o momento k . Lembrando que $m = 4$ é o tamanho mínimo de bloco necessário para processar os dois níveis. Caso fossem necessários três níveis, seria preciso um tamanho de $m = 2^3 = 8$. Em geral, para n níveis, é necessário um tamanho de 2^n .

$$\mathbf{x}_k^n = \begin{bmatrix} x(n, k-3) \\ x(n, k-2) \\ x(n, k-1) \\ x(n, k) \end{bmatrix}$$

Das equações (4.13) e (4.14) serão retirados os indicadores de níveis das matrizes $\mathbf{A}$ e

$\mathbf{C}$, assumindo que elas sejam invariantes com o tempo.

$$x(n, k+1) = \mathbf{A}x(n, k) + w(n, k) \quad (4.15)$$

Chamando $x(n, k)$ recursivamente

$$\begin{aligned} x(n, k+1) &= A(Ax(n, k-1) + w(n, k-1)) + w(n, k) \\ &= A^2x(n, k-1) + Aw(n, k-1) + w(n, k) \end{aligned} \quad (4.16)$$

e aplicando (4.15) novamente a $x(n, k-1)$ em (4.16), tem-se

$$\begin{aligned} x(n, k+1) &= A^2x(n, k-1) + Aw(n, k-1) + w(n, k) \\ &= A^2(Ax(n, k-2) + w(n, k-2)) + Aw(n, k-1) + w(n, k) \\ &= A^3x(n, k-2) + A^2w(n, k-2) + Aw(n, k-1) + w(n, k) \end{aligned} \quad (4.17)$$

repetindo a operação uma vez mais, tem-se

$$x(n, k+1) = A^4x(n, k-3) + A^3w(n, k-3) + A^2w(n, k-2) + Aw(n, k-1) + w(n, k) \quad (4.18)$$

Adicionado as equações (4.15), (4.16), (4.17) e (4.18) e dividindo ambos os lados por

quatro, obtém-se:

$$\begin{aligned}
 x(n, k+1) &= \frac{1}{4}A^4x(n, k-3) + \frac{1}{4}A^3x(n, k-2) \\
 &\quad + \frac{1}{4}A^2x(n, k-1) + \frac{1}{4}Ax(n, k) \\
 &\quad + \frac{1}{4}A^3w(n, k-3) + \frac{1}{2}A^2w(n, k-2) \\
 &\quad + \frac{3}{4}Aw(n, k-1) + w(n, k)
 \end{aligned} \tag{4.19}$$

Propagando para $x(n, k+2)$

$$x(n, k+2) = Ax(n, k+1) + w(n, k+1) \tag{4.20}$$

Chamando $x(n, k+1)$ recursivamente

$$\begin{aligned}
 x(n, k+2) &= A(Ax(n, k) + w(n, k)) + w(n, k+1) \\
 &= A^2x(n, k) + Aw(n, k) + w(n, k+1)
 \end{aligned} \tag{4.21}$$

Esta equação pode ser reescrita como:

$$\begin{aligned}
 x(n, k+2) &= A^2(Ax(n, k-1) + w(n, k-1)) + Aw(n, k) + w(n, k+1) \\
 &= A^3x(n, k-1) + A^2w(n, k-1) + Aw(n, k) + w(n, k+1)
 \end{aligned} \tag{4.22}$$

Ou como:

$$\begin{aligned}
x(n, k+2) &= A^3(Ax(n, k-2) + w(n, k-2)) + A^2w(n, k-1) \\
&\quad + Aw(n, k) + w(n, k+1) \\
&= A^4x(n, k-2) + A^3w(n, k-2) + A^2w(n, k-1) \\
&\quad + Aw(n, k) + w(n, k+1)
\end{aligned} \tag{4.23}$$

Adicionando as equações (4.21), (4.22) e (4.23) e dividindo ambos os lados por três, obtém-se:

$$\begin{aligned}
x(n, k+2) &= \frac{1}{3}A^4x(n, k-2) + \frac{1}{3}A^3x(n, k-1) + \frac{1}{3}A^2x(n, k) \\
&\quad + \frac{1}{3}A^3w(n, k-2) + \frac{2}{3}A^2w(n, k-1) \\
&\quad + Aw(n, k) + w(n, k+1)
\end{aligned} \tag{4.24}$$

Operações análogas podem ser realizadas para se encontrar $x(n, k+3)$ e $x(n, k+4)$:

$$\begin{aligned}
x(n, k+3) &= \frac{1}{2}A^4x(n, k-1) + \frac{1}{2}A^3x(n, k) + \frac{1}{2}A^3w(n, k-1) \\
&\quad + A^2w(n, k) + Aw(n, k+1) + w(n, k+2)
\end{aligned} \tag{4.25}$$

$$\begin{aligned}
x(n, k+4) &= A^4x(n, k) + A^3w(n, k) + A^2w(n, k+1) \\
&\quad + Aw(n, k+2) + w(n, k+3)
\end{aligned} \tag{4.26}$$

Reunindo as equações (4.19), (4.24), (4.25) e (4.26) sob a forma matricial, tem-se:

$$\underbrace{\begin{bmatrix} x(n, k+1) \\ x(n, k+2) \\ x(n, k+3) \\ x(n, k+4) \end{bmatrix}}_{\mathbf{x}_{k+1}^n} = \underbrace{\begin{bmatrix} \frac{1}{4}A^4 & \frac{1}{4}A^3 & \frac{1}{4}A^2 & \frac{1}{4}A \\ 0 & \frac{1}{3}A^4 & \frac{1}{3}A^3 & \frac{1}{3}A^2 \\ 0 & 0 & \frac{1}{2}A^4 & \frac{1}{2}A^3 \\ 0 & 0 & 0 & A^4 \end{bmatrix}}_{\bar{\mathbf{A}}} \times \underbrace{\begin{bmatrix} x(n, k-3) \\ x(n, k-2) \\ x(n, k-1) \\ x(n, k) \end{bmatrix}}_{\mathbf{x}_k^n} + \mathbf{W} \quad (4.27)$$

em que

$$\mathbf{W} = \begin{bmatrix} \frac{1}{4}A^3w(n, k-3) + \frac{1}{2}A^2w(n, k-2) + \frac{3}{4}Aw(n, k-1) + w(n, k) \\ \frac{1}{3}A^3w(n, k-2) + \frac{2}{3}A^2w(n, k-1) + Aw(n, k) + w(n, k+1) \\ \frac{1}{2}A^3w(n, k-1) + A^2w(n, k) + Aw(n, k+1) + w(n, k+2) \\ A^3w(n, k) + A^2w(n, k+1) + Aw(n, k+2) + w(n, k+3) \end{bmatrix} \quad (4.28)$$

com

$$E\{\mathbf{W}\} = 0, \quad E\{\mathbf{W}\mathbf{W}^T\} = \mathbf{Q}$$

em que

$$\mathbf{Q} = Q \begin{bmatrix} \frac{1}{16}A^6 + \frac{1}{4}A^4 + \frac{9}{16}A^2 + 1 & \frac{1}{6}A^5 + \frac{1}{2}A^3 + A & \frac{3}{8}A^4 + A^2 & A^3 \\ \frac{1}{6}A^5 + \frac{1}{2}A^3 + A & \frac{1}{9}A^6 + \frac{4}{9}A^4 + A^2 + 1 & \frac{1}{3}A^5 + A^3 + A & A^4 + A^2 \\ \frac{3}{8}A^4 + A^2 & \frac{1}{3}A^5 + A^3 + A & \frac{1}{4}A^6 + A^4 + A^2 + 1 & A^5 + A^3 + A \\ A^3 & A^4 + A^2 & A^5 + A^3 + A & A^6 + A^4 + A^2 + 1 \end{bmatrix}$$

Na última matriz, pode-se usar o fato de que

$$E\{w(n, j)w(n, k)\} = \begin{cases} 0 & \text{if } j \neq k \\ Q & \text{if } j = k \end{cases}$$

O modelo de medição associado com a equação (4.27) pode ser construído por

$$\underbrace{\begin{bmatrix} y(n, k-3) \\ y(n, k-2) \\ y(n, k-1) \\ y(n, k) \end{bmatrix}}_{\mathbf{y}_k^n} = \underbrace{\begin{bmatrix} C & 0 & 0 & 0 \\ 0 & C & 0 & 0 \\ 0 & 0 & C & 0 \\ 0 & 0 & 0 & C \end{bmatrix}}_{\mathbf{C}} \underbrace{\begin{bmatrix} x(n, k-3) \\ x(n, k-2) \\ x(n, k-1) \\ x(n, k) \end{bmatrix}}_{\mathbf{x}_k^n} + \underbrace{\begin{bmatrix} z(n, k-3) \\ z(n, k-2) \\ z(n, k-1) \\ z(n, k) \end{bmatrix}}_{\mathbf{z}_k^n}$$

em que

$$E\{\mathbf{z}_k^n\} = 0 \quad \text{e} \quad E\{\mathbf{z}_k^n(\mathbf{z}_k^n)^T\} = \mathbf{R}$$

e

$$\mathbf{R} = \begin{bmatrix} R & 0 & 0 & 0 \\ 0 & R & 0 & 0 \\ 0 & 0 & R & 0 \\ 0 & 0 & 0 & R \end{bmatrix}$$

A decomposição em dois níveis é dada pela seguinte equação:

$$\begin{bmatrix} \mathbf{x}_{k_L}^{\frac{n}{4}} \\ \mathbf{x}_{k_H}^{\frac{n}{4}} \\ \mathbf{x}_{k_H}^{\frac{n}{2}} \end{bmatrix} = \begin{bmatrix} \mathbf{L}^2 \mathbf{L}^4 \\ \mathbf{B}^2 \mathbf{L}^4 \\ \mathbf{B}^4 \end{bmatrix} \mathbf{x}_k^n = \mathbf{T}^{n-2} \mathbf{x}_k^n \quad (4.29)$$

A equação (4.29) é equivalente a

$$\mathbf{x}_k^n = (\mathbf{T}^{n-2})^T \begin{bmatrix} \mathbf{x}_{k_L}^{\frac{n}{4}} \\ \mathbf{x}_{k_H}^{\frac{n}{4}} \\ \mathbf{x}_{k_H}^{\frac{n}{2}} \end{bmatrix} \quad (4.30)$$

pois a transposta de uma matriz ortogonal é sua inversa.

Substituindo a equação (4.30) na equação (4.27), resulta-se em

$$\mathbf{x}_{k+1}^n = \bar{A} \left((\mathbf{T}^{n-2})^T \begin{bmatrix} \mathbf{x}_{k_L}^{\frac{n}{4}} \\ \mathbf{x}_{k_H}^{\frac{n}{4}} \\ \mathbf{x}_{k_H}^{\frac{n}{2}} \end{bmatrix} \right) + \mathbf{W}_k^n \quad (4.31)$$

Quando se aplica a transformada de ondaleta em dois níveis à $\mathbf{x}_{k+1}^n$, obtém-se:

$$\begin{bmatrix} \mathbf{x}_{k+1_L}^{\frac{n}{4}} \\ \mathbf{x}_{k+1_H}^{\frac{n}{4}} \\ \mathbf{x}_{k+1_H}^{\frac{n}{2}} \end{bmatrix} = (\mathbf{T}^{n-2}) \left(\bar{A} (\mathbf{T}^{n-2})^T \begin{bmatrix} \mathbf{x}_{k_L}^{\frac{n}{4}} \\ \mathbf{x}_{k_H}^{\frac{n}{4}} \\ \mathbf{x}_{k_H}^{\frac{n}{2}} \end{bmatrix} + \mathbf{W}_k^n \right) \quad (4.32)$$

Simplificando esta última equação, fica-se com

$$\begin{bmatrix} \mathbf{x}_{k+1_L}^{\frac{n}{4}} \\ \mathbf{x}_{k+1_H}^{\frac{n}{4}} \\ \mathbf{x}_{k+1_H}^{\frac{n}{2}} \end{bmatrix} = (\mathbf{T}^{n-2}) \bar{A} (\mathbf{T}^{n-2})^T \begin{bmatrix} \mathbf{x}_{k_L}^{\frac{n}{4}} \\ \mathbf{x}_{k_H}^{\frac{n}{4}} \\ \mathbf{x}_{k_H}^{\frac{n}{2}} \end{bmatrix} + (\mathbf{T}^{n-2}) \mathbf{W}_k^n \quad (4.33)$$

Seja

$$\underline{W}_k^n = (\mathbf{T}^{n-2}) \mathbf{W}_k^n \quad (4.34)$$

em que

$$E\{\underline{W}_k^n\} = 0$$

$$E\{\underline{W}_k^n(\underline{W}_k^n)^T\} = \mathbf{Q}$$

Expandindo esta última expressão

$$\begin{aligned} E\{\underline{W}_k^n(\underline{W}_k^n)^T\} &= E\{T^{n-2}\mathbf{W}_k^n(\mathbf{T}^{n-2})\mathbf{W}_k^n)^T\} \\ &= E\{T^{n-2}\mathbf{W}_k^n(\mathbf{W}_k^n)^T(\mathbf{T}^{n-2})^T\} \\ &= T^{n-2}E\{\mathbf{W}_k^n(\mathbf{W}_k^n)^T\}(\mathbf{T}^{n-2})^T \\ &= T^{n-2}\bar{Q}(\mathbf{T}^{n-2})^T \end{aligned} \tag{4.35}$$

desde que

$$\bar{Q} = E\{\mathbf{W}_k^n(\mathbf{W}_k^n)^T\}$$

Se se substitui a equação (4.30) na equação (4.6), obtém-se o modelo de medição associado:

$$Y_k^n = C \begin{bmatrix} X_{k_L}^{\frac{n}{4}} \\ X_{k_H}^{\frac{n}{4}} \\ X_{k_H}^{\frac{n}{2}} \end{bmatrix} + Z_k^n \tag{4.36}$$

em que

$$C = \bar{C}(T^{n-2})^T$$

As equações (4.33) e (4.36) são equações que governam um sistema estocástico. Pode-se aplicar um filtro de Kalman ao sistema descrito por essas equações para obter estimativas ótimas do sinal transformado.

4.7 O esquema de lifting para ondaletas

Na última seção, usou-se a notação matricial para misturar o filtro de Kalman com as ondaletas. Embora a notação matricial torne mais fácil a manipulação desse tipo de problema, ela é, computacionalmente, ineficiente, pois possui complexidade $O(n^2)$. O estado da arte na implementação das ondaletas usa um esquema denominado *lifting* (SWEL-DENS; SCHRÖDER, 1996), que possui complexidade $O(n)$, que é mais rápida que a transformada rápida de Fourier. Além disto, todas as operações podem ser realizadas sem ocupar espaço extra na memória do computador. Isto pode ser importante, quando se lida com volumes imensos de dados, como é o caso do problema analisado neste trabalho.

Como foi feito para o estudo das ondaletas, será mostrado o uso do esquema de *lifting* para o caso de Haar.

Considerem-se duas amostras vizinhas a e b em uma seqüência. Devido a esta proximidade, existe uma correlação entre a e b . Defina-se a seguinte transformação linear, que substitui a e b pela *média* s e pela *diferença* d entre estes números:

$$\begin{aligned} s &= \frac{a+b}{2} \\ d &= b-a \end{aligned} \tag{4.37}$$

Note-se que a partir dos valores s e d , pode-se recuperar a e b , basta para isto fazer-se:

$$\begin{aligned} a &= s - \frac{d}{2} \\ b &= s + \frac{d}{2} \end{aligned} \tag{4.38}$$

As equações em (4.37) formam a transformada de ondaleta de Haar, enquanto as equações em (4.38) formam a transformada inversa.

Considere-se um sinal s_n com 2^n valores amostrados $s_{n,l}$:

$$s_n = \{s_{n,l} \mid 0 \leq l < 2^n\}$$

A transformada de Haar será aplicada a cada par $a = s_{2l}$ e $b = s_{2l+1}$. Existem 2^{n-1} tais pares ($l = 0 \dots 2^{n-1}$). Os resultados serão denotados por $s_{n-1,l}$ e $d_{n-1,l}$:

$$\begin{aligned} s_{n-1,l} &= \frac{s_{n,2l} + s_{n,2l+1}}{2} \\ d_{n-1,l} &= s_{n,2l+1} - s_{n,2l} \end{aligned}$$

O sinal de entrada s_n , que possui 2^n amostras, é dividido em dois sinais: s_{n-1} , com 2^{n-1} médias $s_{n-1,l}$, e d_{n-1} , com 2^{n-1} diferenças $d_{n-1,l}$. A partir das médias e das diferenças pode-se recuperar o sinal original s_n .

Como visto anteriormente, pode-se pensar no sinal s_{n-1} como a saída de um filtro passa-baixa e o sinal d_{n-1} como a saída de um filtro passa-alta, também chamado de detalhe, pois representa aquilo que se deve acrescentar a s_{n-1} de forma a obter o sinal original s_n .

Pode-se aplicar a transformada de Haar novamente ao sinal s_{n-1} . Tomando-se as médias e as diferenças, pode-se dividi-lo em um novo sinal s_{n-2} e um novo sinal diferença d_{n-2} , tal que cada um deles contenha 2^{n-2} amostras. Pode-se fazer isto n vezes antes de ficar sem amostras, como visto na Figura 4.2. Após o término desse processo, existirão n sinais detalhes d_j com $0 \leq j < n - 1$, cada um dos quais com 2^j coeficientes, e um sinal

s_0 , no menor nível de resolução. O sinal s_0 representa o nível DC ou frequência zero do sinal.

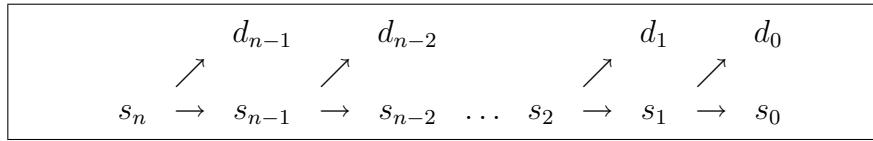


FIGURA 4.2 – Estrutura da transformada de ondaleta: recursivamente dividida em médias e diferenças

Para usar a transformada inversa de Haar, parte-se de s_0 e d_j , com $0 \leq j < n - 1$ até obter o sinal s_n original, como ilustrado na Figura 4.3.

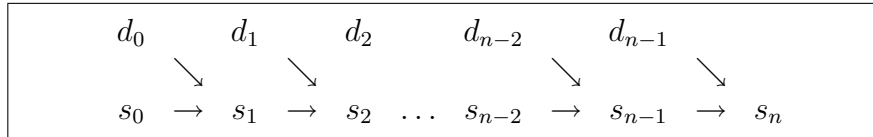


FIGURA 4.3 – Estrutura da transformada inversa de ondaleta: recursivamente reúne as médias e diferenças

Até este ponto, tudo o que foi visto não é muito diferente da exposição sobre ondaletas da Seção 4. A novidade consistirá no modo de calcular a diferença e a média de dois números a e b . Suponha-se que se deseje calcular toda a transformada de ondaleta sem usar qualquer memória adicional, sobrescrevendo as posições que contenham os valores a e b com as suas médias e diferenças, respectivamente. Não se pode utilizar, diretamente, as equações (4.37), pois isto causaria erro. Um plano alternativo será adotado:

1. Calcula-se a diferença

$$d = b - a$$

que será armazenada na posição de memória em que b se encontra.

2. Calcula-se a média usando

$$s = a + \frac{d}{2}$$

que será armazenada na posição de a .

Essas duas equações resultam nos mesmos valores produzidos pela equação (4.37), pois:

$$\begin{aligned} s &= a + \frac{d}{2} \\ &= a + \frac{b-a}{2} \\ &= \frac{a+b}{2} \end{aligned}$$

Estas equações poderiam ser implementadas em dois passos:

$$\begin{aligned} b &\leftarrow b - a \\ a &\leftarrow a + b/2 \end{aligned}$$

A vantagem em se calcular a transformada em dois passos, como feito acima, consiste na possibilidade de sobrescrever b com d e a com s , sem usar memória adicional. Além disto, pode-se calcular a transformada inversa apenas trocando os sinais das operações e invertendo a ordem.

Supondo que a contém a média e b contém a diferença, então, a seqüência

$$\begin{aligned} a &\leftarrow a - \frac{b}{2} \\ b &\leftarrow b + a \end{aligned}$$

recupera os valores de a e b em suas posições originais de memória. Este modo particular de escrever uma transformada é um exemplo simples do uso do esquema de *lifting*.

5 Aproximação de curvas e superfícies

O problema da reconstrução de superfícies suaves a partir de dados discretos e esparsos aparece em muitos campos da ciência e engenharia e tem sido estudado por quase cinquenta anos. Dentre as várias fontes fornecedoras dos dados, incluem-se valores medidos (Meteorologia, Oceanografia, Ótica, Geologia etc.), resultados experimentais (da Física, Química ou Engenharia) e valores computacionais (avaliação de funções matemáticas, soluções por elementos finitos de equações diferenciais parciais ou resultados de outras simulações numéricas). Tal variedade de fontes deu origem a diferentes métodos, cada um deles adaptado a um problema específico (HORMANN, 2000).

A abordagem que interessa a este trabalho é ajustar uma superfície a uma nuvem de pontos de dados esparsos e com ruído. Pode-se realizar o ajuste de dois modos, interpolação e aproximação (ZHOU, 1999). No caso da interpolação, os dados são parte da superfície. Assim, a interpolação não é apropriada para uma nuvem de pontos de dados que possuam ruídos de medição. Na aproximação, a idéia é encontrar uma superfície que minimize o erro total entre o modelo e a nuvem.

5.1 Polinômios de Bernstein

Os polinômios de Bernstein foram criados pelo matemático ucraniano Sergei Natanovich Bernstein, que os utilizou em uma demonstração do teorema de aproximação de Weierstrass (BERNSTEIN, 1912). Esses polinômios podem representar funções monotônicas suaves que possuam derivadas à esquerda e à direita, em qualquer ponto, sem a ocorrência do fenômeno de Gibbs (GZYL; PALACIOS, 2003).

Os polinômios de Bernstein possibilitam aproximações globais de dados em contraste a métodos mais populares de aproximações locais, como as *splines* (ZHOU, 1999). Eles são particularmente úteis na modelagem de conjuntos de pontos de dados incompletos e com ruídos (que não respondem bem às aproximações locais) e são utilizados para representações compactas de curvas e superfícies na modelagem geométrica auxiliada por computador em decorrência de suas propriedades de monotonicidade e de preservação de forma (FARIN, 2002).

Os polinômios de Bernstein de grau n são definidos por

$$B_k^n(u) = \binom{n}{k} u^k (1-u)^{(n-k)}$$

para $k = 0, 1, \dots, n$, em que

$$\binom{n}{k} = \frac{n!}{k! \times (n-k)!} \quad (5.1)$$

Alguns exemplos de polinômios de Bernstein:

- Com grau 1:

$$B_0^1(u) = 1 - u$$

$$B_1^1(u) = u$$

- Com grau 2:

$$B_0^2(u) = (1 - u)^2$$

$$B_1^2(u) = 2u(1 - u)$$

$$B_2^2(u) = u^2$$

- Com grau 3:

$$B_0^3(u) = (1 - u)^3$$

$$B_1^3(u) = 3u(1 - u)^2$$

$$B_2^3(u) = 3u^2(1 - u)$$

$$B_3^3(u) = u^3$$

5.1.1 Propriedades dos polinômios de Bernstein

Os polinômios de Bernstein possuem as seguintes propriedades básicas (LORENTZ, 1986):

1. Todos os polinômios de Bernstein são não-negativos quando $0 \leq u \leq 1$;
2. Os $n + 1$ polinômios de Bernstein de grau n formam uma partição da unidade, i.e, se somados, o resultado é 1 para todos os valores de u . Esta propriedade é importante, pois, para todo u e para um conjunto de pontos $\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_n$, a expressão

$$\mathbf{P}(u) = \mathbf{P}_0 B_0^n(u) + \mathbf{P}_1 B_1^n(u) + \dots + \mathbf{P}_n B_n^n(u)$$

é uma combinação afim¹ do conjunto de pontos $\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_n$ e se $0 \leq u \leq 1$,

¹Sejam $v_1, \dots, v_k$ k vetores de $\mathbb{R}^n$. Sejam $\lambda_1, \dots, \lambda_k$ k escalares satisfazendo $\sum_{i=1}^k \lambda_i = 1$. O vetor

então, ela também é uma combinação convexa² destes pontos.

3. Os polinômios de Bernstein de ordem n formam uma base para o espaço de polinômios de grau menor ou igual a n .
4. Estes polinômios obedecem à seguinte relação de recorrência bastante útil em aplicações computacionais:

$$B_k^n(u) = (1 - u)B_k^{n-1}(u) + uB_{k-1}^{n-1}(u) \quad (5.2)$$

5.2 Curvas de Bézier

As curvas de Bézier foram descobertas, de maneira independente, por Paul de Casteljau, na Citroën, e por Pierre E. Bézier, na Renault, em torno de 1960.

Uma *curva de Bézier* de grau p é composta por pontos $\mathbf{C}$ definidos por

$$\mathbf{C}(t) = \sum_{i=0}^p B_i^p(t) \mathbf{P}_i \quad (5.3)$$

em que $\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_p$ são $p + 1$ *pontos de controle*, $B_i^p(t)$ é um polinômio da base de Bernstein e t pertence ao intervalo $[0, 1]$.

Como pode ser observado na equação 5.3, o ponto que corresponde ao parâmetro u sobre a curva de Bézier é a média ponderada de todos os pontos de controle, em que os pesos são os coeficientes B_i^p . Os segmentos de linha $\mathbf{P}_0\mathbf{P}_1, \mathbf{P}_1\mathbf{P}_2, \dots, \mathbf{P}_{p-1}\mathbf{P}_p$, juntos nesta ordem, formam o *polígono de controle* (SHENE, 2003).

$v = \sum_{i=1}^k \lambda_i v_i$ é uma combinação afim de $v_1, \dots, v_k$.

²Uma combinação convexa é uma combinação afim com o requisito adicional de todos os escalares λ_i serem *não negativos*.

5.2.1 Parametrização de curvas

Dada a importância da propriedade 2 dos polinômios de Bernstein, vai-se impor às abscissas uma variação entre 0 e 1. Uma forma de alcançar isto é parametrizando u_i por meio de t_i . Pode-se criar esses parâmetros com as seguintes especificidades:

1. $0 \leq t_i \leq 1$ sempre que $u_0 \leq u \leq u_p$, em que u_0 e u_p são o menor e o maior valor possível para u , respectivamente.
2. $t_0 < t_1 < \dots < t_p$, com t_0 associado a u_0 , t_1 a u_1 , etc.

Um método popular de parametrização de curvas e superfícies é o método do comprimento da corda.

Suponha-se que os pontos de dados sejam $\mathbf{D}_0, \mathbf{D}_1, \dots, \mathbf{D}_n$. O comprimento da corda ligando $\mathbf{D}_{i-1}$ a $\mathbf{D}_i$ é $|\mathbf{D}_i - \mathbf{D}_{i-1}|$. Desta forma, o comprimento total da cordas ligando estes pontos é a soma dos comprimentos individuais dessas cordas:

$$L = \sum_{i=1}^n |\mathbf{D}_i - \mathbf{D}_{i-1}|$$

A razão entre o comprimento da corda entre os pontos de dados $\mathbf{D}_0$ a $\mathbf{D}_k$ e o comprimento total das cordas, denotada por L_k , é:

$$L_k = \frac{\sum_{i=1}^k |\mathbf{D}_i - \mathbf{D}_{i-1}|}{L}$$

Assim, se o domínio do parâmetro t_k é $[0, 1]$, então, este parâmetro deve estar localizado no valor de L_k :

$$t_0 = 0$$

$$t_k = \frac{1}{L}(\sum_{i=1}^k |\mathbf{D}_i - \mathbf{D}_{i-1}|) = t_{k-1} + \frac{1}{L}(|\mathbf{D}_k - \mathbf{D}_{k-1}|)$$

$$t_n = 1$$

Procedendo-se da maneira descrita acima, os parâmetros dividem o domínio na razão entre os comprimentos das cordas.

5.2.2 Aproximação global de curvas

O problema que esta seção trata pode ser descrito como: dados um conjunto de $n + 1$ pontos de dados, $\mathbf{D}_0, \mathbf{D}_1, \dots, \mathbf{D}_n$, um conjunto de $n + 1$ parâmetros $t_0, t_1, \dots, t_n$ e um grau p , encontrar uma curva de Bézier de grau p definida por $p + 1$ pontos de controle tal que esta curva aproxime-se do polígono de dados no sentido dos mínimos quadrados.

A curva de Bézier de grau p que aproxima os dados é da forma

$$\mathbf{C}(u) = \sum_{i=0}^p B_i^p(u) \mathbf{P}_i$$

em que $\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_p$ são os $p + 1$ pontos de controle incógnitos, e $B_i^p(u)$ é um polinômio da base de Bernstein dado por

$$B_i^p(u) = \binom{p}{i} (1-u)^{p-i} u^i$$

Defina-se a seguinte função

$$\chi^2(\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_p) = \sum_{k=0}^n \|\mathbf{D}_k - \mathbf{C}(t_k)\|^2$$

O objetivo é encontrar os pontos de controle $\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_p$ que minimizam essa função, e, desta forma, a aproximação é feita no sentido dos *mínimos quadrados*.

5.2.2.1 Resolução pelas equações normais

Lembrando que o produto escalar $\mathbf{x} \cdot \mathbf{x} = \|\mathbf{x}\|^2$, é possível reescrever a norma quadrada da função χ^2 como

$$\begin{aligned} \|\mathbf{D}_k - \mathbf{C}(t_k)\|^2 &= \left\| \mathbf{D}_k - \sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \right\|^2 \\ &= \left(\mathbf{D}_k - \sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \right) \cdot \left(\mathbf{D}_k - \sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \right) \\ &= \mathbf{D}_k \cdot \mathbf{D}_k - 2 \left(\sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \cdot \mathbf{D}_k \right) + \left(\sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \right) \cdot \left(\sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \right) \end{aligned}$$

Assim, a função χ^2 torna-se

$$\chi^2(\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_p) = \sum_{k=0}^n \left[\mathbf{D}_k \cdot \mathbf{D}_k - 2 \left(\sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \cdot \mathbf{D}_k \right) + \left(\sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \right) \cdot \left(\sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \right) \right]$$

Uma forma de minimizar esta função é derivando-a em relação a cada um dos pontos de controle $\mathbf{P}_g$. Depois, iguala-se cada derivada parcial a zero. Os valores $\mathbf{P}_g$ para os quais as derivadas são nulas são aqueles que minimizam a função. Assim, tem-se que

$$\frac{\partial \chi^2(\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_p)}{\partial \mathbf{P}_g} = 0, \quad g = 0, 1, \dots, p$$

Calculando as derivadas parciais tem-se

$$\begin{aligned}
& \frac{\partial \chi^2(\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_p)}{\partial \mathbf{P}_g} \\
&= \sum_{k=0}^n \left[\frac{\partial \mathbf{D}_k \cdot \mathbf{D}_k}{\partial \mathbf{P}_g} - 2 \frac{\partial \left(\sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \cdot \mathbf{D}_k \right)}{\partial \mathbf{P}_g} \right. \\
&\quad \left. + \frac{\partial \left(\sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \right) \cdot \left(\sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \right)}{\partial \mathbf{P}_g} \right] \\
&= \sum_{k=0}^n \left[-2B_g^p(t_k) \mathbf{D}_k + 2B_g^p(t_k) \left(\sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \right) \right]
\end{aligned}$$

Igualando esta equação a zero e rearranjando, chega-se ao seguinte:

$$\sum_{k=0}^n B_g^p(t_k) \left(\sum_{i=0}^p B_i^p(t_k) \mathbf{P}_i \right) = \sum_{k=0}^n B_g^p(t_k) \mathbf{D}_k, \quad g = 0, 1, \dots, p$$

Reescrevendo esta equação de forma a realçar os coeficientes de $\mathbf{P}_i$, tem-se

$$\sum_{i=0}^p \left(\sum_{k=0}^n B_g^p(t_k) B_i^p(t_k) \right) \mathbf{P}_i = \sum_{k=0}^n B_g^p(t_k) \mathbf{D}_k, \quad g = 0, 1, \dots, p$$

Agora, definindo-se as seguintes matrizes

$$\begin{aligned}
 \mathbf{P} &= \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \vdots \\ \mathbf{P}_p \end{bmatrix}_{(p+1) \times 1} \\
 \mathbf{Q} &= \begin{bmatrix} \sum_{k=0}^n B_0^p(t_k) \mathbf{D}_k \\ \sum_{k=0}^n B_1^p(t_k) \mathbf{D}_k \\ \vdots \\ \sum_{k=0}^n B_p^p(t_k) \mathbf{D}_k \end{bmatrix}_{(p+1) \times 1} \\
 \mathbf{B} &= \begin{bmatrix} B_0^p(t_0) & B_1^p(t_0) & \cdots & B_p^p(t_0) \\ B_0^p(t_1) & B_1^p(t_1) & \cdots & B_p^p(t_1) \\ \vdots & & \ddots & \vdots \\ B_0^p(t_n) & B_1^p(t_n) & \cdots & B_p^p(t_n) \end{bmatrix}_{(n+1) \times (p+1)}
 \end{aligned}$$

Com o auxílio destas matrizes, pode-se reescrever o sistema de equações como

$$(\mathbf{B}^T \mathbf{B}) \mathbf{P} = \mathbf{Q} \quad (5.4)$$

Esta equação, ao ser resolvida para $\mathbf{P}$, fornece os pontos de controle desejados.

5.2.2.2 Suavização da curva de ajuste

O método dos mínimos quadrados somente objetiva minimizar a função χ^2 , sem levar em conta a forma da curva resultante. Um modo de suavizar a forma da curva é observar

a forma do seu polígono de controle. Se o polígono comportar-se conforme esperado, assim também será com a curva de aproximação.

Pode-se conceber várias formas de fazer a medição da forma do polígono de controle. Neste trabalho, usa-se o método das diferenças segundas. Se todas as diferenças segundas dos pontos de controle $\Delta^2 \mathbf{P}_i = \mathbf{P}_{i+2} - 2\mathbf{P}_{i+1} + \mathbf{P}_i$ forem pequenas, então, o polígono de controle não oscilará muito (FARIN, 2002). Em adição às equações do sistema em (5.4), pode-se requerer que:

$$\begin{aligned} \mathbf{P}_0 - 2\mathbf{P}_1 + \mathbf{P}_2 &= \mathbf{0} \\ \vdots \\ \mathbf{P}_{n-2} - 2\mathbf{P}_{n-1} + \mathbf{P}_n &= \mathbf{0} \end{aligned}$$

Colocando este sistema na forma matricial,

$$\underbrace{\begin{pmatrix} 1 & -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ \vdots & & & \vdots & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 1 & -2 & 1 \end{pmatrix}}_{\mathbf{S}} \underbrace{\begin{pmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \vdots \\ \mathbf{P}_n \end{pmatrix}}_{\mathbf{P}} = \underbrace{\begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{\mathbf{0}}$$

Ou,

$$\mathbf{S}_{(p-1) \times (p+1)} \mathbf{P}_{(p+1) \times 1} = \mathbf{0}_{(p-1) \times 1} \quad (5.5)$$

Anexando-se estas equações ao sistema original $(\mathbf{B}^T \mathbf{B}) \mathbf{P} = \mathbf{Q}$, obter-se-á

$$\begin{pmatrix} \mathbf{B}^T \mathbf{B} \\ \mathbf{S} \end{pmatrix} \mathbf{P} = \begin{pmatrix} \mathbf{Q} \\ \mathbf{0} \end{pmatrix} \quad (5.6)$$

Este sistema é resolvido do mesmo modo que (5.4), ou seja, formando um sistema linear simétrico de equações normais. O sistema é solúvel, pois a matriz de coeficientes de (5.6) ainda possui $p + 1$ colunas linearmente independentes, herdadas da matriz inicial $\mathbf{B}^T \mathbf{B}$.

Na equação (5.6), tem-se pouco controle sobre o efeito das equações adicionadas para controlar a forma da curva. Aumenta-se o controle pela adição de um fator de ponderação ω aos dois diferentes componentes:

$$\underbrace{\begin{pmatrix} (1 - \omega)\mathbf{B}^T \mathbf{B} \\ \omega \mathbf{S} \end{pmatrix}}_{\mathbf{N}_{(2p) \times (p+1)}} \mathbf{P} = \underbrace{\begin{pmatrix} (1 - \omega)\mathbf{Q} \\ \mathbf{0} \end{pmatrix}}_{\mathbf{M}_{(2p) \times 1}} \quad (5.7)$$

Para $0 \leq \omega < 1$, tem-se uma ponderação entre as equações de ajuste inicial em (5.6) e as equações que otimizam a forma (5.5). Para $\omega = 0$, obtém-se (5.4); para valores de ω próximos de 0, dá-se um peso maior às equações de ajuste. Para valores de ω próximos de 1, abandona-se o ajuste (ver Figura 5.1).

Para completar o algoritmo, vai-se resolver a equação (5.7):

$$\begin{aligned} \mathbf{N} \cdot \mathbf{P} &= \mathbf{M} \\ \mathbf{N}^T \cdot \mathbf{N} \cdot \mathbf{P} &= \mathbf{N}^T \cdot \mathbf{M} \\ (\mathbf{N}^T \cdot \mathbf{N})^{-1} \cdot (\mathbf{N}^T \cdot \mathbf{N}) \cdot \mathbf{P} &= (\mathbf{N}^T \cdot \mathbf{N})^{-1} \cdot \mathbf{N}^T \cdot \mathbf{M} \\ \mathbf{P} &= (\mathbf{N}^T \cdot \mathbf{N})^{-1} \cdot \mathbf{N}^T \cdot \mathbf{M} \end{aligned} \quad (5.8)$$

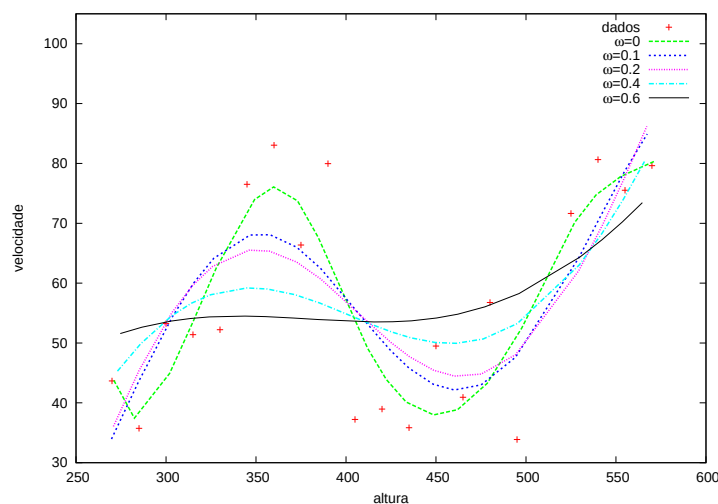


FIGURA 5.1 – Aproximação de dados usando as curvas de Bézier com suavização pelo método das diferenças segundas. O grau dos polinômios de Bernstein em cada curva é 6. Quando ω varia de 0 a 1, a curva se suaviza, porém o ajuste é deteriorado.

5.3 Superfícies de Bézier

A idéia de uma *superfície de Bézier* é a de uma curva que se move no espaço e que, durante este movimento, sofre deformação. Para mover uma curva, é necessário mover os seus pontos de controle. A trajetória definida pelo movimento de cada um destes pontos de controle é, ela mesma, uma outra curva. Esta segunda curva possui, então, seus próprios pontos de controle. Logo, tem-se uma curva, definida pelos seus pontos de controle, e sua trajetória, também definida por pontos de controle. A descrição matemática de uma superfície de Bézier, composta por pontos $\mathbf{S}$ definidos por um conjunto de pontos de controle $\mathbf{P}_{ij}$, é dada pela equação

$$S(u, v) = \sum_{i=0}^p \sum_{j=0}^q B_i^p(t_1) B_j^q(t_2) \mathbf{P}_{ij} \quad (5.9)$$

Os parâmetros t_1 e t_2 variam entre $[0, 1]$. $B_i^p(t_1)$ e $B_j^q(t_2)$ são polinômios de Bernstein de graus p e q , avaliados em t_1 e t_2 .

Ao conjunto de pontos de controle é dado o nome de *malha de controle*. No caso de uma superfície em $\mathbb{R}^3$, os pontos da malha de controle são tridimensionais. Os índices i e j correspondem aos índices de uma matriz bidimensional com $p + 1$ linhas e $q + 1$ colunas, onde os pontos estão inseridos.

5.3.1 Superfícies definidas pelo produto tensorial de polinômios de Bernstein

A técnica do produto tensorial constrói superfícies por meio da multiplicação de várias curvas. Dadas duas curvas de Bézier, o método do produto tensorial constrói uma superfície por meio da multiplicação das funções de base da primeira curva com as funções de base da segunda e usa os resultados como funções de base para um conjunto bidimensional de pontos de controle. Superfícies geradas deste modo são denominadas *superfícies do produto tensorial*. Assim, as superfícies de Bézier são superfícies do produto tensorial (SHENE, 2003).

Um produto tensorial de polinômios de Bernstein, que envolve o mapeamento de $[0, 1]^m$ para $\mathbb{R}^n$, pode ser definido como uma soma ponderada dos $(w_1 + 1)(w_2 + 1) \dots (w_m + 1)$ pontos de controle $\mathbf{P}_{\vec{i}} = (P_{i_1}, \dots, P_{i_m})^T$, $\vec{i} = (i_1, \dots, i_m)^T$:

$$\sum_{i_1=0}^{w_1} \sum_{i_2=0}^{w_2} \dots \sum_{i_m=0}^{w_m} \mathbf{P}_{\vec{i}} B_{i_1}^{w_1}(t_1) B_{i_2}^{w_2}(t_2) \dots B_{i_m}^{w_m}(t_m) \quad (5.10)$$

onde os pesos B_i^w são os polinômios de Bernstein (RAIDL; KODYDEK, 1998).

Os produtos tensoriais de polinômios de Bernstein mais estudados são as curvas de Bézier ($[0, 1] \rightarrow \mathbb{R}^2$) e as superfícies de Bézier ($[0, 1]^2 \rightarrow \mathbb{R}^3$). Para estes casos, já foram derivados muitos critérios de qualidade, como a envoltória convexa e a variação decrescente

(*variation diminishing*). Juntos, esses critérios indicam que tais curvas, superfícies ou funções de transformação são muito suaves e não oscilam entre os pontos de controle (FARIN, 2002; RAIDL; WURM, 1998).

5.3.2 Parâmetros para superfícies

Uma superfície de Bézier, dada pela equação (5.9) anterior, precisa de $(w_1 + 1)(w_2 + 1)$ pontos de controle. Isto requer dois conjuntos de parâmetros para a aproximação dessa superfície.

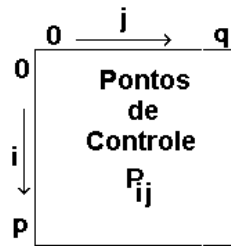


FIGURA 5.2 – Disposição dos pontos de controle em uma grade.

Suponha-se que os pontos de dados $\mathbf{D}_{ij}$ estejam arranjados em uma grade bidimensional de $m + 1$ linhas e $n + 1$ colunas, como na Figura 5.3, com $0 \leq i \leq m$ e $0 \leq j \leq n$. Assim, serão necessários $m + 1$ parâmetros $t_0^1, t_1^1, \dots, t_m^1$ na direção de t_1 , ou seja, um para cada linha de pontos de dados, e $n + 1$ parâmetros $t_0^2, \dots, t_n^2$ na direção de t_2 (um para cada coluna de pontos de dados) para que o ponto (t_c^1, t_d^2) , no domínio da superfície, corresponda ao ponto $B^{w_1, w_2}(t_c^1, t_d^2)$ da superfície, que, por sua vez, corresponde ao ponto de dado $\mathbf{D}_{cd}$. Colocando isto na forma de uma equação, o ponto sobre a superfície avaliado em (t_c^1, t_d^2) e mostrado abaixo

$$B^{w_1, w_2}(t_c^1, t_d^2) = \sum_{i_1=0}^{w_1} \sum_{i_2=0}^{w_2} \mathbf{P}_{i_1, i_2} B_{i_1}^{w_1}(t_c^1) B_{i_2}^{w_2}(t_d^2)$$

corresponde ao ponto de dado $\mathbf{D}_{cd}$, em que t_c^1 e t_d^2 são os parâmetros nas direções t_1 e t_2 , respectivamente.

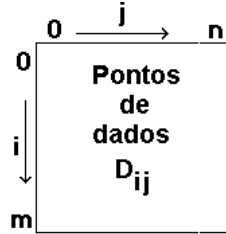


FIGURA 5.3 – Disposição dos pontos de dados em uma grade.

Nessa equação, a direção de t_1 corresponde ao índice i_1 em $b_{i_1}^{w_1}$ e $\mathbf{P}_{i_1, i_2}$. Desde que i varia de 0 até m , $B_0^p(u), B_1^p(u), \dots, B_m^p(u)$ são coeficientes das colunas dos pontos de controle. Logo, na direção t_1 , precisa-se de $m + 1$ parâmetros, e usando-se um método de cálculos de parâmetros que receba o grau p e os pontos de dados na coluna j , pode-se calcular os $m + 1$ parâmetros $u_{0,j}, u_{1,j}, \dots, u_{m,j}$. Isto é mostrado na Figura 5.3. Os parâmetros desejados $t_0^1, t_1^1, \dots, t_m^1$ são, simplesmente, a média de cada linha. Mais precisamente, o parâmetro t_i^1 é a média dos parâmetros da linha i , $t_i^1 = u_{i,0} + u_{i,1} + \dots + u_{i,n}/(n + 1)$.

O cálculo dos parâmetros na direção t_2 é similar. Cada linha de pontos de dados possui $n + 1$ pontos e, assim, requer $n + 1$ parâmetros. Logo, para os pontos de dados na linha i , pode-se calcular $n + 1$ valores do parâmetro $v_{i,0}, v_{i,1}, \dots, v_{i,n}$. Desde que se tenha $m + 1$ linhas, esses valores podem ser organizados em uma matriz $(m + 1) \times (n + 1)$, e o parâmetro t_j^2 é simplesmente a média dos parâmetros na coluna j , $t_j^2 = (v_{0,j} + v_{1,j} + \dots + v_{m,j})/(m + 1)$. Deste modo, pode-se obter $n + 1$ parâmetros para a direção v .

Esse procedimento pode ser resumido como mostrado na Figura 5.4.

Entrada: $(m+1) \times (n+1)$ pontos de dados $\mathbf{D}_{i,j}$.

Saída: Os parâmetros na direção $t_1, t_0^1, \dots, t_m^1$ e os parâmetros na direção $t_2, t_0^2, \dots, t_n^2$

Algoritmo: /* calcula $t_0^1, \dots, t_m^1$ */

para j **de** 0 **até** n **faça** /* coluna j */

 Calcula pelo método das cordas um conjunto de $m+1$ parâmetros $u_{0,j}, u_{1,j}, \dots, u_{m,j}$,

para i **de** 0 **até** m **faça**

$t_i^1 \leftarrow \frac{u_{i,0} + u_{i,1} + \dots + u_{i,n}}{n+1}$

 /* os parâmetros para a direção t_1 estão disponíveis */

 /* calcula $t_0^2, \dots, t_n^2$ */

para i **de** 0 **até** m **faça** /* linha j */

 Utilize o método das cordas para calcular um conjunto de $n+1$ parâmetros $v_{i,0}, v_{i,1}, \dots, v_{i,n}$

para j **de** 0 **até** n **faça**

$t_j^2 \leftarrow \frac{v_{0,j} + v_{1,j} + \dots + v_{m,j}}{m+1}$

 /* os parâmetros para a direção t_2 estão disponíveis */

FIGURA 5.4 – Algoritmo para o cálculo dos parâmetros de uma superfície de Bézier.

5.3.3 Aproximação global de superfícies

O número de pontos de dados K é muito maior que o número de pontos de controle, ou seja,

$$K = (m+1) \times (n+1) \gg (p+1) \times (q+1).$$

Cada ponto D_{cd} está associado a um par de parâmetros (t_c^1, t_d^2) . Quando se aplica a equação da superfície a este par de parâmetros, obtém-se o ponto de dado:

$$S(t_c^1, t_d^2) \approx D_{cd}$$

Ao se expandir a equação acima, usando a equação 5.9, obtém-se

$$S(t_c^1, t_d^2) = \sum_{i=0}^p \sum_{j=0}^q \mathbf{P}_{ij} B_i^p(t_c^1) B_j^q(t_d^2) \approx D_{cd} \quad (5.11)$$

Repetindo-se a operação para cada um dos pontos de dados, gera-se um conjunto de K equações, cujas incógnitas são os pontos de controle $\mathbf{P}_{ij}$.

$$\begin{aligned} \sum_{i=0}^p \sum_{j=0}^q \mathbf{P}_{ij} B_i^p(t_0^1) B_j^q(t_0^2) &\approx D_{00} \\ \sum_{i=0}^p \sum_{j=1}^q \mathbf{P}_{ij} B_i^p(t_0^1) B_j^q(t_1^2) &\approx D_{01} \\ &\vdots \\ \sum_{i=0}^p \sum_{j=n}^q \mathbf{P}_{ij} B_i^p(t_0^1) B_j^q(t_n^2) &\approx D_{0n} \\ &\vdots \\ \sum_{i=m}^p \sum_{j=0}^q \mathbf{P}_{ij} B_i^p(t_m^1) B_j^q(t_0^2) &\approx D_{m0} \\ &\vdots \\ \sum_{i=m}^p \sum_{j=n}^q \mathbf{P}_{ij} B_i^p(t_m^1) B_j^q(t_n^2) &\approx D_{mn} \end{aligned} \quad (5.12)$$

Colocando-se todo este equacionamento na forma matricial, tem-se:

$$\mathbf{B}(u) \mathbf{P} \mathbf{B}^T(v) \approx \mathbf{D} \quad (5.13)$$

em que

$$\begin{aligned}
 \mathbf{B}(u) &= \begin{pmatrix} B_0^p(t_0^1) & B_1^p(t_0^1) & \cdots & B_p^p(t_0^1) \\ B_0^p(t_1^1) & B_1^p(t_1^1) & \cdots & B_p^p(t_1^1) \\ \vdots & \vdots & \cdots & \vdots \\ B_0^p(t_m^1) & B_1^p(t_m^1) & \cdots & B_p^p(t_m^1) \end{pmatrix}_{(m+1) \times (p+1)} \\
 \mathbf{P} &= \begin{pmatrix} P_{00} & P_{01} & \cdots & P_{0q} \\ P_{10} & P_{11} & \cdots & P_{1q} \\ \vdots & \vdots & \cdots & \vdots \\ P_{p0} & P_{p1} & \cdots & P_{pq} \end{pmatrix}_{(p+1) \times (q+1)} \\
 \mathbf{B}(v) &= \begin{pmatrix} B_0^q(t_0^2) & B_1^q(t_0^2) & \cdots & B_q^q(t_0^2) \\ B_0^q(t_1^2) & B_1^q(t_1^2) & \cdots & B_q^q(t_1^2) \\ \vdots & \vdots & \cdots & \vdots \\ B_0^q(t_n^2) & B_1^q(t_n^2) & \cdots & B_q^q(t_n^2) \end{pmatrix}_{(n+1) \times (q+1)} \\
 \mathbf{D} &= \text{Matriz de pontos de dados}
 \end{aligned}$$

Na equação (5.9), a superfície $S(u, v)$ é uma função vetorial de dois valores paramétricos u e v . Uma descrição matemática desse relacionamento é mostrada abaixo na equação (5.14).

$$S(u, v) = \left\{ \begin{aligned} S_x(u, v) &= \sum_{i=0}^p \sum_{j=0}^q P_{x_{ij}} B_i^p(u) B_j^q(v) \\ S_y(u, v) &= \sum_{i=0}^p \sum_{j=0}^q P_{y_{ij}} B_i^p(u) B_j^q(v) \\ S_z(u, v) &= \sum_{i=0}^p \sum_{j=0}^q P_{z_{ij}} B_i^p(u) B_j^q(v) \end{aligned} \right\} \quad (5.14)$$

em que x , y e z são as coordenadas no espaço do modelo. $P_{x_{ij}}$, $P_{y_{ij}}$ e $P_{z_{ij}}$ são as co-

ordenadas do ponto de controle $\mathbf{P}_{ij}$. Desde que se supõe que o número K de pontos de dados é maior que os graus da superfície, este sistema é superdeterminado. Para resolvê-lo, necessita-se usar um método de otimização. Usa-se também o método dos mínimos quadrados de Gauss.

Definindo-se uma função

$$\chi^2 = \sum_{c=0}^m \sum_{d=0}^n \|D_{cd} - S(t_c^1, t_d^2)\|^2 \quad (5.15)$$

Pode-se minimizar a equação 5.15, fazendo com que as derivadas parciais de χ^2 em relação a todos os $(p+1) \times (q+1)$ pontos de controle $\mathbf{P}_{ij}$ sejam nulas, ou seja:

$$\frac{\partial \chi^2}{\partial \mathbf{P}_{ij}} = 0$$

O conjunto de equações resultantes forma um sistema linear em P_{ij} , embora os polinômios de Bernstein sejam não lineares.

6 Modelagem da deriva zonal do plasma na região F

A deriva do plasma é um parâmetro ionosférico importante. As derivas eletrodinâmicas da ionosfera na região F , em baixas e médias latitudes, são influenciadas, principalmente, pelos campos elétricos de dínamo das regiões E e F , mas, em tempos geomagneticamente ativos, elas podem ser afetadas fortemente pelos campos elétricos do dínamo perturbado e pelos campos elétricos de penetração direta da magnetosfera. As complexas interações entre estes processos causam uma forte variabilidade na deriva do plasma com uma maior extensão temporal. A compreensão da morfologia e das fontes dessa variabilidade é essencial para o melhoramento das previsões ionosféricas e para a verificação das previsões dos modelos de convecção e ionosféricos globais (FEJER *et al.*, 2005).

Medições do radar de espalhamento incoerente das derivas verticais do plasma equatorial na região F são realizadas rotineiramente, desde 1968, no Rádio-Observatório de Jicamarca no Peru. Essas observações foram usadas em numerosos casos e estudos climatológicos e também no desenvolvimento de modelos empíricos da deriva vertical equatorial, para condições calmas e perturbadas (FEJER; SCHERLIESS, 1997; SCHERLIESS; FEJER, 1999). Medições simultâneas das derivas zonal e vertical do plasma na região F ,

têm sido realizadas em Jicamarca desde 1970. Entretanto as derivas verticais são menos freqüentes, pois a medição simultânea desses dois componentes da velocidade pode gerar grandes erros nas derivas verticais. Como um resultado, a base de dados de derivas zonais em Jicamarca é muito menor que aquela das derivas verticais.

Neste trabalho, usaram-se as observações da deriva zonal em Jicamarca para modelar essas derivas como uma função da estação do ano, ciclo solar e atividade magnética.

6.1 Medições e dados

O procedimento básico para a medição das derivas na região F , em Jicamarca, foram descritas por Woodman (WOODMAN, 1972). Em resumo, uma grande antena de 50 MHz é aberta em dois feixes perpendiculares ao campo geomagnético e apontada 2.5° para leste, e 4.3° , para oeste da vertical, dando uma abertura líquida em torno de 6.8° . Essas derivas da linha de visada são combinadas para produzir as componentes de deriva zonal e vertical. As medições foram feitas em torno de 45 a 900 km com uma resolução de, aproximadamente, 15 km e um tempo de integração em torno de 5 minutos. Os dados usados neste estudo correspondem às médias da velocidade de deriva próximas ao pico da região F (tipicamente entre 300 a 500 km), onde eles não mudam muito com a altitude, e as relações sinal-ruído são maiores. Medições confiáveis não são possíveis durante períodos em que ocorrem os traços F espalhados (*spread-F*) em um grande intervalo de altitudes.

Foram usados 206 dias de medições feitas entre 1970 e 1992, e de 86 dias entre 1994 e 2003. Não houve medições durante 1979, 1982, 1983 e 1993. O número de observações noturnas é muito menor que o das diurnas, devido à ocorrência freqüente do fenômeno F-espalhado em todo o intervalo altitudinal medido. Para um tempo de integração de

5 minutos, a acurácia típica das medições da deriva zonal diurna ficou em torno de 15 m/s, antes de 1994, e em torno de 5 m/s, após 1994, quando técnicas de análise mais poderosas foram implementadas (KUDEKI; BHATTACHARYYA; WOODMAN, 1999). As medições noturnas apresentam erros maiores, chegando a 30 m/s particularmente durante os períodos de mínimos solares da madrugada, decorrentes da redução da relação sinal-ruído.

Os dados usados neste trabalho foram agrupados em intervalos de meia hora, centrados entre 15 e 45 minutos de cada hora, e em grupos sazonais, representando o solstício de junho (maio a agosto), solstício de dezembro (novembro a fevereiro) e o equinócio (março e abril, setembro e outubro). Definiu-se a atividade solar como uma função do índice de fluxo solar $F_{10,7}$ cm. Posto que os índices AE não estavam disponíveis para todo o período das medições, a atividade geomagnética foi definida em termos do índice K_p , com condições de tempo calmo, definidas por um índice K_p médio menor ou igual a 3.0, nas últimas 9 horas.

6.2 Formulação matemática

Pode-se enunciar o problema da aproximação de nuvens de pontos de dados da seguinte forma:

Definição 1 *Suponha-se que sejam dados $p + 1$ pontos $\mathbf{P}_i = (h_i, v_i)$, com $i = 0, \dots, p$, resultantes de medições. Cada ponto possui um erro associado dado por σ_i e parametrizado por t_i . Deseja-se encontrar um polinômio $\mathbf{x}(t)$ tal que as distâncias $\|\mathbf{P}_i - \mathbf{x}(t_i)\|$ sejam mínimas.*

A forma geral deste tipo de modelo é

$$\mathbf{x}(t) = \sum_{k=0}^n \mathbf{a}_k B_k^n(t)$$

em que $B_0^n, \dots, B_n^n$ são funções da base de polinômios de Bernstein de grau n . A curva desejada é, então, uma curva de Bézier.

Como visto no Capítulo 5, pode-se resolver o problema posto usando-se o método dos mínimos quadrados. O primeiro passo consiste na definição da função

$$\chi^2 = \sum_{i=0}^p \left(\left(h_i, \frac{v_i}{\sigma_i} \right) - \sum_{k=0}^n \left(a_{h_k}, \frac{a_{v_k}}{\sigma_i} \right) B_k^n(t_i) \right)^2 \quad (6.1)$$

chamada de *qui-quadrado*.

Os parâmetros $\mathbf{a}_i$ que melhor ajustam o modelo ao conjunto de dados são aqueles que minimizam χ^2 .

O método dos mínimos quadrados, aqui esboçado, assume os seguintes fatos sobre os dados (PRESS *et al.*, 1993):

1. As variáveis independentes (neste caso, h_i) são *conhecidos com alta precisão*, e os erros ocorrem somente nos dados medidos, ou seja, em *única variável dependente* (neste caso, v_i), dados pelos desvios-padrão σ_i .
2. Os erros de medições seguem uma distribuição Gaussiana em torno do valor verdadeiro da variável observada.

Pode-se encontrar esse mínimo usando as equações normais. Antes de usá-las, precisa-se de mais algumas definições.

Seja $\mathbf{A}$ uma matriz cujos $(p+1) \times (n+1)$ elementos são constituídos das $n+1$ funções da base de Bernstein, avaliadas nos $p+1$ parâmetros t_i com seus respectivos $p+1$ erros σ_i , tal que:

$$A_{ij} = \frac{B_j^n(t_i)}{\sigma_i}$$

A matriz $\mathbf{A}$ é chamada *matriz de projeto* do problema de ajuste. Esta matriz é mostrada, esquematicamente, na Figura 6.1.

$$\begin{pmatrix} \frac{B_0^n(t_0)}{\sigma_0} & \frac{B_1^n(t_0)}{\sigma_0} & \dots & \frac{B_n^n(t_0)}{\sigma_0} \\ \frac{B_0^n(t_1)}{\sigma_1} & \frac{B_1^n(t_1)}{\sigma_1} & \dots & \frac{B_n^n(t_1)}{\sigma_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{B_0^n(t_p)}{\sigma_p} & \frac{B_1^n(t_p)}{\sigma_p} & \dots & \frac{B_n^n(t_p)}{\sigma_p} \end{pmatrix}$$

FIGURA 6.1 – Matriz de projeto para o ajuste por mínimos quadrados.

Defina-se um vetor $\mathbf{B}$ de comprimento $p+1$ por

$$\mathbf{B}_i = \frac{\mathbf{P}_i}{\sigma_i} \quad (6.2)$$

Denote-se um vetor $\mathbf{a}$ com seus $n+1$ elementos dados pelos parâmetros $\mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_n$ a serem ajustados.

O mínimo de (6.1) ocorre quando a derivada de χ^2 com respeito a todos os $n+1$ parâmetros a_k é nula. Assim, esta condição produz $n+1$ equações

$$0 = \sum_{i=0}^p \frac{1}{\sigma_i^2} \left(\mathbf{P}_i - \sum_{j=0}^n \mathbf{a}_j B_j^n(t_i) \right) B_k^n(t_i) \quad k = 0, 1, \dots, n \quad (6.3)$$

Invertendo-se a ordem dos somatórios, pode-se reescrever a equação anterior na forma matricial

$$\sum_{j=0}^n \alpha_{kj} \mathbf{a}_j = \beta_k \quad (6.4)$$

em que

$$\alpha_{kj} = \sum_{i=0}^p \frac{B_j^n(t_i) B_k^n(t_i)}{\sigma_i^2}, \quad \text{ou, equivalentemente,} \quad \alpha = \mathbf{A}^T \cdot \mathbf{A} \quad (6.5)$$

é uma matriz $(n+1) \times (n+1)$, e

$$\beta_k = \sum_{i=0}^p \frac{\mathbf{P}_i B_k^n(t_i)}{\sigma_i^2}, \quad \text{ou, equivalentemente,} \quad \beta = \mathbf{A}^T \cdot \mathbf{B} \quad (6.6)$$

é um vetor com $n+1$ elementos.

As equações (6.3) ou (6.4) são chamadas *equações normais* do problema dos mínimos quadrados. Elas podem ser resolvidas para o vetor de parâmetros $\mathbf{a}$ por meio da decomposição *LU* e substituição regressiva (PRESS *et al.*, 1993). Em forma matricial, as equações normais podem ser reescritas como

$$\alpha \cdot \mathbf{a} = \beta \quad \text{ou como} \quad (\mathbf{A}^T \mathbf{A}) \cdot \mathbf{a} = \mathbf{A}^T \cdot \mathbf{B} \quad (6.7)$$

A matriz inversa $C_{jk} \equiv \alpha_{jk}^{-1}$ é relacionada aos desvios-padrões dos parâmetros estimados $\mathbf{a}$ da seguinte forma:

$$\sigma^2(\mathbf{a}_j) = C_{jj} \quad (6.8)$$

Em outras palavras, os elementos da diagonal principal de $\mathbf{C}$ são as variâncias dos parâmetros de ajuste $\mathbf{a}$. Os outros elementos C_{jk} , fora desta diagonal, são as covariâncias entre $\mathbf{a}_j$ e $\mathbf{a}_k$.

O método descrito acima minimiza a função χ^2 , sem levar em conta a forma da curva resultante. No Capítulo 5, foi visto que o método de suavização pelas diferenças segundas pode resolver esse problema. A equação (5.5) define uma matriz $\mathbf{S}$ em que as diferenças segundas $\Delta^2 \mathbf{a}_i = \mathbf{a}_{i+2} - 2\mathbf{a}_{i+1} + \mathbf{a}_i$ dos pontos de controle são postas. A resolução do sistema resultante, conforme a equação 5.8, produz

$$\mathbf{a} = (\mathbf{N}^T \cdot \mathbf{N})^{-1} \cdot \mathbf{N}^T \cdot \mathbf{Q}$$

que determina os $n + 1$ coeficientes.

6.2.1 Derivação dos modelos

Os modelos de tempos calmo e perturbado foram desenvolvidos em dois passos. No caso do modelo de tempo calmo, primeiro, usou-se, para cada estação do ano e hora local, um polinômio de Bernstein de segundo grau com $\sigma = 1$ e $\omega = 0.4$, modelando, assim, a dependência das derivas em relação ao índice de fluxo solar 10.7 cm. Este primeiro ajuste forneceu os desvios-padrões (σ_n). Em seguida, esse modelo de fluxo solar foi usado para calcular as derivas em 48 intervalos de tempo igualmente espaçados e com oito valores do índice de fluxo solar incrementados em 25 unidades de 66 a 241. Os valores de deriva, para um dado índice de fluxo solar, foram determinados pela interpolação linear entre os dois valores calculados mais próximos. No passo final, esses dados foram modelados com polinômios de Bernstein de grau 11 com $\sigma = \sigma_m$ e $\omega = 0.0005$. Assim, no modelo calmo,

a matriz $\mathbf{N}$ possui 58×12 elementos ($m=48$ e $n=11$).

No desenvolvimento dos modelos de deriva perturbados, iniciou-se pela consideração dos efeitos do fluxo solar. Assim, as derivas residuais (ou seja, os valores medidos menos os valores médios correspondentes à estação e à dependência do ciclo solar) foram colocadas em duas janelas sobrepostas do índice $F_{10,7}$, a saber, $F_{10,7} \leq 140$ e $F_{10,7} > 120$. As derivas perturbadas do equinócio e do solstício de dezembro foram combinadas para melhorar as estatísticas. Não foram examinados os efeitos da atividade magnética durante o solstício de junho, pois eles parecem ser negligenciáveis. As derivas de perturbação devidas ao fluxo solar baixo ou alto foram modeladas, em grupos de meia hora, como funções lineares dos índices k_p ponderados pelo tempo. As variações devidas à hora local das derivas perturbadas foram também modeladas com os polinômios de Bernstein de grau 11 com $\omega = 0.0005$.

Ao todo, os modelos de deriva de tempos calmo e perturbado, que estimam valores médios e suas variabilidades, possuem 648 e 336 coeficientes, respectivamente. As entradas do modelo são: o dia do ano (no calendário Juliano), o índice $F_{10,7}$ em e o índice K_p das últimas 24 horas. A derivas do modelo, para um determinado dia, são dadas pela soma dos componentes de deriva em tempo calmo e com perturbação.

7 Resultados e discussão

Este capítulo descreve os resultados obtidos nas várias tentativas de obtenção da melhor curva de aproximação para a velocidade de deriva zonal. A apresentação desses resultados segue a mesma ordem cronológica pela qual os vários métodos de tratamento de ruído foram estudados e implementados no decorrer desta pesquisa.

7.1 Preparação dos dados

No início desta pesquisa, os dados eram provenientes do magnetômetro de Jicamarca-Piura, instalado na Rádio-Observatório de Jicamarca, no Peru. Entretanto, estes dados foram logo substituídos pelos dados resultantes de medições simultâneas das derivas zonal e vertical do plasma na região F feitas pelo radar de espalhamento incoerente instalado no Rádio-Observatório de Jicamarca.

Para cada dia de medição, designado por `yyymmdd` para ano, mês e dia, existem quatro arquivos:

- um com as velocidades de deriva verticais, nomeado por `vvyymmdd.dat`. Um exemplo é mostrado na Figura 7.1.
- outro com os erros de medição da deriva vertical, nomeado por `vv_erryyymmdd.dat`. (ver

DAY:	12/12/2001	F10.7:	229.4		
LT/H	0.000	0.083	0.167	14.417	23.917
45.	1.***	1.***	1.***	1.***	1.***
60.	-12.704	-12.177	-11.086	4.567	-0.268
75.	-6.855	-10.191	-10.772	6.752	1.901
90.	-12.526	-11.721	-11.204	6.495	1.468
105.	-10.640	-10.640	-11.105	10.103	1.***
120.	-15.663	-14.475	-13.538	13.186	1.***
135.	-11.042	-14.378	-11.007 [...]	10.983 [...]	1.***
150.	-15.967	-14.652	-11.089	14.408	1.***
165.	1.***	1.***	-4.301	8.279	1.***
180.	2.385	1.***	-14.016	6.697	-2.357
195.	1.***	1.***	-17.491	10.487	1.***
210.	1.***	1.***	1.***	10.042	1.***
225.	1.***	1.***	-11.947	10.185	1.***

FIGURA 7.1 – Fragmento de um arquivo contendo dados de velocidade de deriva vertical medidos pelo radar de espalhamento incoerente para o dia 12/12/2001. As velocidades são expressas em m/s.
Fonte: Rádio-Observatório de Jicamarca.

Figura 7.2).

- um terceiro com as velocidades de deriva zonal, nomeado por `zvyymmdd.dat` (ver Figura 7.3).
- um quarto com os erros de medição da velocidade de deriva zonal, nomeado por `zv_errymmdd.dat`. Os arquivos são similares àquele mostrado na Figura 7.2.

Em cada arquivo, a primeira linha contém a data e o valor do índice de fluxo solar $F_{10,7}$ cm, medido em sfu^1 , ocorrido no dia em que os dados foram gravados. A linha seguinte contém a hora local em decimal, e a primeira coluna contém a altitude, em km, ambas para cada medição. Os outros elementos das matrizes são as velocidades, medidas em m/s, ou erros, também em m/s, medidos para cada hora local e altitude. A ausência de medições é representada por 1.***. Para esses dados, foram feitos tratamentos de

¹uma unidade de fluxo solar, cuja sigla em inglês é sfu, é igual a 10^{-22} Watts por metro quadrado por Hertz

DAY: 12/12/2001	F10.7: 229.4				
LT/H	0.000	0.083	0.167	14.417	23.917
45.	1.***	1.***	1.***	1.***	1.***
60.	0.062	0.068	0.070	0.076	0.108
75.	0.078	0.083	0.085	0.102	0.303
90.	0.070	0.083	0.073	0.073	0.119
105.	0.079	0.069	0.072	0.096	1.***
120.	0.082	0.081	0.082	0.165	1.***
135.	0.066	0.071	0.072 [...]	0.098 [...]	1.***
150.	0.067	0.076	0.076	0.265	1.***
165.	1.***	1.***	0.091	0.404	1.***
180.	2.122	1.***	0.342	0.419	0.492
195.	1.***	1.***	0.321	0.544	1.***
210.	1.***	1.***	1.***	0.491	1.***
225.	1.***	1.***	1.063	0.497	1.***

FIGURA 7.2 – Fragmento de um arquivo contendo erros de medições, em m/s, da deriva vertical feitas pelo radar de espalhamento incoerente do Rádio-Observatório de Jicamarca para o dia 12/12/2001.

erros.

DAY: 12/12/2001	F10.7: 229.4				
LT/H	0.000	0.083	0.167	14.417	23.917
45.	1.***	1.***	1.***	1.***	1.***
60.	91.430	-4.082	133.009	17.120	128.299
75.	45.214	38.361	128.350	-36.110	72.027
90.	95.114	-13.161	125.945	46.505	111.483
105.	124.407	40.542	130.891	-82.838	1.***
120.	83.287	132.606	186.849	-62.599	1.***
135.	129.425	126.466	138.195 [...]	-51.072 [...]	1.***
150.	92.606	136.267	136.578	-80.636	1.***
165.	1.***	1.***	24.687	-135.069	1.***
180.	145.378	1.***	42.423	-23.206	-57.028
195.	1.***	1.***	21.672	-103.603	1.***
210.	1.***	1.***	1.***	-58.950	1.***
225.	1.***	1.***	30.598	-48.783	1.***

FIGURA 7.3 – Fragmento de um arquivo contendo dados de velocidades de deriva zonal, em m/s, medidas pelo radar de espalhamento incoerente do Rádio-Observatório de Jicamarca para o dia 12/12/2001.

7.2 Filtragem com ondaletas

Uma vez realizada a limpeza e preparados os dados, começou-se a investigar como seria possível a construção de um sistema computacional capaz das seguintes tarefas:

1. Procurar dias calmos. Nesses dias, os valores do índice K_p devem ser menores ou iguais a 3.
2. Filtrar as perturbações indesejadas em cada perfil vertical da deriva zonal e obter a sua variação vertical.

Inicialmente, a ondaleta de Haar foi escolhida por se tratar de uma ondaleta simples e, portanto, de mais fácil interpretação. O resultado é mostrado na Figura 7.4.

Como intuito de filtrar as perturbações, as únicas saídas consideradas foram aquelas dos filtros passa-baixa, representados pela coluna esquerda da figura, já que os ruídos, normalmente, são de alta frequência.

Foram testadas outras formas de ondaletas com a finalidade de detectar os perfis de variação vertical da velocidade de deriva zonal. As transformadas de ondaletas biortogonais de Cohen-Daubechies-Feauveau (CDF) foram tentadas, e as curvas resultantes da aplicação da CDF(2,2) e da CDF(4,2) são mostradas nas figuras 7.5 e 7.6, respectivamente.

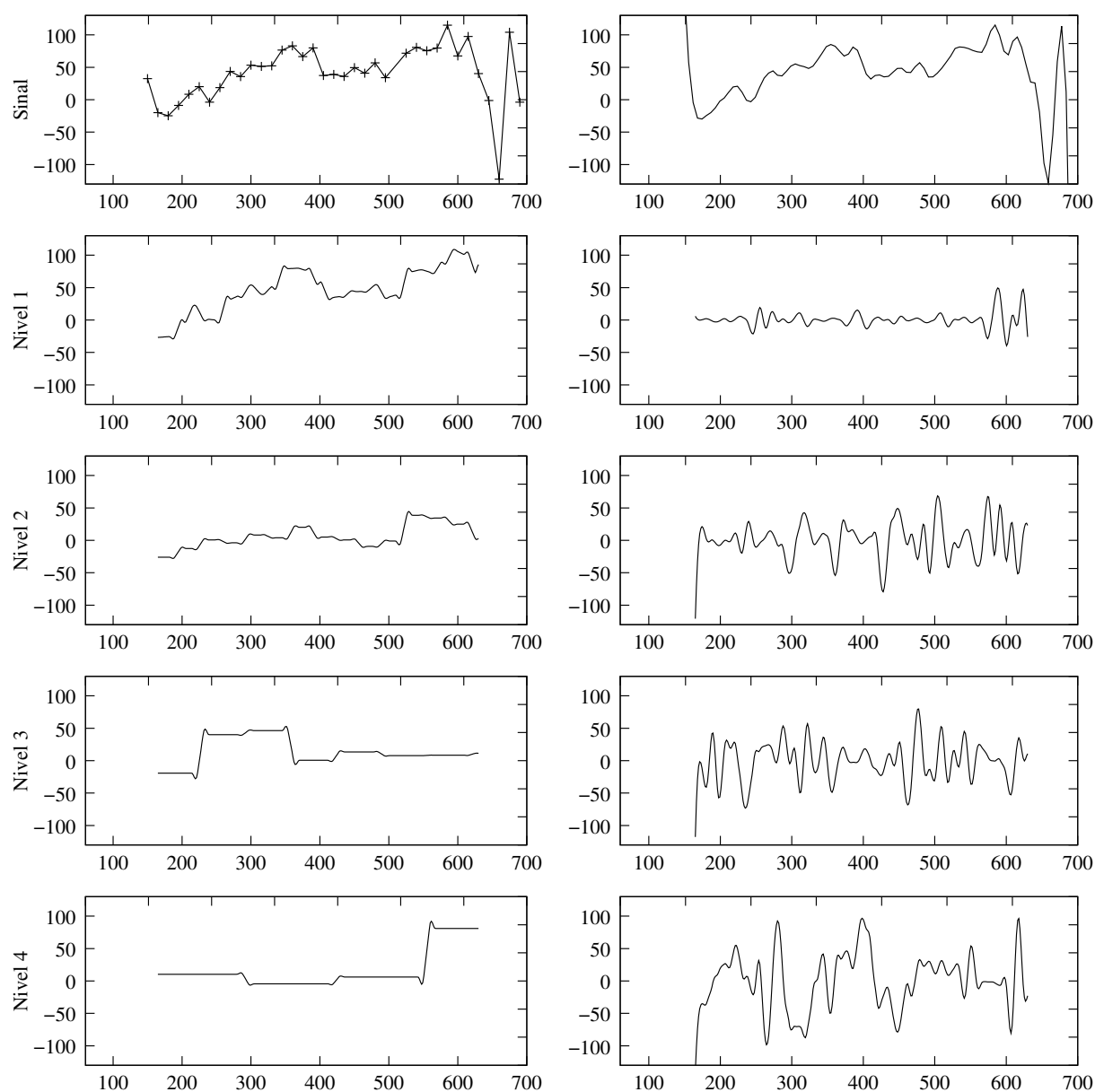


FIGURA 7.4 – Ondaleta de Haar aplicada à variação vertical da velocidade de deriva zonal. No eixo horizontal, está a altitude em km e, no eixo vertical, a velocidade de deriva zonal em m/s. O primeiro gráfico, no topo à esquerda, é a plotagem de uma parte das velocidades de deriva zonal do dia 12/12/2001, vinda dos equipamentos em Jicamarca. O outro gráfico, no topo à direita, é o resultado do uso de uma spline cúbica nesses mesmos dados. Todos os outros gráficos são resultados da aplicação da ondaleta de Haar, um para cada nível de decomposição. Os resultados dos filtros passa-baixa são mostrados à esquerda e aqueles dos filtros passa-alta, à direita.

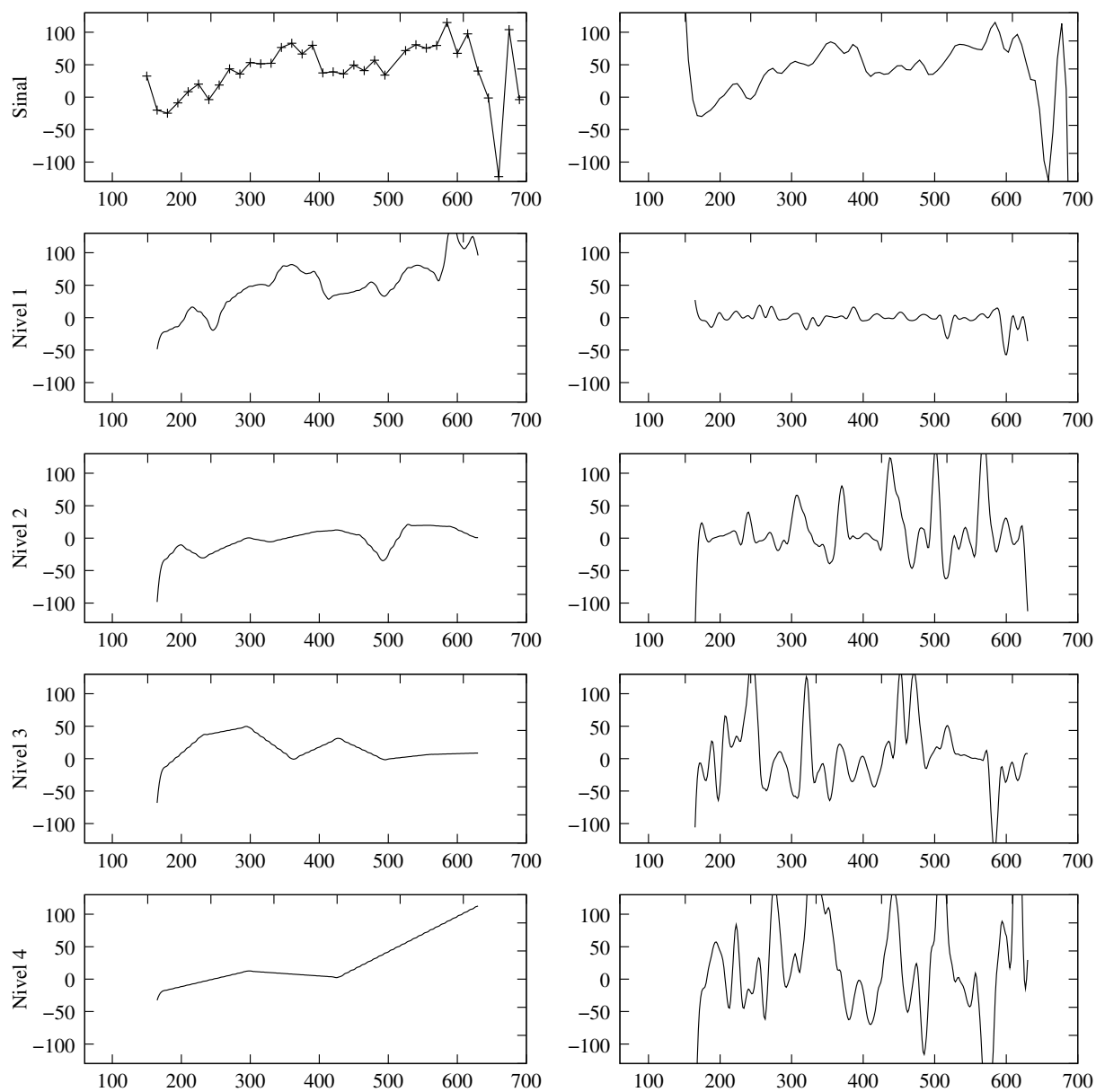


FIGURA 7.5 – Ondaleta CDF(2,2) aplicada à variação vertical da velocidade de deriva zonal. A interpretação dos gráficos é análoga àquela da Figura 7.4.

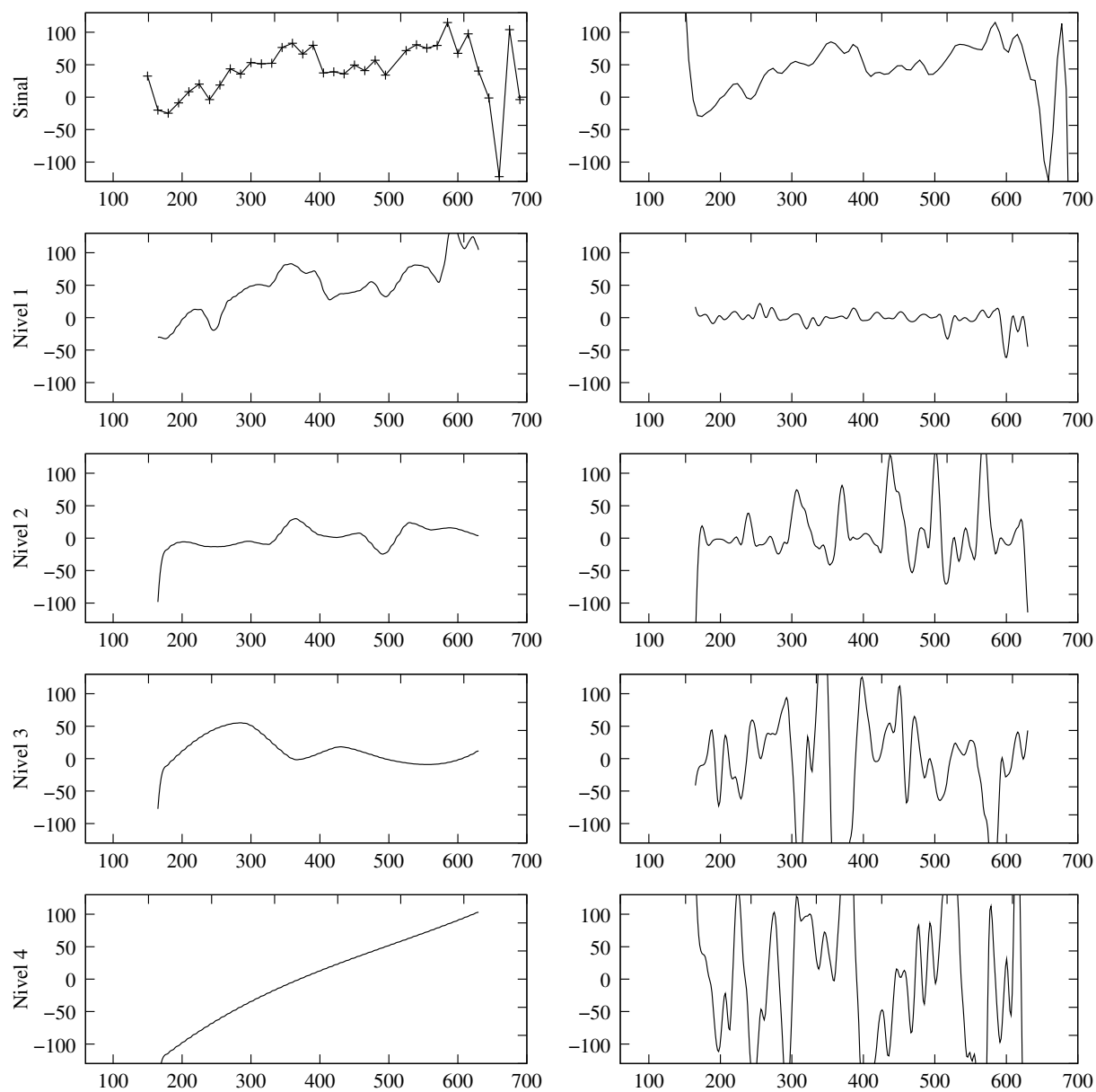


FIGURA 7.6 – Ondaleta CDF(4,2) aplicada à variação vertical da velocidade de deriva zonal. A interpretação dos gráficos é análoga àquela da Figura 7.4.

7.2.1 Aproximação com curvas de Bézier e mínimos quadrados

Nesta seção, serão mostrados os resultados da modelagem da variação vertical das velocidades de derivas zonais para épocas geomagneticamente calmas.

7.2.1.1 Modelo de derivas zonais em épocas geomagneticamente calmas

A Figura 7.7 compara as derivas zonais médias com o modelo desenvolvido, usando as curvas de Bézier para três níveis de fluxo solar, em três épocas geomagneticamente calmas (K_p médio em torno de 1.8 nas últimas nove horas). As curvas foram modeladas segundo a equação (6.1) do Capítulo 6 e repetida abaixo para facilidade de referência

$$\chi^2 = \sum_{i=0}^p \left(\left(h_i, \frac{v_i}{\sigma_i} \right) - \sum_{k=0}^n \left(a_{h_k}, \frac{a_{v_k}}{\sigma_i} \right) B_k^n(t_i) \right)^2 \quad (7.1)$$

os valores dos parâmetros deste modelo foram explicados na seção 6.2.1 do Capítulo 6.

As barras verticais, na Figura 7.7, denotam a variabilidade dos dados agrupados; normalmente, os erros de medição possuem valores muito menores. Essa figura mostra que os resultados do modelo empírico estão em boa concordância com os dados médios.

A Figura 7.8 mostra as derivas modeladas para épocas calmas para condições típicas de fluxo solar baixo e alto.

A Figura 7.9 mostra a boa concordância entre os desvios-padrões dos dados agrupados e aqueles calculados pelo modelo para três níveis de fluxo solar. Os cálculos são menos acurados na parte das altas horas da madrugada, em razão dos grandes erros e também do menor número de medições.

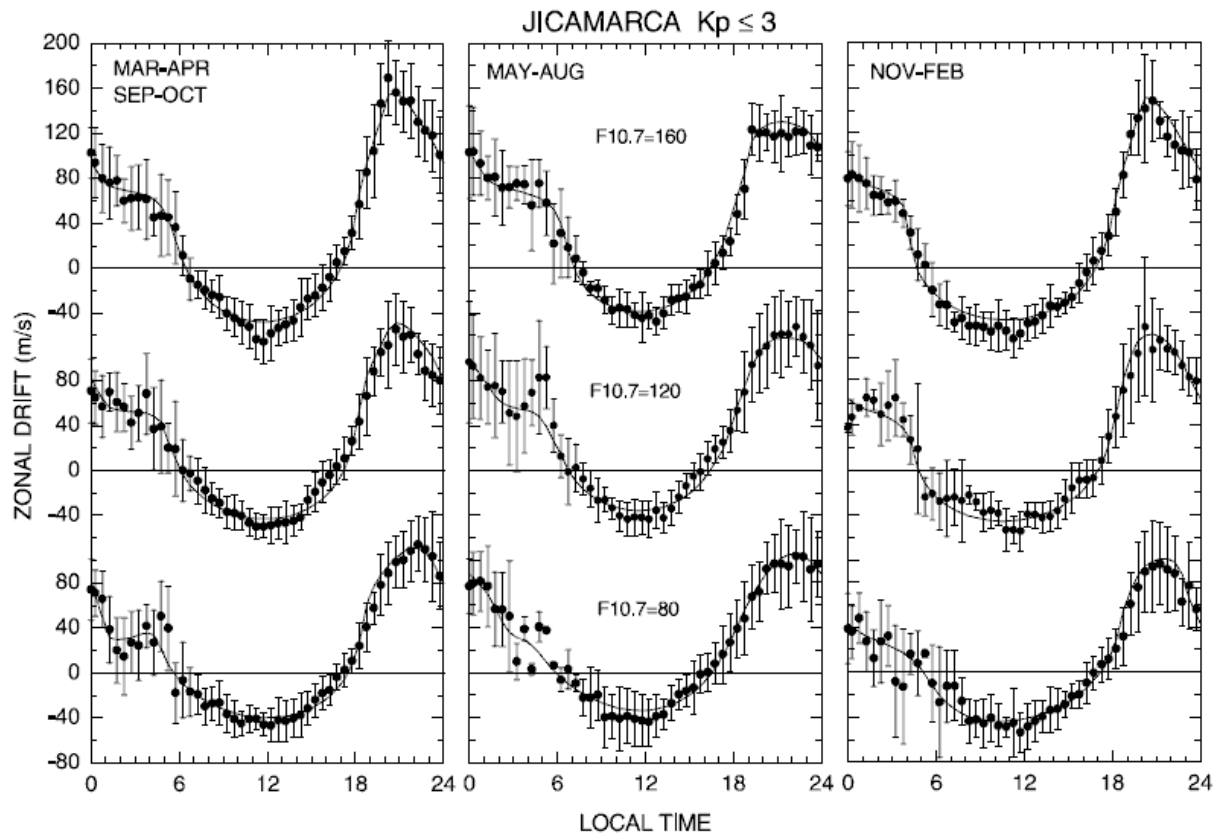


FIGURA 7.7 – Dependência sazonal e do ciclo solar das derivas zonais médias em tempo calmo da região F sobre Jicamarca. As barras denotam os desvios-padrões e as curvas sólidas representam os resultados do modelo.

Fonte: (FEJER *et al.*, 2005)

7.2.1.2 Modelo de derivas zonais em épocas geomagneticamente perturbadas

A Figura 7.10 mostra a hora local e as variações sazonais das derivas zonais médias em Jicamarca, sob condições de tempo calmo e de tempo perturbado, para fluxos solares baixo e moderadamente alto.

A Figura 7.11 ilustra a excelente concordância entre o modelo empírico de deriva zonal com perturbação e os correspondentes dados agrupados, juntamente com a dependência do ciclo solar dessas derivas perturbadas.

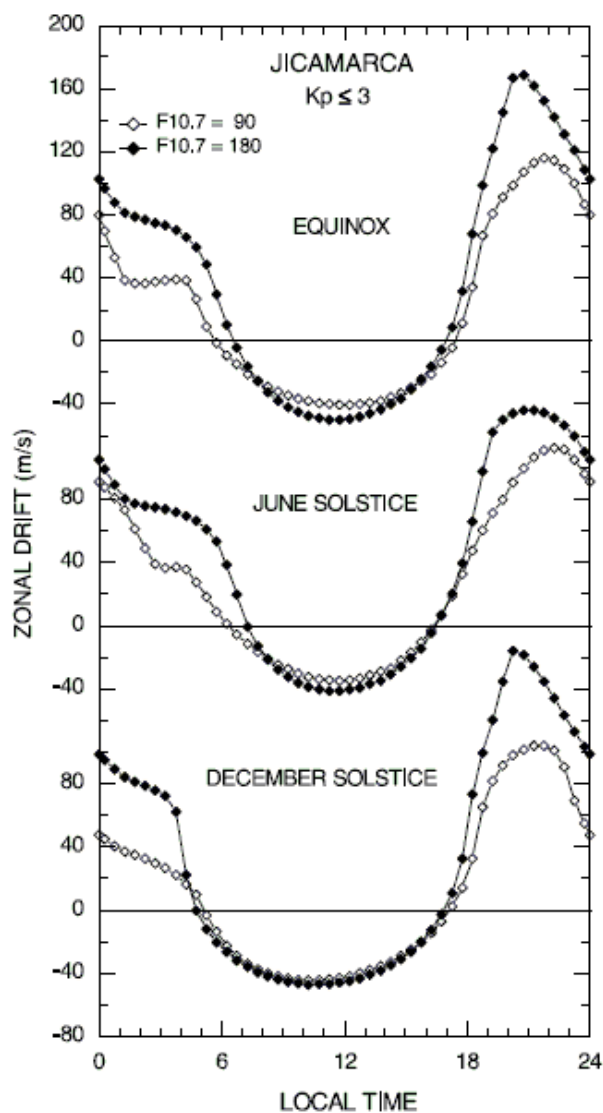


FIGURA 7.8 – Comparações das derivas zonais do modelo de tempo calmo em Jicamarca para condições de fluxos solares baixo e moderadamente alto.
Fonte: (FEJER *et al.*, 2005)

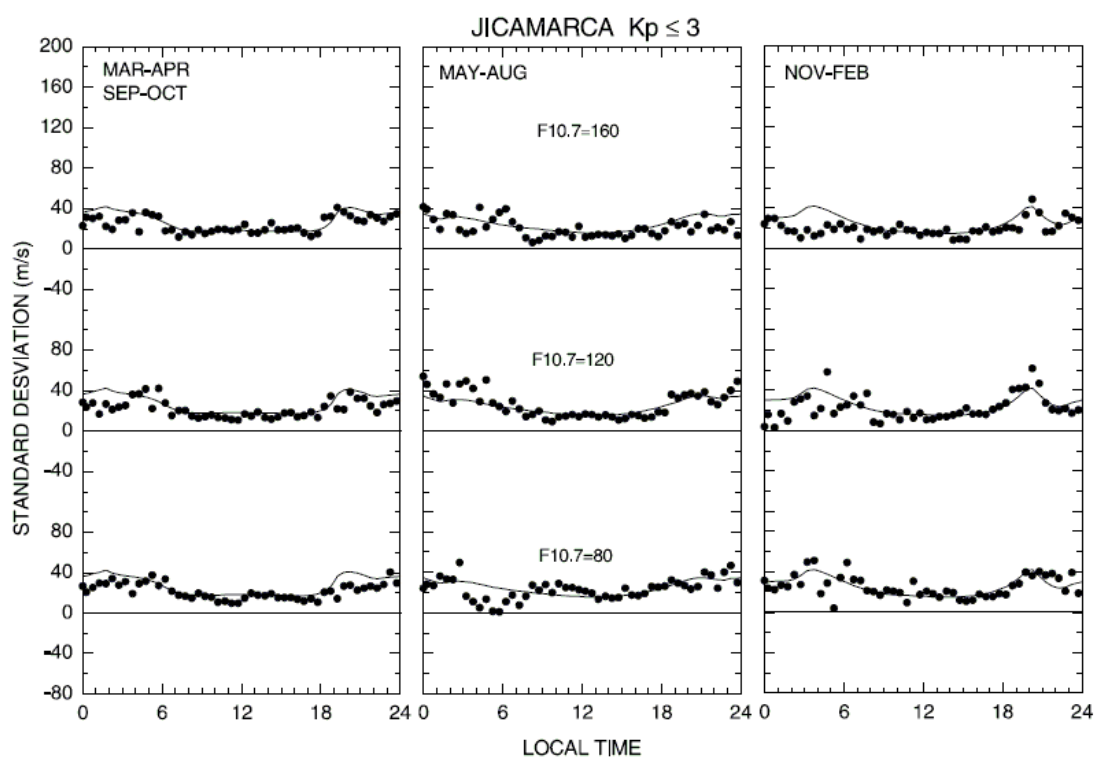


FIGURA 7.9 – Desvios-padrões da deriva zonal obtidos pelo agrupamento de dados e calculados usando o modelo empírico-analítico.

Fonte: (FEJER *et al.*, 2005)

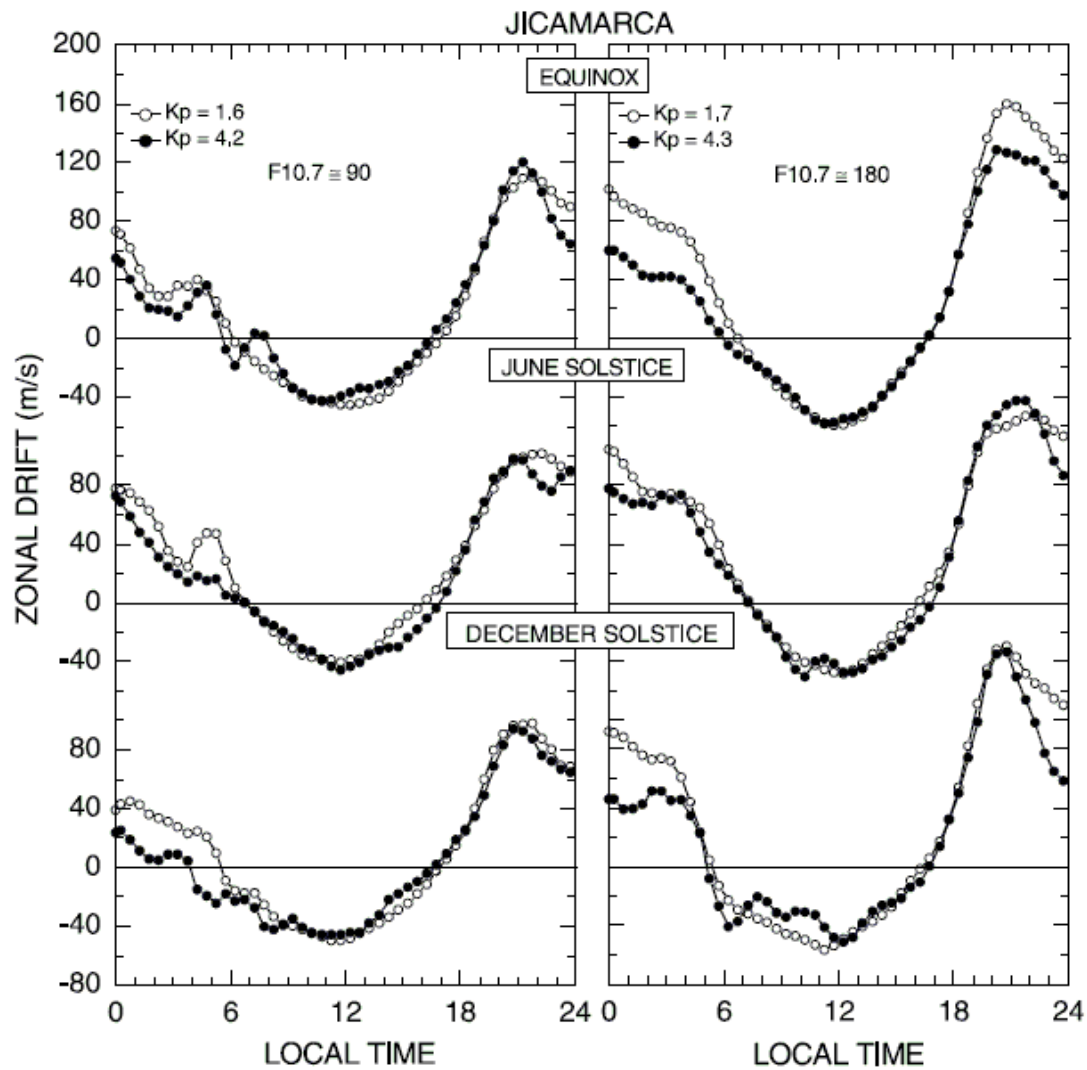


FIGURA 7.10 – Dependência do ciclo solar e da sazonalidade das derivas zonais médias em tempos calmo e perturbado sobre Jicamarca.

Fonte: (FEJER *et al.*, 2005)

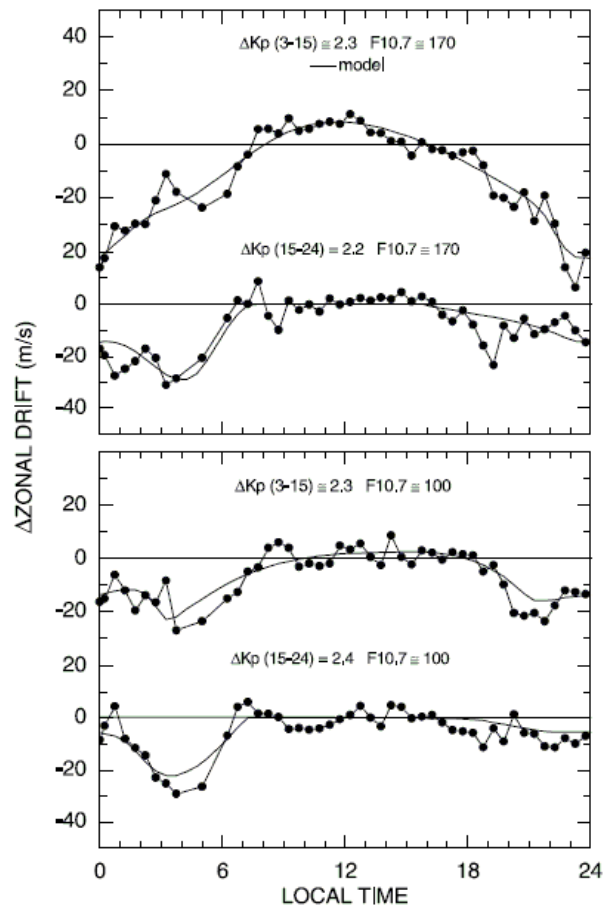


FIGURA 7.11 – Comparações entre as derivas perturbadas modeladas com aquelas agrupadas.

Fonte: (FEJER *et al.*, 2005)

7.3 Detalhes de implementação

Todos os modelos desta pesquisa foram implementados na linguagem *Objective Caml*²(LEROY *et al.*, 2004). As principais motivações para esta escolha foram (GRINGS, 2006):

1. Programação multiparadigma. Embora a OCaml, fundamentalmente, pertença à família das linguagens funcionais, ela também permite os estilos de programação imperativo e orientado a objetos, todos de maneira integrada no próprio projeto da linguagem;
2. Gerência automática de memória. Todas as alocações e desalocações de memória estão a cargo do próprio compilador, o que torna a programação menos propensa a erros.
3. Portabilidade. Os programas escritos nessa linguagem podem ser compilados, praticamente, sem alteração, em diferentes versões do Unix (incluindo Linux e MacOS X) e do Microsoft Windows.
4. Geração de código eficiente. OCaml possui um gerador de código nativo rápido e eficiente. A qualidade do código gerado é competitiva com a qualidade daqueles gerados por compiladores C e, algumas vezes, até melhor³.
5. A linguagem possui um interpretador. Além do compilador, OCaml também possui um interpretador, o que acelera o ciclo de desenvolvimento de programas.
6. Grande número de bibliotecas. OCaml possui várias bibliotecas construídas na própria linguagem para um grande número de aplicações, indo desde estruturas de

²Disponível em <http://caml.inria.fr>

³Para uma comparação entre diferentes implementações de várias linguagens de programação, consulte <http://shootout.alioth.debian.org>

dados até rotinas de cálculo numérico. Além disto, devido à facilidade de integração de códigos escritos em outras linguagens, o sistema permite que virtualmente todas as bibliotecas escritas nessas linguagens sejam facilmente incorporadas.

7. Comunidade ativa de usuários. Objective Caml possui uma grande e ativa comunidade de usuários espalhados pelo mundo. Assim, a resolução de eventuais problemas e dúvidas sobre algum ponto de implementação é grandemente facilitada.
8. Boa documentação. Os usuários e os implementadores da linguagem escrevem boa documentação, prontamente acessível pelo site da linguagem ou na Internet.

Um dos objetivos secundários desta pesquisa foi demonstrar a grande adaptabilidade e eficiência de linguagens declarativas, sobretudo, as linguagens funcionais, em problemas em que, normalmente, são usadas linguagens imperativas como C ou Fortran. Para alcançar este objetivo, todos os procedimentos numéricos foram implementados ou reimplementados em Ocaml.

Muitas rotinas para a manipulação matricial e resolução de sistemas lineares, como a decomposição LU, foram originalmente escritas em C ou Fortran (PRESS *et al.*, 1993). Resolveu-se, então, reimplementá-las em OCaml, e isto foi essencial para a manutenção e comunicação dos códigos entre os vários membros da equipe concentrados em Uberlândia e em Utah, nos Estados Unidos.

As implementações dos filtros de Kalman e dos filtros H_∞ foram portadas de Matlab (SIMON, 2001b; SIMON, 2001a) para Ocaml, com ganho considerável de eficiência.

As transformadas de ondaletas, implementadas segundo o esquema de *lifting*, foram, em parte, escritas diretamente em OCaml, a partir de descrições algorítmicas (SWEL-

DENS; SCHRÖDER, 1996) e, em parte, portadas do código da biblioteca LIFTPACK⁴, escrita em C.

Todo o programa envolvendo as curvas e superfícies de Bézier foi escrito diretamente a partir da modelagem matemática descrita nos capítulos 5 e 6. Este código está incluído no apêndice B.

⁴Disponível em <http://www.cs.dartmouth.edu/~sp/liftpack/>

8 Conclusão

As tentativas iniciais de usar o filtro de Kalman ou o filtro H_∞ , para encontrar um modelo preditivo para a ocorrência de variações verticais das velocidades de derivas zonais, não se revelaram promissoras. A razão para esse insucesso parece estar no fato de que ambos os filtros necessitam de um modelo físico completo do fenômeno sendo estudado. Nenhum modelo com tal exatidão é conhecido pelos geofísicos para as derivas iônicas. Devido a esta lacuna, a aplicação imediata dos filtros não pareceu ser viável.

A abordagem utilizando as ondaletas para filtrar as perturbações indesejadas para obter um modelo de deriva zonal, considerando as variações verticais e tempo geomagneticamente calmo, produziu resultados razoáveis. Embora a forma das curvas obtidas se aproximasse da intuição dos geofísicos e também dos dados experimentais, os resultados, em geral, ficaram abaixo do esperado. Tentaram-se muitas formas de ondaletas-mãe, mas em nenhum dos experimentos, conseguiu-se suavizar a curva resultante, eliminando as oscilações, principalmente aquelas causadas pelos traços F espalhados.

Não foi possível a utilização de uma mistura de ondaletas com filtro de Kalman, descrita no Capítulo 4, por razões similares às aquelas do insucesso do uso do filtro de Kalman sozinho: o não conhecimento de um modelo físico subjacente.

O uso das curvas de Bézier com mínimos quadrados produziu um melhor ajuste do

que todas as abordagens anteriores e possibilitou a construção do modelo empírico das derivas de plasma usado pela equipe de geofísicos (FEJER *et al.*, 2005).

O desenvolvimento de ferramentas numéricas para a construção de modelos empíricos voltados para a modelagem das derivas zonais do plasma ionosférico, baseado em polinômios de Bernstein, é a principal contribuição deste trabalho. Apesar de sua simplicidade e do número relativamente pequeno de parâmetros, essa técnica de modelagem não havia, ainda, sido usada em geofísica espacial, embora seja utilizada em outras áreas, tais como a modelagem geométrica auxiliada por computador. A junção da curva de Bézier, com o controle da suavização de sua forma no ajuste pelo método dos mínimos quadrados, que leva em consideração erros de medição, foi uma das principais razões do sucesso desta empreitada; sendo a outra, a alta competência técnica dos pesquisadores geofísicos envolvidos, vindos do Instituto de Pesquisas Espaciais, no Brasil, e do Center for Atmospheric and Space Sciences, da Universidade Estadual de Utah, nos EUA.

O objetivo secundário desta pesquisa foi efetivamente alcançado, pois confirmou-se a adequação de linguagens declarativas (tal como as linguagens funcionais, representadas neste trabalho pelo Objective Caml) em problemas de modelagem numérica, em que, normalmente, linguagens imperativas como C ou Fortran, são usadas. O número de linhas de código implementadas em todos os experimentos, incluindo os não bem sucedidos, não superou 1600, contando, nesse número, a vasta quantidade comentários espalhados ao longo do código. A eficiência do código gerado foi de tão alta qualidade que, em momento algum, se pensou em substituir OCaml por outra linguagem. A clareza do código permitiu que as equipes compartilhassem informações e sugerissem modificações de melhora ao longo do processo de pesquisa, funcionando como uma linguagem de especificação executável.

8.1 Trabalhos futuros

Em virtude da restrição do tempo e também da indisponibilidade de conhecimento geofísico local, algumas idéias não foram completamente desenvolvidas ou não foram desenvolvidas absolutamente.

As tempestades magnéticas possuem um padrão de comportamento medido pelos vários índices, como o índice K_p , o AE e outros. Assim, deve ser possível construir um sistema que as localize automaticamente com base nos dados coletados por décadas. A teoria das ondaletas pode ser usada para a caracterização dos dias geomagneticamente perturbados. Essa caracterização, aliada a algum procedimento estatístico de classificação (redes neurais, redes bayesianas, máquinas com vetor de suporte, etc.), usando algumas das técnicas já consagradas de inteligência artificial (árvores de decisão, computação evolutiva etc.) poderia resolver este problema. Essas mesmas técnicas poderiam ser usadas na previsão do aparecimento do traço F espalhado.

A mistura das ondaletas com o filtro de Kalman ou com o filtro H_∞ merece ser mais bem investigada. Isto possibilitaria a construção de um modelo dinâmico para a climatologia espacial. Talvez, o modelo construído com as curvas de Bézier fizesse o papel do modelo físico nos filtros.

Muitas variáveis estão presentes na previsão do comportamento das derivas zonais. Em todos os modelos vistos nesta tese, analisou-se somente a influência de uma variável por vez, por exemplo, a hora local, mantendo-se as outras constantes (o índice K_p ou o fluxo solar $F_{10,7}$ cm). Seria desejável um modelo que levasse em conta todas as variáveis importantes de uma só vez. Isto pode ser feito por meio da generalização das curvas de Bézier para superfícies de Bézier ou para o produto tensorial dos polinômios de Berns-

tein, vistos no Capítulo 5, embora a parametrização dessas hipersuperfícies possa não ser trivial.

Referências Bibliográficas

BARSKY, B. Computer-aided geometric design: A bibliography with keywords and classified index. **IEEE Computer Graphics and Applications**, v. 1, n. 3, p. 67–109, Jul 1981.

BERNSTEIN, S. Démonstration du théorème de weierstrass fondée sur le calcul des probabilités. **Comm. Soc. Math. Kharkov**, v. 13, 1-2, 1912.

BERTONI, F. C. P. **Derivas ionosféricas em latitudes equatoriais**: observações e modelagem. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, fevereiro 2004. Disponível em: <<http://mtc-m16.sid.inpe.br/col/sid.inpe.br/jeferson/2004/04.14.11.32/doc/publicacao.pdf>>. Acesso em: 5 de out. de 2006.

BOLZAN, M. J. A. Análise da transformada em ondeletas aplicada em sinal geofísico. **Revista Brasileira de Ensino de Física**, v. 26, n. 1, p. 37–41, 2004. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0102-47442004000100006&lng=en&nrm=iso>. Acesso em: 20 de ago. 2006.

CAMPBELL, W. H. **Introduction to Geomagnetic Fields**. 2nd. ed. Cambridge: Cambridge University Press, 2003.

CARRASCO, A. J. **O Pico Pré-Inversão na Deriva Vertical de Plasma e a Camada E Esporádica**. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, julho 2005.

DOMINGUES, M. O.; MENDES, O. J.; COSTA, A. M. On wavelet techniques in atmospheric sciences. **Advances in Space Research**, v. 35, n. 5, p. 831–842, 2005. Disponível em: <<http://www.lac.inpe.br/%7Emargarete/JASRWavelet.pdf>>. Acesso em: 5 de out. de 2006.

ELIEZER, S.; ELIEZER, Y. **The Fourth State of Matter**: An introduction to plasma science. 2nd. ed. Bristol: Institute of Physics Publishing, 2001.

FARIA, R. R. A. **Aplicação de Wavelets na Análise de Gestos Musicais em Timbres de Instrumentos Acústicos Tradicionais**. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, São Paulo, 1997. Disponível em: <<http://www.lsi.usp.br/~regis/DissertacaoMSc-RegisFaria.pdf>>. Acesso em: 12 de out. de 2006.

- FARIN, G. **Curves and Surfaces for Computer-Aided Geometric Design — A Practical Guide**. 5th. [S.l.]: Morgan Kaufmann Publishers (Academic Press), 2002. (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling).
- FEJER, B. G.; SCHERLIESS, L. Empirical models of storm time equatorial zonal electric fields. **J. Geophys. Res.**, v. 102(A11), n. 24,047-24,056, 1997.
- FEJER, B. G.; SOUZA, J. R.; SANTOS, A. S.; PEREIRA, A. E. C. Climatology of f region zonal plasma drifts over jicamarca. **J. Geophys. Res.**, v. 110, n. A12310, doi:10.1029/2005JA011324, December 2005.
- GLATZMAIER, G. A.; OLSON, P. Probing the geodynamo. **Scientific American**, v. 292, p. 50–57, April 2005. Disponível em: <<http://www.jhu.edu/~eps/faculty/olson/pdfs/Geodynamo.pdf>>. Acesso em: 6 de out. 2006.
- GOMES, F. do E. **Observação da luminescência atmosférica no estado da Paraíba**. Campina Grande, Agosto 2004. Disponível em: <<http://aerolume.df.ufcg.edu.br/~fabio/relatorio%20final2004.doc>>. Acesso em: 30 de set. 2006.
- GRINGS, A. **Regressão simbólica via Programação Genética: um estudo de caso com modelagem geofísica**. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Faculdade de Computação, fevereiro 2006.
- GZYL, H.; PALACIOS, J. L. On the approximation properties of bernstein polynomials via probabilistic tools. **Boletín de la Asociación Matemática Venezolana**, X, n. 1, p. 5–13, 2003. Disponível em: <<http://www.emis.de/journals/BAMV/conten%20-%20vol10/jopalagzyl.pdf>>. Acesso em: 3 de set. 2005.
- HONG, L.; CHENG, G.; CHUI, C. A filter-bank-based kalman filtering technique for wavelet estimation and decomposition of random signals. **IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing**, v. 45, p. 237–241, Feb 1998.
- HORMANN, K. Fitting free form surfaces. In: GIROD, B.; GREINER, G.; NIEMANN, H. (Ed.). **Principles of 3D Image Analysis and Synthesis**. Boston: Kluwer Academic Publishers, 2000, (The International Series in Engineering and Computer Science, v. 556). cap. 4.7, p. 192–202. Disponível em: <<http://www2.in.tu-clausthal.de/~hormann/papers/Hormann.2000.FFF.pdf>>. Acesso em: 30 de jul. 2005.
- KALMAN, R. E. A new approach to linear filtering and prediction problems. **Transactions of the ASME—Journal of Basic Engineering**, v. 82, n. Series D, p. 35–45, 1960. Disponível em: <<http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf>>. Acesso em: 30 de jul. 2006.
- KAPPENMAN, J. G.; ZANETTI, L. J.; RADASKY, W. A. Geomagnetic storms can threaten electric power grid. **American Geophysical Union**, v. 9, n. 7, p. 9–11, March 1997. Disponível em: <http://www.agu.org/sci_soc/eiskappenman.html>. Acesso em: 10 de mar. 2007.

- KELLEY, M. C. **The Earth's ionosphere**. New York: Academic Press, 1989. 487 p.
- KUDEKI, E.; BHATTACHARYYA, S.; WOODMAN, R. A new approach in incoherent scatter f-region e x b drift measurements atjicamarca. **Journal of Geophysical Research**, v. 104, n. A12, p. 28,145–28,162, 1999.
- LEROY, X.; DOLIGEZ, D.; GARRIGUE, J.; RÉMY, D.; VOUILLON, J. **The Objective Caml system release 3.09**: Documentation and user's manual. [S.l.], 2004. Disponível em: <<http://caml.inria.fr/pub/docs/manual-ocaml/>>. Acesso em: 14 de nov. de 2004.
- LORENTZ, G. G. **Bernstein Polynomials**. 2nd. ed. [S.l.]: Chelsea Pub Co, 1986.
- MILONE, A. de C.; WUENSCH, C. A.; VILEGARODRIGUES, C.; JABLONSKI, F. J.; CAPELATO, H. V.; VILAS-BOAS, J.; CECATTO, J. R.; NETO, T. V. **Introdução à astronomia e astrofísica**. São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2003. (INPE-7177-PUD/38). Disponível em: <<http://www.cea.inpe.br/webdas/curso/PDF/completo.pdf>>. Acesso em: 1 de out. 2006.
- MIRANDA, J. M. A. de. **Introdução ao Geomagnetismo**. Lisboa: Instituto Geofísico do Infante Dom Luiz, 2002. Disponível em: <<http://www.igidl.ul.pt/download.htm>>. Acesso em: 5 de out. 2006.
- MORETTIN, P. A. **Ondas e Ondaletas**: Da análise de fourier à análise de ondaletas. São Paulo: Editora da Universidade de São Paulo, 1999.
- POLIKAR, R. **The Wavelet Tutorial**. 2001. Disponível em: <<http://users.rowan.edu/~polikar/WAVELETS/WTtutorial.html>>. Acesso em: 12 de jan. de 2005.
- POULARIKAS, A. D. **The Transforms and Applications Handbook**. 2nd. ed. Boca Raton: CRC Press, 2000.
- PRESS, W. H.; TEUKOLSKY, S. A.; VETTERLING, W. T.; FLANNERY, B. **Numerical Recipes in C: The Art of Scientific Computing**. 2nd edition. ed. Cambridge, UK: Cambridge University Press, 1993.
- PROTÁZIO, J. M. B. **Análise Wavelet Aplicada a Sinais Geofísicos**. Dissertação (Mestrado) — Universidade Estadual de Campinas, Campinas, 2001.
- RAIDL, G. R.; KODYDEK, G. Evolutionary optimized tensor product bernstein polynomials versus backpropagation networks. In: **Proc. of the Int. ICSC/IFAC Symposium on Neural Computation**. Vienna, Austria: [s.n.], 1998. p. 885–890. Disponível em: <<http://www.ads.tuwien.ac.at/publications/bib/pdf/raidl-98.pdf>>. Acesso em: 24 de abr. de 2005.
- RAIDL, G. R.; WURM, C. Approximation with evolutionary optimized tensor product bernstein polynomials. In: **Proc. of the Int. Conference on Artificial Intelligence in Industry: From Theory to Practice**. High Tatras, Slovakia: [s.n.], 1998. p. 247–256. Disponível em: <<http://www.ads.tuwien.ac.at/publications/bib/pdf/raidl-98d.pdf>>. Acesso em: 24 de abr. de 2005.

SCHERLIESS, E.; FEJER, B. G. Radar and satellite global equatorial f-region vertical drift model. **J. Geophys. Res.**, v. 104, n. 6829-6842, 1999.

SHENE, C.-K. **CS3621 Introduction to Computing with Geometry Notes**.

Michigan: Michigan Technological University, 2003. Disponível em:

<<http://www.cs.mtu.edu/~shene/COURSES%20-%20cs3621/NOTES/notes.html>>. Acesso em: 25 de nov. 2004.

SIMON, D. From here to infinity. **Embedded Systems Programming**, v. 14, n. 11, p. 20–32, October 2001. Disponível em:

<<http://academic.csuohio.edu/simond/courses/eec641/hinfinity.pdf>>. Acesso em: 30 de set. 2006.

SIMON, D. Kalman filtering. **Embedded Systems Programming**, v. 14, n. 6, p. 72–79, June 2001. Disponível em:

<<http://academic.csuohio.edu/simond/courses/eec644/kalman.pdf>>. Acesso em: 30 de set. 2006.

SIMON, D. **Optimal State Estimation: Kalman, h infinity, and nonlinear approaches**. [S.l.]: Jonh Wiley & Sons, 2006.

SORENSEN, H. W. Least-squares estimation: from gauss to kalman. **IEEE Spectrum**, v. 7, p. 63–68, July 1970. Disponível em:

<<http://www.cs.unc.edu/~welch/kalman/media/pdf/Sorenson1970.pdf>>. Acesso em: 30 de jul. 2006.

STARK, H.-G. **Wavelets and Signal Processing: An application-based introduction**. Berlin: Springer-Verlag, 2005.

SWELDENS, W.; SCHRÖDER, P. Building your own wavelets at home. In: **Wavelets in Computer Graphics**. ACM SIGGRAPH Course notes, 1996. p. 15–87. Disponível em: <<http://cm.bell-labs.com/who/wim/papers/athome.pdf>>. Acesso em: 25 de ago. 2003.

WIKIPÉDIA. **Fluxo solar — Wikipédia, a enciclopédia livre**. 2006. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Fluxo_solar&oldid=2276331>. Acesso em: 7 de out. de 2006.

WOODMAN, R. East-west ionospheric drifts at the magnetic equator. **Space Research XII**, v. 12, p. 969–974, 1972.

YAMASHITA, C. S. **Efeito das tempestades magnéticas intensas na ionosfera de baixa latitude**. Dissertação (Mestrado) — INPE, São José dos Campos, agosto 1999.

ZHOU, Y. **Surface Interpolation and Approximation**. Dissertação (Mestrado) — Michigan Technological University, Michigan, February 1999. Disponível em: <<http://www.cs.mtu.edu/~shene/NSF-2/YanZhouReport.tar.gz>>. Acesso em: 22 de abr. de 2004.

Apêndice A - Climatology of F region zonal plasma drifts over Jicamarca

Este apêndice contém o principal artigo derivado desta pesquisa, publicado no *Journal of Geophysical Research* em dezembro de 2005.

Climatology of F region zonal plasma drifts over Jicamarca

B. G. Fejer,¹ J. R. Souza,² A. S. Santos,³ and A. E. Costa Pereira⁴

Received 18 July 2005; revised 16 September 2005; accepted 22 September 2005; published 21 December 2005.

[1] We use extensive incoherent scatter radar observations made at the Jicamarca Radio Observatory between 1970 and 2003 to study and model empirically the equatorial zonal plasma drifts near the F region peak using Bernstein polynomials as base functions. Our quiet-time model results confirm that the daytime drifts are westward and are nearly season and solar cycle independent. The nighttime drifts are eastward, have larger magnitudes, and increase strongly with solar flux, particularly near equinox and December solstice. Enhanced geomagnetic activity drives small eastward perturbation drifts during the day and much larger westward disturbance drifts at night. The nighttime drift perturbations are largest near midnight and increase strongly with solar flux near equinox and December solstice but are essentially absent near June solstice. The Jicamarca zonal disturbance drifts can be largely accounted for by disturbance dynamo electric fields with a dominant time delay of about 3–15 hours following enhanced geomagnetic activity. In the postmidnight sector, there are also smaller westward disturbance drifts associated with time delays of about 15–24 hours and perhaps even longer. Our results strongly suggest that the longitudinal dependence of both the quiet and disturbed equatorial nighttime zonal drifts varies with season.

Citation: Fejer, B. G., J. de Souza, A. S. Santos, and A. E. Costa Pereira (2005), Climatology of F region zonal plasma drifts over Jicamarca, *J. Geophys. Res.*, 110, A12310, doi:10.1029/2005JA011324.

1. Introduction

[2] Electric fields and plasma drifts are fundamentally important ionospheric parameters. Middle and low-latitude F region ionospheric electrodynamic drifts are mostly driven by E and F region dynamo electric fields [e.g., Richmond, 1995], but during geomagnetically active times they can be strongly affected by solar wind-magnetospheric and ionospheric disturbance dynamo electric fields [e.g., Fejer, 1997]. The complex interactions of these processes cause strong plasma drift variability with a large range of timescales. The understanding of the morphology and sources of this variability is essential for improved ionospheric forecasting and testing of predictions from global ionospheric and convection models.

[3] Incoherent scatter radar measurements of F region equatorial vertical plasma drifts have routinely been made since 1968 at the Jicamarca Radio Observatory near Lima, Peru (11°57'S, 76°52'W; magnetic dip 2°N) [e.g., Woodman, 1970]. These observations were used in numerous case and climatological studies and, more recently, also in the development of empirical equatorial vertical drift

models for quiet and disturbed conditions [Fejer and Scherliess, 1997; Scherliess and Fejer, 1999]. Simultaneous measurements of F region vertical and zonal plasma drifts [Woodman, 1972] have been made at Jicamarca since 1970 but less frequently than the vertical drifts, since the simultaneous measurement of these two velocity components leads to larger errors on the vertical drifts. As a result, the database of Jicamarca zonal drifts is much smaller than that of the vertical drifts.

[4] The Jicamarca F region quiet time zonal drifts are westward during the day with typical values of about 40 m/s, and they do not change much with altitude, season, and solar activity [Fejer et al., 1985, 1991]. The nighttime zonal drifts are eastward near the F region peak and above with maximum values of about 100 and 150 m/s near solar minimum and maximum, respectively. These drifts are largest at about 2100–2200 local time, and they decrease toward dawn. The zonal nighttime drifts exhibit noticeable day-to-day, seasonal, and solar cycle effects. Between about dusk and midnight, there is a strong shear on the zonal plasma flow below the F peak, where the eastward drifts decrease with height and become westward at lower altitudes [e.g., Fejer et al., 1985; Kudeki et al., 1981; Kudeki and Bhattacharyya, 1999]. These early night eastward and westward zonal drifts are associated with downward and upward motions, respectively, which results in a plasma flow vortex below the F layer peak. At higher altitudes, the nighttime eastward drifts decrease slowly with height.

[5] The Jicamarca zonal drifts are in good agreement with global equatorial in situ drift measurements from the DE-2 satellite [Coley and Heelis, 1989] and are also consistent with satellite thermospheric zonal wind observations [e.g.,

¹Center for Atmospheric and Space Sciences, Utah State University, Logan, Utah, USA.

²Instituto de Pesquisas Espaciais (INPE), São José dos Campos, São Paulo, Brazil.

³Department of Computer Science, Federal University of Goiás, Catalão, Goiás, Brazil.

⁴Department of Electrical Engineering, Universidade Federal de Uberlândia, Uberlândia, Minas Gerais, Brazil.

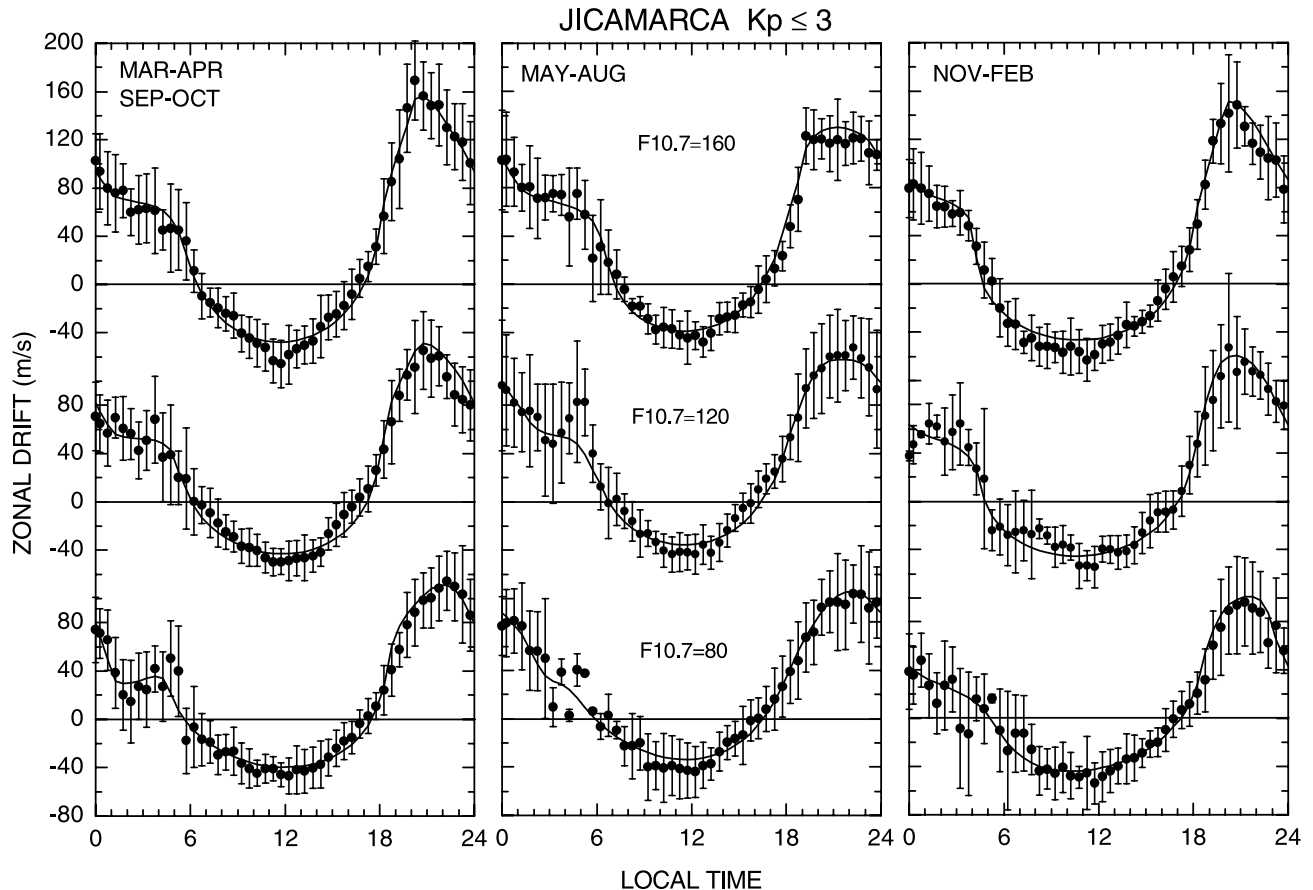


Figure 1. Seasonal and solar cycle dependence of average F region quiet-time zonal drifts over Jicamarca. The scatter bars denote the standard deviations and the solid curves represent the model results.

Herrero et al., 1985; Fejer et al., 1985]. The nighttime radar drifts are in good general agreement with equatorial spaced receiver scintillation drifts [Valladares et al., 1996; Sheehan and Valladares, 2004] and also with zonal drifts of ionospheric plasma bubbles inferred from ground-based [e.g., Sobral and Abdu, 1991; Sobral et al., 1999; Martinis et al., 2003; Terra et al., 2004] and satellite optical imaging data [Immel et al., 2004a].

[6] The Jicamarca zonal drifts sometimes show significant geomagnetic effects at night. However, since they also exhibit strong solar cycle dependence and large quiet time variability throughout the day, relatively large databases are needed to determine their storm-time dependence. Initial climatological studies showed essentially no daytime geomagnetic effects and only relatively small decreases on the average nighttime eastward drifts [e.g., Fejer et al., 1985, 1991]. Later studies indicated that the perturbation or disturbance drifts, obtained by subtracting the quiet time values, are eastward with small values during the day, and westward, with larger magnitudes, at night [Fejer, 1997]. These past studies have not examined the time constants of zonal disturbance drifts and their possible dependence on season and solar flux.

[7] In this paper we use extensive Jicamarca zonal drift observations to study and model these drifts as a function of season, solar cycle, and magnetic activity. In the following sections, we first give a short description of the Jicamarca

zonal drift measurements. Next, we present our results and compare with those from earlier studies. The methodology for deriving our empirical statistical models, which use Bernstein polynomials [e.g., Lorentz, 1986] as bases functions, is presented in Appendix A.

2. Measurements and Data

[8] The basic procedure for measuring F region drifts at Jicamarca was described in detail by Woodman [1972]. Briefly, the large 50 MHz antenna is split into two beams perpendicular to the geomagnetic field and pointed about 2.5° to the east and 4.3° to the west of vertical, giving a net split of about 6.8° . These line-of-sight drifts are combined to give the vertical and zonal drift components. Our measurements were usually made from about 200 to 900 km with a resolution of about 15 km, and an integration time of about 5 min. The data used in this study correspond to averages near the F region peak (typically between about 300 and 500 km but higher near dusk), where they do not change much with altitude, and the signal-to-noise ratios are highest. Reliable drift measurements are not possible during periods of spread F occurrence over a large range of altitudes.

[9] We have used 206 days of measurements from 1970 through 1992 and 86 days from 1994 through 2003; there were no measurements during 1979, 1982–1983, and 1993.

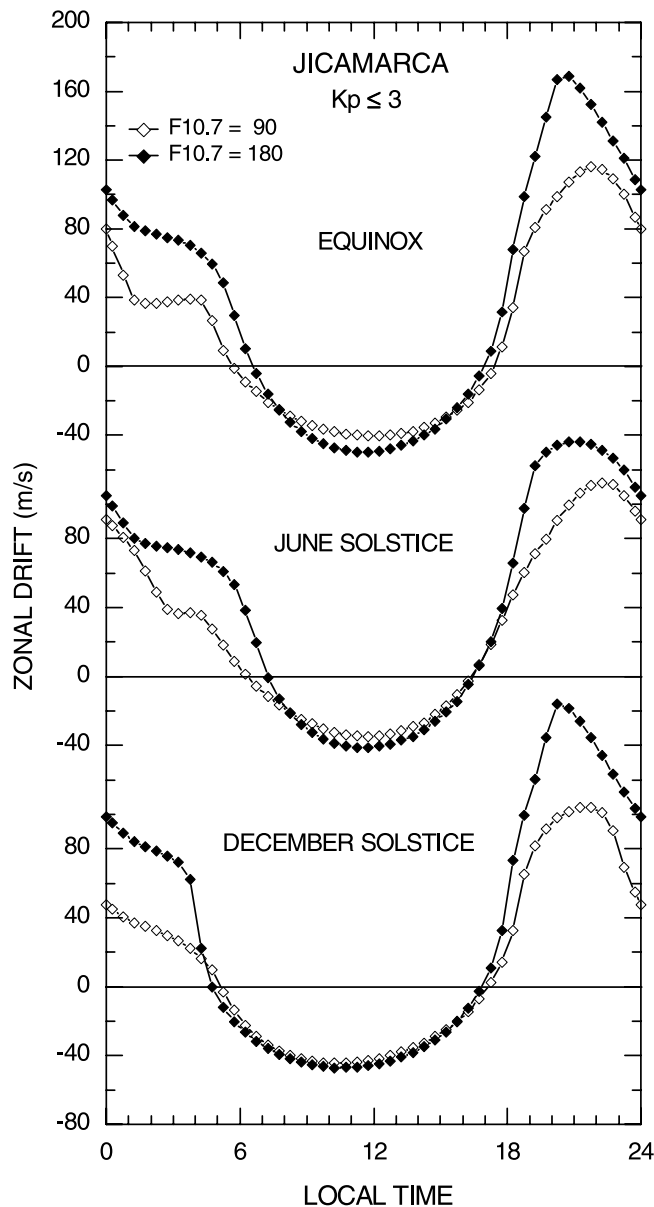


Figure 2. Comparisons of quiet-time Jicamarca model zonal drifts for low and moderately high solar flux conditions.

The number of early night observations (during equinoctial and southern hemisphere summer) is much smaller than during the day, due to frequent spread F occurrence over the entire measured altitudinal range. For an integration time of 5 min, the typical accuracy of the daytime zonal drift measurements was about 15 m/s before 1994 and about 5 m/s after 1994, when the more powerful present data analysis technique [Kudeki et al., 1999] was implemented. The nighttime measurements have larger errors (up to about 30 m/s), particularly during late night solar minimum periods, as a result of much lower signal-to-noise ratios.

[10] In the present study, the drift data were binned in half an hour intervals centered 15 and 45 min after the hour, and into seasonal bins representing June solstice (May–August), December solstice (November–February), and equinox (March–April, September–October). We define

solar activity as a function of the $F_{10.7}$ cm solar flux index. Since the AE indices are not available over the entire period of our measurements, we define geomagnetic activity in terms of the K_p index with quiet time conditions defined by an average $K_p \leq 3.0$ over the preceding 9 hours. The use of the K_p index does not allow us to examine prompt penetration electric field effects, but this is not a major limitation since prompt penetration equatorial zonal drifts are generally much smaller than disturbance dynamo drifts [e.g., Fejer et al., 1990; Fejer and Emmert, 2003].

3. Results and Discussion

[11] In this section, we first briefly present our zonal drift results for geomagnetically quiet times, which are largely an extension of the results from our previous studies. Then, we examine in detail the characteristics of the zonal disturbance drifts obtained by modeling the residual (i.e., disturbed minus corresponding average quiet time values) drifts as a function of local time, storm-time (i.e., time delay between increased geomagnetic activity and generation of disturbance drifts), season and solar flux. Finally, we discuss the drift disturbance mechanisms, and compare our results with those from other recent studies.

3.1. Quiet Time Zonal Drifts

[12] Figure 1 compares the seasonally averaged and our model zonal drifts for three solar flux levels and geomagnetically quiet times (average $K_p \approx 1.8$ over the preceding 9 hours). The methodology used for deriving our quiet time drift model is presented in Appendix A. The vertical bars denote the variability of the binned data; the measurement errors usually have much smaller values. Figure 1 shows that the empirical model results are in good agreement with the averaged data. The general characteristics of the zonal drifts, presented in Figure 1, are consistent with those from earlier studies [e.g., Fejer et al., 1985, 1991; Coley and Heelis, 1989]. These drifts are westward and largely season-independent during the day, with peak values of about 40 m/s, and eastward at night, with much larger magnitudes.

[13] Figure 2 shows the quiet time model drifts for typical low and high solar flux conditions. This figure illustrates that the magnitudes of the daytime westward drifts and their reversal times near dawn and dusk do not change much with solar flux. It also illustrates that near solar maximum the nighttime eastward drifts have larger magnitudes and that the times of peak eastward drifts occur earlier close to solar maximum. Figure 3 shows the good agreement of the binned and calculated standard deviations for three solar flux levels. Our calculations are less accurate in the late night sector due to the larger error and smaller number of measurements.

[14] Fejer et al. [1991] reported that the yearly averaged maximum eastward velocity over Jicamarca can be expressed as $V(\text{m/s}) = 60 + 0.45\phi$, where ϕ is the decimetric solar flux index. Figure 2 indicates, however, that the increase of the Jicamarca nighttime peak eastward drifts with solar activity is much stronger during December solstice and equinox than during June solstice. This is shown in more detail in Figure 4, where the average eastward drifts between 2030 and 2130 local time are plotted as a function of the 10.7 cm solar flux index. For

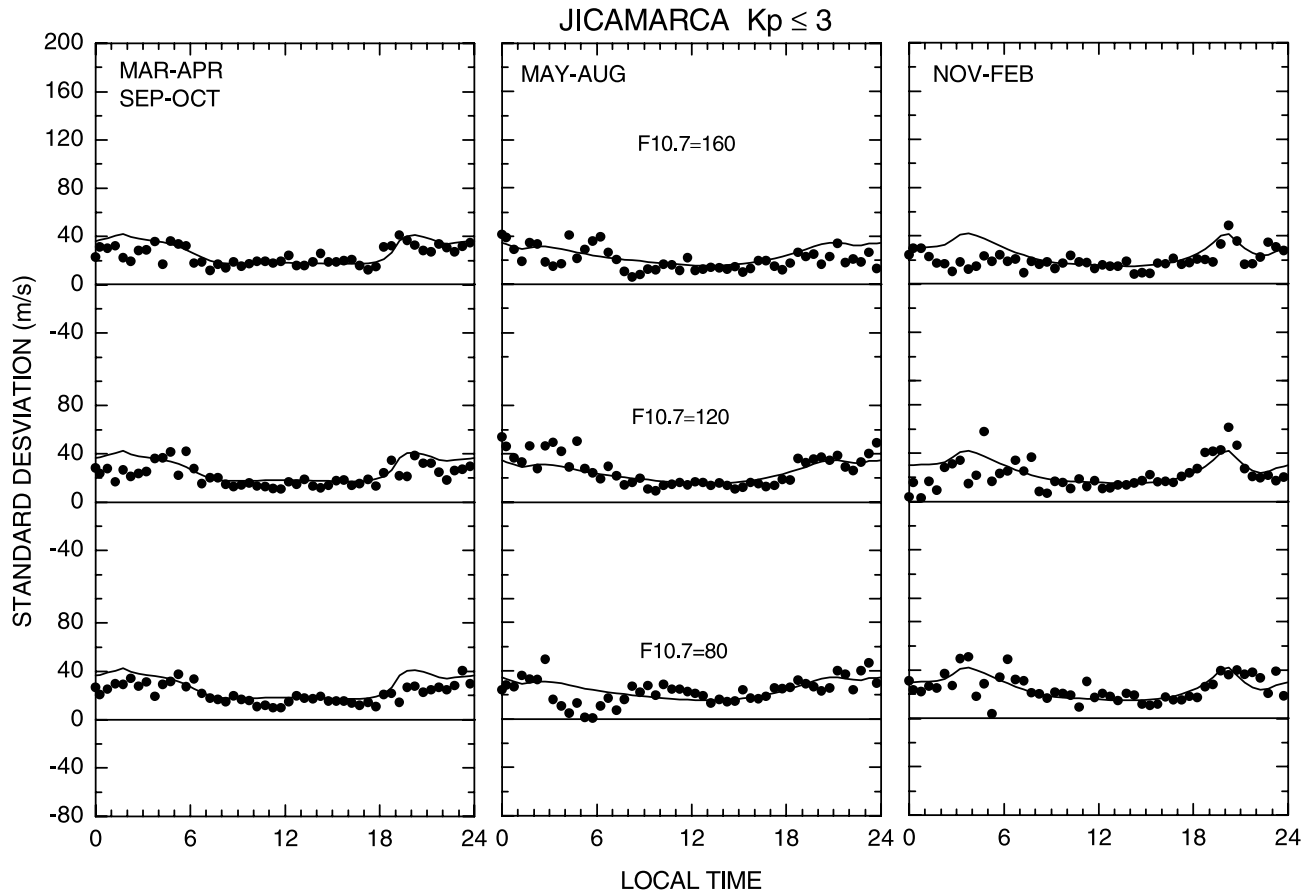


Figure 3. Standard deviations of the zonal drift obtained by binning the data and calculated using our analytical empirical model.

large solar fluxes, the equinox and December solstice peak eastward drift values are nearly identical and much larger than during June solstice. It is well known that the Jicamarca evening upward drifts, which lift the F layer to higher altitudes and play an important role on the occurrence of spread F , are also largest during equinox and December solstice solar maximum conditions. This strong correlation of the evening upward and nighttime eastward plasma drifts is not surprising since they are both strongly affected by conductivity-dependent F region dynamo electric fields [e.g., *Rishbeth, 1971*].

[15] Ground-based and satellite optical measurements and spaced receiver UHF scintillation observations [*Sobral et al., 1999; Martinis et al., 2003; Immel et al., 2004a; Terra et al., 2004; Sheehan and Valladares, 2004*] indicate that the nighttime zonal drifts of equatorial plasma irregularities and depletions have similar local time and solar flux dependence as the radar drifts presented above. On the other hand, scintillation zonal drift measurements over Peru during June solstice show that the drifts of the plasma irregularities have generally larger magnitudes and stronger increase with solar flux than our incoherent radar drifts [e.g., *Sheehan and Valladares, 2004*]. The occurrence of equatorial spread F over Jicamarca during June solstice requires evening upper drift velocities and layer heights which are much larger than their typical values. Therefore these results also indicate that larger upward drift velocities are indeed associated with larger eastward plasma drifts.

[16] *Immel et al. [2004a]* showed that the zonal plasma drift speeds of equatorial nighttime plasma depletions observed with the imager aboard the high-apogee IMAGE satellite during March–May 2002 had a strong longitudinal dependence with a peak around the Indian sector. This longitudinal variation of the zonal drifts should be due at least in part to the strong dependence of the evening upward drifts on longitude. Since the longitudinal variation of the

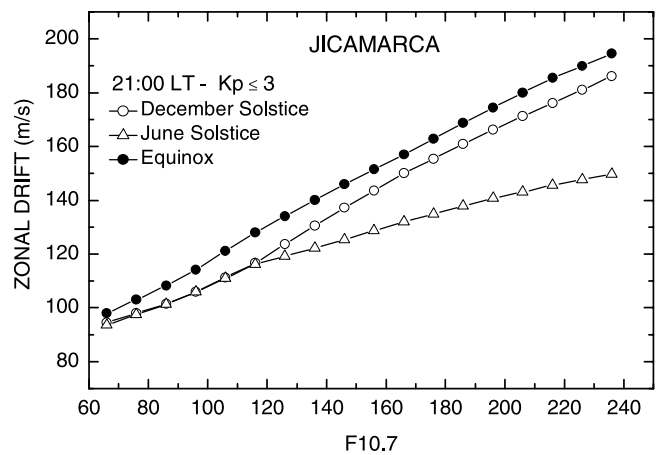


Figure 4. Solar flux dependence of the peak nighttime eastward drifts.

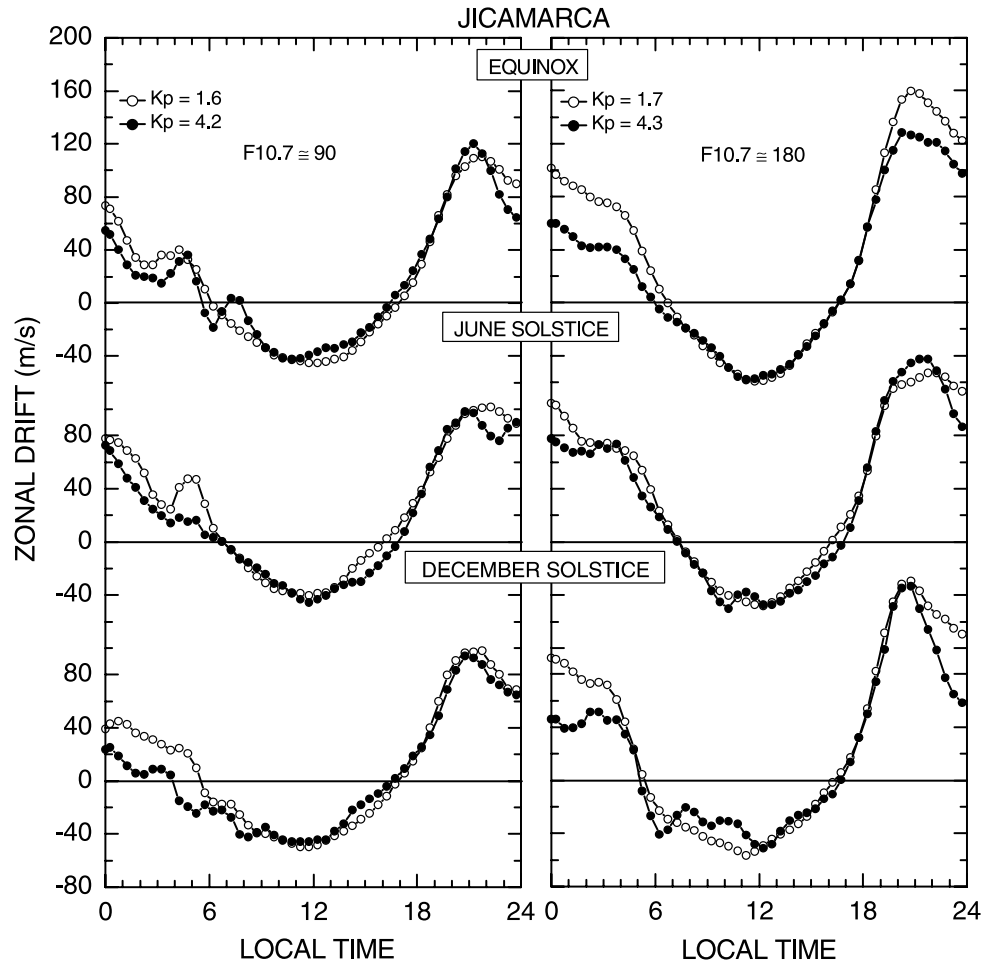


Figure 5. Seasonal and solar cycle dependence of quiet and disturbed average Jicamarca zonal drifts.

evening upward plasma drifts is strongly season-dependent [Scherliess and Fejer, 1999], we expect that the longitudinal variation of the zonal drifts to change with season.

3.2. Disturbed Zonal Drifts

[17] Figure 5 shows the local time and seasonal variations of the Jicamarca average zonal drifts during quiet time and disturbed conditions for low and moderately high solar fluxes. This figure indicates that enhanced geomagnetic activity drives mostly nighttime westward perturbations with largest magnitudes near midnight and during equinox and December solstice high solar flux periods. Surprisingly, it also shows that over Jicamarca the zonal disturbance drifts are very small during June solstice. We will show later that the local time variation of the perturbation drifts between September and April is consistent with the predictions of the ionospheric disturbance dynamo theory [Blanc and Richmond, 1980]. However, this theory did not address the seasonal and solar cycle dependence of these perturbation drifts.

[18] Scherliess and Fejer [1997] showed that the Jicamarca disturbance dynamo vertical drifts have typical lifetimes of several hours and that they occur up to about 30 hours after large increases in geomagnetic activity. We have studied the dependence of the zonal drifts on storm-time (i.e., a period after a given change in magnetic activity)

by using time series of K_p indices lasting up to 15 hours and time delays up to 30 hours; the use of AE indices would have decreased significantly our database. Following Fejer and Scherliess [1997], we first determined the perturbation drifts by subtracting from each half an hour averaged drift the corresponding season and solar cycle dependent average quiet time value. To account for solar cycle effects, we have studied separately the perturbations drifts corresponding to decimetric solar flux indices smaller or equal to 140, and larger than 120. These slightly overlapping bins optimize the seasonal and solar cycle-dependent distribution of our data.

[19] Our initial analysis showed that the present database is not extensive enough to fully characterize the complex local time, season, and solar flux dependence of the perturbation drifts. In the evening sector, for example, the time delays between enhanced geomagnetic activity and peak westward disturbance drifts seem to be shortest during equinox solar maximum conditions. Since Figure 5 indicates that the disturbed drift patterns are similar during equinox and December solstice, we have combined the perturbation drifts from these two seasons in order to improve the statistical significance of our results. The June solstice disturbance drifts will not be studied further since their magnitudes are generally very small and their database is also more limited.

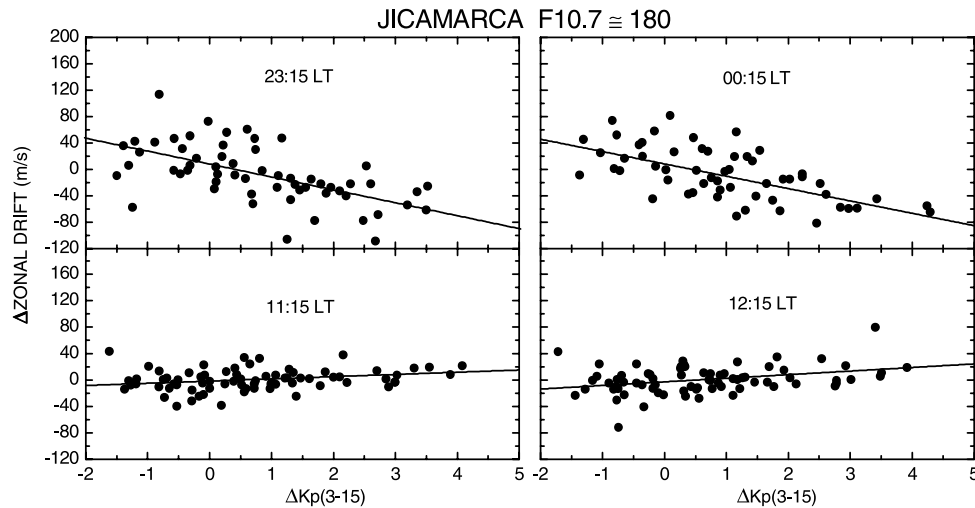


Figure 6. Variation of zonal perturbation drifts with magnetic activity near midnight and noon. Here $\Delta K_p(3-15) = 0$ corresponds to an average $K_p = 1.8$ over the preceding 3 to 15 hours.

[20] We observed that the equinoctial and December solstice Jicamarca disturbance drifts during low and high solar flux periods are most closely associated with the average levels of enhanced geomagnetic activity over the previous 12–15 hours. We have modeled these perturbation drifts as a function of $\Delta K_p(3-15)$ hours, the average K_p values over periods of 3–15 hours above our average quiettime value ($K_p \approx 1.8$), although nearly identical results were obtained using $\Delta K_p(0-12)$ hours, and for high solar fluxes also for $\Delta K_p(3-9)$ hours). Figure 6 shows the variation of the moderate to high solar flux perturbation drifts near midnight and noon as a function of $\Delta K_p(3-15)$ hours. These zonal disturbance drifts vary nearly linearly over the entire range of this parameter. Figure 6 also indicates that geomagnetic activity affects the equatorial zonal drifts even under nearly quiet conditions. For $\Delta K_p(3-15) = 5$, the average magnitudes of these perturbation drifts are about -90 m/s and 15 m/s near midnight and noon, respectively.

[21] Our analysis also indicated the occurrence of noticeable late night perturbations drifts associated with a time delay of about 15–24 hours. These late-night disturbance drifts vary nearly linearly with $K_p(15-24)$ hours. It is possible that the zonal drifts are affected by geomagnetic activity associated time delays even longer than 24 hours, but these longer-term effects cannot be derived reliably from our database. Therefore we modeled our disturbance drifts using $\Delta K_p(3-15)$ hours and $\Delta K_p(15-24)$ hours as input parameters.

[22] Figure 7 shows the low and solar flux average zonal disturbance drift patterns associated with our two dominant disturbance time delays. The low and high flux perturbation drifts have similar local time patterns but have larger magnitudes during high solar flux conditions. The top panel shows that $\Delta K_p(3-15)$ perturbations are eastward with small values during the day and, as expected, westward with larger magnitudes at night. The $\Delta K_p(3-15)$ premidnight low solar flux disturbance drifts have much larger magnitudes than the perturbation drifts inferred from the low solar flux equinoctial and December solstice data shown in Figure 5, but the same is not true for the higher

flux drifts. These results again indicate that the time delay between the onset of geomagnetic activity and the development of equatorial zonal disturbance drifts are longer near solar minimum than close to solar maximum. The solid curve in Figure 7, which corresponds to an average decimetric solar flux index of about 130 units, is in good agreement with the solar flux averaged disturbance drifts presented by Fejer [1997], which correspond to $\Delta K_p(1-12)$ hours = 3. The lower panel in Figure 7 indicates that for time delays longer than about 15 hours the daytime perturbation drifts vanish and that the magnitude of the westward perturbation drifts decrease significantly in the early night sector. They remain large in the late night period, where they have largest values at about 0400 LT and for high solar flux conditions. Figure 8 illustrates the excellent agreement

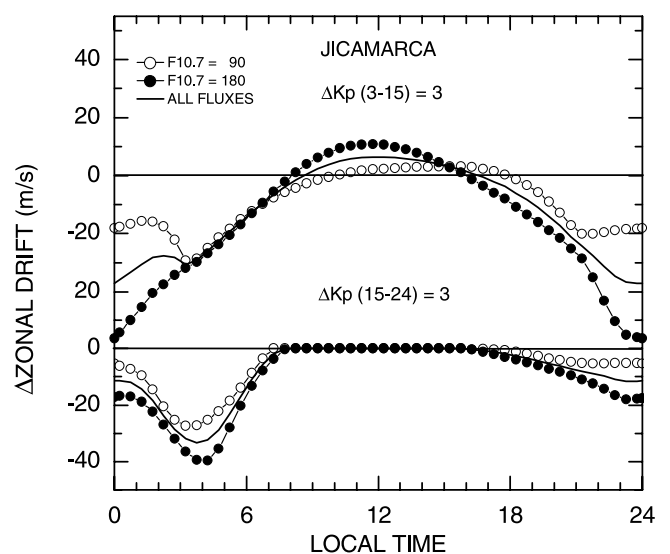


Figure 7. Average zonal disturbance drifts corresponding to $\Delta K_p = 3$ over 2 preceding periods. The solid curves denote the disturbance drifts for moderate solar flux conditions.

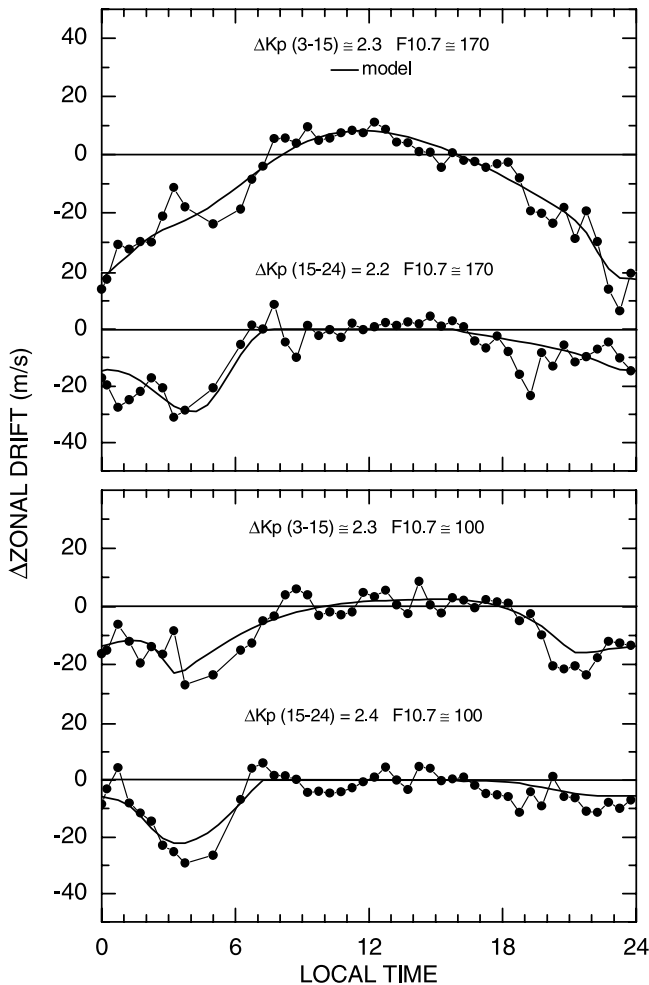


Figure 8. Comparisons of modeled and binned zonal disturbance drifts.

between our empirical disturbance model drifts and the corresponding binned data and again the solar cycle dependence of these disturbance drifts.

3.3. Comparison With Previous Zonal Disturbed Drift Studies

[23] Optical and radio measurements have recently been used to measure the effect of magnetic activity on the zonal drifts of nighttime equatorial plasma depletions. *Sheehan and Valladares* [2004] modeled the dependence of nighttime equatorial spaced receiver scintillation drifts observed in the Peruvian equatorial region as a function of $F_{10.7}$ solar flux and K_p index. They showed that the eastward scintillation drifts decrease and even reverse to westward as geomagnetic activity increases, with the largest changes in the postmidnight sector and from January through April. This study suggests that the effect of geomagnetic activity on the scintillation drifts is independent of solar flux. *Immel et al.* [2004a] reported that the largest decrease of the eastward drifts of equatorial plasma depletions observed by NASA-IMAGE satellite occurs during periods of large negative Dst values. More recently, *Immel et al.* [2004b] pointed out that these westward perturbation drifts are associated with increased K_p indices occurring from a few

hours to about a day earlier, which is consistent with our results.

[24] The main processes responsible for significant equatorial disturbance drifts are short-lived (timescales of about an hour) prompt penetration electric fields resulting from the transient leakage of high-latitude electric fields and currents to lower latitudes during periods of large changes in magnetospheric convection [e.g., *Kelley et al.*, 1979; *Senior and Blanc*, 1984; *Spiro et al.*, 1988], and longer-lasting (lifetimes from a few hours to about a day) ionospheric disturbance dynamo electric fields driven by enhanced energy deposition (mostly through Joule heating) into the high-latitude ionosphere during periods of strong convection [e.g., *Blanc and Richmond*, 1980]. Figure 9 shows in the upper panel the zonal prompt penetration drift pattern for an increase in the cross polar cap potential by 45 kV (corresponding approximately to $\Delta K_p = 3$) derived from Rice Convection Model (RCM) perturbation electric fields run presented by *Spiro et al.* [1988] and *Fejer et al.* [1990]. The prompt penetration drifts were obtained by first dividing the equatorward (horizontal) model electric field given in Figure 6 of *Fejer et al.* [1990] by the sine of the dip angle and then calculated considering that in the Peruvian equatorial region a downward electric field of 1 mV/m corresponds to an F region eastward drift velocity of 40 m/s. Other convection models [e.g., *Peymirat et al.*, 2000; *Tsunomura*, 1999] and more recent RCM simulations also give very similar results. Figure 9 shows westward daytime prompt penetration drifts with largest values near noon and eastward drifts at night with very small magnitudes, except near dawn. The convection models indicate that after sudden decreases in the polar cap potential drop of the same magnitude, the prompt penetration drifts have opposite polarity and comparable magnitudes. Evidently, these short-lived prompt penetration drifts do not play an important role on our longer-lasting average perturbation drifts, which are largest near midnight. However, during periods of large and rapid changes in magnetospheric convection, they should significantly affect the variability of daytime and late-night zonal drifts, especially near solar minimum when prompt penetration electric fields are largest.

[25] The bottom panel shows the zonal disturbance drift pattern predicted by the ionospheric disturbance dynamo theory 9 hours after a step function increase in the energy deposition into the high-latitude ionosphere, corresponding approximately to $\Delta K_p = 3$. Figures 8 and 9 show good agreement between the disturbance dynamo drifts and the Jicamarca short-term perturbation drifts near solar minimum, but not close to solar maximum, when the empirical disturbance drifts have largest magnitudes centered near midnight. Surprisingly, the nighttime theoretical drift pattern is also consistent with the longer-term perturbation drifts shown in Figure 8. The Blanc-Richmond disturbance theory does address the solar flux and seasonal dependence of the disturbance dynamo drifts. Recently, *Huang et al.* [2005] presented numerical simulations showing that post-storm meridional ionospheric potentials can last for several days.

[26] We have seen that the evening and nighttime vertical drifts and quiet and disturbance Jicamarca zonal drifts have similar seasonal and solar cycle dependence. In this local

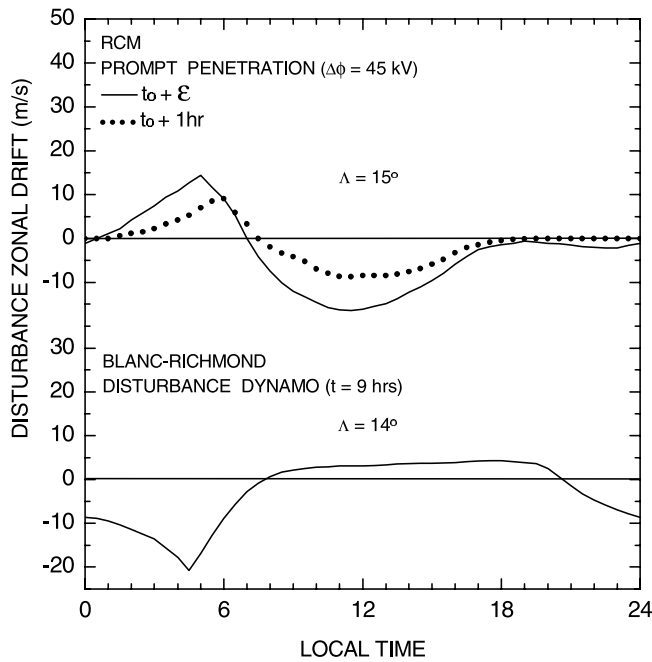


Figure 9. (a) Theoretical prompt penetration zonal drift following $\Delta K_p \sim 3$ (adapted from Fejer *et al.* [1990]). (b) Model disturbance dynamo drifts following an increase in the energy input into the high latitude ionosphere over the preceding 12 hours and corresponding to $\Delta K_p \sim 3$ (adapted from Blanc and Richmond [1980]). Here Λ denotes the invariant latitude.

time, these drifts are strongly affected by conductivity-dependent F region dynamo electric fields. Therefore our results suggest that during the solstices, when equatorial evening and nighttime vertical drift velocities vary most strongly with longitude [Scherliess and Fejer, 1999], so should the disturbance dynamo drifts, particularly during solar maximum.

[27] Equatorial satellite and ground-based spaced receiver measurements sometimes show very large (up to about 200 m/s) premidnight westward disturbance plasma drifts when the height of the F layer is raised to very high altitudes by strong upward prompt penetration plasma drifts [e.g., Basu *et al.*, 2001]. These large perturbation drifts cannot be explained as due entirely to either the prompt penetration and disturbance dynamo mechanisms discussed above, and they are most likely due to strong ion Hall conduction effects [Haerendel *et al.*, 1992; Fejer and Emmert, 2003]. This process drives westward drifts proportional to the ratio of the field line integrated Hall to Pedersen conductivities times the eastward electric field [Haerendel *et al.*, 1992] and is largely responsible for the postsunset velocity vortex in the bottom-side of the equatorial F layer [Kudeki and Bhattacharyya, 1999; Martinis *et al.*, 2003]. Large prompt penetration eastward electric fields (upward plasma drifts) near dusk strongly enhance this mechanism driving very large westward drifts at lower altitudes. Large upward drifts also lift the F layer peak to higher altitudes (sometimes above 800 km) and drive intense equatorial spread F and scintillations over a broad range of altitudes. Since the measurements used here are not possible during periods of strong spread F and were restricted to altitudes near and above the F layer peak during

other times, they are not strongly affected by ion Hall conduction effects. On the other hand, this is not the case for spaced receiver scintillation and DMSP satellite drift observations near the equator, which can be strongly affected by this process.

4. Summary

[28] We have studied the morphology of equatorial zonal drifts near the F region peak using extensive Jicamarca incoherent scatter observations. Our quiet time Jicamarca zonal drifts are in good agreement with results from previous radar studies and other measurement techniques. We have developed a model of the quiet time Jicamarca drifts using Bernstein polynomials as base functions. We have examined in detail, for the first time, the response of enhanced geomagnetic activity on the Jicamarca zonal drifts. Our results indicate that, surprisingly, geomagnetic activity effects on the zonal drifts are strongly season and solar flux dependent. The Jicamarca zonal disturbance drifts occur predominantly at night, are largest during equinox and December solstice high solar flux conditions, and are essentially negligible near June solstice. The daytime perturbations are eastward, have small magnitudes, and occur about 3–15 hours after enhanced geomagnetic activity. The nighttime disturbance drifts are westward with peak values at about 0300–0400 during low solar flux periods. They increase with solar flux and reach largest values near midnight high solar flux periods. The longer term (delays of 15–24 hours) disturbance drifts have peak values near dawn and are also largest for high solar flux conditions. The Jicamarca zonal disturbance drifts are in good agreement with the predictions of the disturbance dynamo theory, particularly for low solar flux conditions. Our results indicate a strong dependence of the equatorial nighttime zonal drifts on ionospheric conductivity, dependent F -region dynamo electric fields. They also suggest that the longitudinal variation of both quiet and disturbed nighttime zonal drifts is season-dependent. We have combined our quiet and disturbed zonal drift results to develop a season, solar flux and magnetic activity dependent model for these drifts which will be available to the community.

Appendix A: Mathematical Formulation

[29] In this section we first describe briefly the basic properties of Bernstein polynomials [e.g., Lorentz, 1986] and outline the derivation of our empirical models, which use these polynomials as base functions.

[30] Bernstein polynomials are named after Sergei Natanovich Bernstein, a Ukrainian mathematician who used them for a proof of the Weierstrass approximation theorem [Bernstein, 1912]. These polynomials can represent monotone, piecewise smooth functions having left and right derivatives at every point, without the occurrence of the Gibbs phenomenon [e.g., Gyz and Palacios, 2003]. Bernstein polynomials provide global approximations of data point clouds in contrast to local approximations given by other more popular methods and are particularly useful for modeling incomplete and noisy data sets, which are not well suited for local approximations. They are used for compact representations of curves and surfaces in computer aided

design [e.g., Farin, 2002] due to their monotonicity and shape-preserving properties.

[31] Our methodology closely follows the general least squares method outlined by Press *et al.* [1996]. Let us assume $m + 1$ measured data points $\mathbf{p}_i = (t_i, y_i)$ with $i = 0, 1, \dots, m$ with each dependent variable associated with an error described by a standard deviation σ_i (assumed to have a Gaussian distribution). The general form of our model is given by

$$y(t) = \sum_{k=0}^n a_k B_{k,n}(t), \quad (1)$$

where a_k are the coefficients of the model and $B_{0,n}, \dots, B_{n,n}$ is a Bernstein basis of degree n defined as

$$B_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}. \quad (2)$$

These polynomials have the following basic properties:

- [32] 1. They are positive semidefinite for $0 \leq t \leq 1$.
- [33] 2. The $n + 1$ polynomials of degree n provide a partition of the unity, i.e., their sum is one for all values of t . Therefore for every t and for $\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_n$ vectors of $\mathbb{R}^n$, then

$$\mathbf{p}(t) = \mathbf{p}_0 B_{0,n}(t) + \mathbf{p}_1 B_{1,n}(t) + \dots + \mathbf{p}_n B_{n,n}(t) \quad (3)$$

is an affine combination of these vectors, and for $0 \leq t \leq 1$, this combination is convex. Since this is an important property, from now on we will be dealing with a normalized independent variable $t \in [0, 1]$.

- [34] 3. The Bernstein polynomials of degree n form a basis for the space of degree smaller or equal to n .

- [35] 4. These polynomials obey a useful recurrence relation that is important for computer applications:

$$B_{k,n+1}(t) + 1 = (1-t)B_{k,n}(t) + tB_{k-1,n}(t). \quad (4)$$

In order to solve our problem, we first define the chi-square merit function:

$$\chi^2 = \sum_{i=0}^m \left[\frac{1}{\sigma_i} \left(y_i - \sum_{k=0}^n a_k B_{k,n}(t_i) \right) \right]^2. \quad (5)$$

Minimizing χ^2 with respect to the $n + 1$ coefficients, a_k

$$0 = \sum_{i=0}^m \frac{1}{\sigma_i^2} \left[y_i - \sum_{j=0}^n a_j B_{j,n}(t_i) \right] B_{k,n}(t_i), \quad (6)$$

where $k = 0, 1, \dots, n$. Now, we define the so-called design matrix $\mathbf{A}$, whose $(m + 1) \times (n + 1)$ components are functions of the Bernstein basis such that $A_{ij} = B_{j,n}(t_i)/\sigma_i$, and vector $\mathbf{b}$ of length $m + 1$ with components $b_i = y_i/\sigma_i$. Then, interchanging the summation order, equation (6) can be written

$$\sum_{j=0}^n \alpha_{kj} a_j = \beta_k, \quad (7)$$

where

$$\alpha_{kj} = \sum_{i=0}^m \frac{1}{\sigma_i^2} B_{j,n}(t_i) B_{k,n}(t_i) \text{ or } [\alpha] = \mathbf{A}^T \cdot \mathbf{b} \quad (8)$$

is a $(n + 1) \times (n + 1)$ square matrix, and

$$\beta_k = \sum_{i=0}^m \frac{1}{\sigma_i^2} y_i B_{k,n}(t_i) \text{ or } [\beta] = \mathbf{A}^T \cdot \mathbf{b} \quad (9)$$

is a vector of length $n + 1$. The normal equation (8) can be solved using, for example, LU decomposition [e.g., Press *et al.*, 1996]. In matrix notation we have

$$[\alpha] \cdot \mathbf{a} = [\beta] \text{ or } (\mathbf{A}^T \cdot \mathbf{A}) \mathbf{a} = \mathbf{A}^T \cdot \mathbf{b}. \quad (10)$$

[36] The inverse matrix $c_{jk} = [a_{jk}]^{-1}$ is related to the uncertainties of the estimated parameters $\mathbf{a}$. The diagonal elements c_{jj} of $[\mathbf{c}]$ are the variances of the fitted parameters $\mathbf{a}$ and the off-diagonal elements c_{jk} are the covariances between a_j and a_k [Press *et al.*, 1996].

A1. Smoothing

[37] The method described above minimizes the χ^2 function without accounting for the shape of the resulting curve. We can smooth the resulting curves assuming that the second order differences $\Delta^2 a_i = a_{i+2} - 2a_{i+1} + a_i = 0$. This condition defines a $(n - 1) \times (n + 1)$ matrix $\mathbf{S}$. Then, we solve

$$\mathbf{N} = \begin{pmatrix} (1-\omega)\mathbf{A} \\ \omega\mathbf{S} \end{pmatrix} \cdot \mathbf{a} = \begin{pmatrix} (1-\omega)\mathbf{b} \\ \mathbf{0} \end{pmatrix} = \mathbf{Q}, \quad (11)$$

where $0 \leq \omega \leq 1$ is our smoothing parameter. If $\omega = 0$, we get (10) and there is no smoothing. Solving (11), we get

$$\mathbf{a} = (\mathbf{N}^T \cdot \mathbf{N})^{-1} \cdot \mathbf{N}^T \cdot \mathbf{Q}, \quad (12)$$

which determines our $n + 1$ coefficients.

A2. Derivation of the Models

[38] Our quiet time and disturbed models were developed in two steps. In the quiet case, first, for each season and hour of local time, we have used second degree Bernstein polynomials and $\sigma = 1$, $\omega = 0.4$, to model the dependence of the drifts on the 10.7 cm solar flux index. This first fit also provides the standard deviations (σ_n). Then, this solar flux model was used to calculate the drifts for 48 evenly spaced time intervals and for eight values of the solar flux index in increments of 25 units, between 66 and 241. The drift values for a given solar flux index were determined by linear interpolation between the two closest calculated values. In the final step, these data were modeled with 11th degree Bernstein polynomials, and $\sigma = \sigma_m$, $\omega = 0.0005$. Therefore in the quiet model, the matrix $\mathbf{N}$ has 58×12 elements ($m = 48$, $n = 11$).

[39] In the development of the disturbance drift models, we had initially to account for solar flux effects. Therefore the residual drifts (i.e., measured values minus the corresponding season and solar cycle-dependent average values) were placed in two slightly overlapping $F_{10.7}$ index bins, $F_{10.7} \leq 140$, and $F_{10.7} > 120$. We have combined the

perturbation drifts from equinox and December solstice to improve our statistics and did not examine magnetic activity effects during June solstice, since they appear to be largely negligible. The low and high solar flux perturbation drifts were modeled in half an hour bins as linear functions of time averaged K_p indices. The local time variations of the disturbance drifts were also modeled with 11th degree Bernstein polynomials and $\omega = 0.0005$.

[40] Overall, the quiet time and disturbance drift models which estimate average values and their variability have 648 and 336 coefficients, respectively. The model inputs are day of the year (Julian day) the $F_{10.7}$ cm index and the K_p over the previous 24 hours. The model drift for a given day are given by the sum of the quiet and disturbance drift components. The models will be available through the Jicamarca website.

[41] **Acknowledgments.** We thank the Jicamarca staff for the radar measurements. The work at Utah State University was supported by the Aeronomy Program, Division of Atmospheric Sciences of the National Science Foundation through grant ATM-0004380. Jonas R. Souza was supported by CAPES-Brazil under grant BEX2521/02-4. The work of A. S. Santos was performed at the University of Uberlandia.

[42] Arthur Richmond thanks the reviewers for their assistance in evaluating this paper.

References

- Basu, S., et al. (2001), Response of the equatorial ionosphere in the South Atlantic region to the magnetic storm of July 15, 2001, *Geophys. Res. Lett.*, **18**, 3577–3580.
- Bernstein, S. N. (1912), Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités, *Comm. Soc. Math. Kharkov*, **13**, 1–2.
- Blanc, M., and A. D. Richmond (1980), The ionospheric disturbance dynamo, *J. Geophys. Res.*, **85**, 1669–1686.
- Coley, W. R., and R. A. Heelis (1989), Low latitude zonal and vertical ion drifts seen by DE 2, *J. Geophys. Res.*, **94**, 6751–6761.
- Farin, G. (2002), *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide*, Morgan Kaufmann Ser. in Comput. Graphics and Geometric Modeling, 5th ed., Elsevier, New York.
- Fejer, B. G. (1997), The electrodynamics of the low latitude ionosphere: Recent results and future challenges, *J. Atmos. Sol. Terr. Phys.*, **59**, 1456–1482.
- Fejer, B. G., and J. T. Emmert (2003), Low-latitude ionospheric disturbance electric field effects during the recovery phase of the 19–21 October 1998 magnetic storm, *J. Geophys. Res.*, **108**(A12), 1454, doi:10.1029/2003JA010190.
- Fejer, B. G., and L. Scherliess (1997), Empirical models of storm-time equatorial zonal electric fields, *J. Geophys. Res.*, **102**, 24,047–24,056.
- Fejer, B. G., E. Kudeki, and D. T. Farley (1985), Equatorial F region plasma drifts, *J. Geophys. Res.*, **90**, 12,249–12,255.
- Fejer, B. G., R. W. Spiro, R. A. Wolf, and J. C. Foster (1990), Latitudinal variation of perturbation electric fields during magnetically disturbed periods; 1986 SUNDIAL observation and model results, *Ann. Geophys.*, **8**, 441–454.
- Fejer, B. G., E. R. de Paula, S. A. Gonzales, and R. F. Woodman (1991), Average vertical and zonal F region plasma drifts over Jicamarca, *J. Geophys. Res.*, **96**, 13,901–13,906.
- Gyz, H., and J. L. Palacios (2003), On the approximation properties of Bernstein polynomials via probabilistic tools, *Boletín Assoc. Math. Venezolana*, **1**.
- Haerendel, G., J. V. Eccles, and S. Cakir (1992), Theory for modeling the equatorial evening ionosphere and the origin of the shear in the horizontal plasma flow, *J. Geophys. Res.*, **97**, 1209–1223.
- Herrero, F. A., H. G. Mayr, N. W. Spencer, A. E. Hedin, and B. G. Fejer (1985), Interaction of zonal winds with the magnetic pressure bulge in the Earth's thermosphere: Empirical check of momentum balance, *Geophys. Res. Lett.*, **12**, 491–494.
- Huang, C.-M., A. D. Richmond, and M.-Q. Chen (2005), Theoretical effects of geomagnetic activity on low-latitude ionospheric electric fields, *J. Geophys. Res.*, **110**, A05312, doi:10.1029/2004JA010994.
- Immel, T. J., H. U. Frey, S. B. Mende, and E. Sagawa (2004a), Global observations of the zonal drift of equatorial plasma bubbles, *Global observations of the zonal drift speed of equatorial ionospheric plasma bubbles*, *Ann. Geophys.*, **22**, 3099–3107.
- Immel, T. J., E. Sagawa, S. B. Mende, and J. Patel (2004b), The climatology of low-latitude ionospheric densities and zonal drifts from IMAGE-FUV, *Eos Trans. AGU*, **85**(47), Fall Meeting Suppl., Abstract SA31B-03.
- Kelley, M. C., B. G. Fejer, and C. A. Gonzales (1979), An explanation for anomalous electric fields associated with a northward turning of the interplanetary magnetic field, *Geophys. Res. Lett.*, **6**, 301–304.
- Kudeki, E., and S. Bhattacharyya (1999), Postsunset vortex in equatorial F -region plasma drifts and implications for bottomside spread-F, *J. Geophys. Res.*, **104**, 28,163–28,170.
- Kudeki, E., S. Bhattacharyya, and R. F. Woodman (1999), A new approach in incoherent scatter F region $E \times B$ drift measurements at Jicamarca, *J. Geophys. Res.*, **104**, 28,145–28,162.
- Kudeki, E., B. G. Fejer, D. T. Farley, and H. M. Ierick (1981), Interferometer studies of equatorial F region irregularities and drifts, *Geophys. Res. Lett.*, **8**, 377–380.
- Lorentz, G. G. (1986), *Bernstein Polynomials*, Chelsea, New York.
- Martinis, C., J. V. Eccles, J. Baumgardner, J. Manzano, and M. Mendillo (2003), Latitude dependence of zonal plasma drifts obtained from dual-site airglow observations, *J. Geophys. Res.*, **108**(A3), 1129, doi:10.1029/2002JA009462.
- Peymirat, C., A. D. Richmond, and R. G. Koba (2000), Electrodynamical coupling of high and low latitudes: Simulations of shielding/overshielding effects, *J. Geophys. Res.*, **105**, 22,991–23,003.
- Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling (1996), *Numerical Recipes in Pascal*, Cambridge Univ. Press., New York.
- Richmond, A. D. (1995), Ionospheric electrodynamics, in *Handbook of Atmospheric Electrodynamics*, 2, edited by H. Volland, pp. 249–290, CRC Press, Boca Raton, Fla.
- Rishbeth, H. (1971), Polarization fields in the equatorial F region, *Planet Space Sci.*, **19**, 357–369.
- Scherliess, L., and B. G. Fejer (1997), Storm time dependence of equatorial disturbance dynamo zonal electric fields, *J. Geophys. Res.*, **102**, 24,037–24,046.
- Scherliess, L., and B. G. Fejer (1999), Radar and satellite global equatorial F region vertical drift model, *J. Geophys. Res.*, **104**, 6829–6842.
- Senior, C., and M. Blanc (1984), Convection in the inner magnetosphere: Model predictions and data, *J. Geophys. Res.*, **89**, 261–284.
- Sheehan, R. E., and C. E. Valladares (2004), Equatorial ionospheric zonal drift model and vertical drift statistics from UHF scintillation measurements in South America, *Ann. Geophys.*, **22**, 3177–3193.
- Sobral, J. H. A., and M. A. Abdu (1991), Solar activity effects on equatorial plasma bubble zonal velocities as measured by airglow scanning photometers, *J. Atmos. Sol. Terr. Phys.*, **53**, 729–742.
- Sobral, J. H. A., M. A. Abdu, H. Takahashi, H. Sawant, C. J. Zamlutti, and G. L. Borba (1999), Solar and geomagnetic effects on nocturnal zonal velocities of ionospheric plasma depletions, *Adv. Space Res.*, **24**(11), 1507–1510.
- Spiro, R. W., R. A. Wolf, and B. G. Fejer (1988), Penetration of high-latitude electric field effects to low latitudes during SUNDIAL 1984, *Ann. Geophys.*, **6**, 39–50.
- Terra, P. M., J. H. A. Sobral, M. A. Abdu, and H. Takahashi (2004), Plasma bubble zonal velocity variations with solar activity in the Brazilian sector, *Ann. Geophys.*, **22**, 3123–3128.
- Tsunomura, S. (1999), Numerical analysis of global ionospheric current system including the effect of equatorial enhancement, *Ann. Geophys.*, **17**, 692–706.
- Valladares, C. E., R. A. Sheehan, S. Basu, H. Kuenzler, and J. Espinoza (1996), The multi-instrumented studies of equatorial thermosphere aeronomy scintillation system: Climatology of zonal drifts, *J. Geophys. Res.*, **101**, 26,839–26,850.
- Woodman, R. F. (1970), Vertical drift velocities and east-west electric fields at the magnetic equator, *J. Geophys. Res.*, **75**, 6249–6259.
- Woodman, R. F. (1972), East-west ionospheric drifts at the magnetic equator, *Space Res.*, **12**, 969–974.
- A. E. Costa Pereira, Department of Electrical Engineering, Universidade Federal de Uberlandia, 38400-902, Uberlandia, Minas Gerais, Brazil.
- B. G. Fejer, Center for Atmospheric and Space Sciences, Utah State University, 4405 Old Main Hall, Logan, UT 84322-4405, USA. (bfejer@cc.usu.edu)
- A. S. Santos, Department of Computer Science, Federal University of Goiás, 75705-220, Catalão, Goiás, Brazil.
- J. R. Souza, Instituto de Pesquisas Espaciais (INPE), CP 515, 12245-907, Sao José dos Campos, São Paulo, Brazil.

Apêndice B - Código para a curva de Bézier com mínimos quadrados

Enunciado do problema:

Suponha-se que sejam dados $p + 1$ pontos $\mathbf{p}_i = (h_i, v_i)$ com $i = 0, \dots, p$ resultantes de medições, cada um com erro associado dado por σ_i e parametrizado por t_i . Deseja-se encontrar um polinômio $\mathbf{x}(t)$ de Bernstein, tal que as distâncias $\|\mathbf{p}_i - \mathbf{x}(t_i)\|$ sejam mínimas.

A forma geral deste tipo de modelo é

$$\mathbf{x}(t) = \sum_{k=0}^n \mathbf{a}_k B_{k,n}(t)$$

em que $B_{0,n}, \dots, B_{n,n}$ são funções da base de polinômios de Bernstein de grau n .

1. Polinômios de Bernstein

Os polinômios de Bernstein de grau n são definidos por

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

para $i = 0, 1, \dots, n$, em que

$$\binom{i}{n} = \frac{n!}{i! \times (n-i)!} \quad (\text{B.1})$$

O primeiro passo é calcular os coeficientes binomiais. Para fazer isto, pode-se usar a equação B.1. Entretanto os fatoriais tornam-se muito grandes, mesmo para números relativamente pequenos. Então, uma melhor estratégia é usar o triângulo de Pascal para realizar a tarefa.

```

let pascal np =
let bin = Array.create (np + 1) 0.0 in
  bin.(0) ← 1.0;
  for j = 1 to np do
    for i = j downto 1 do
      bin.(i) ← bin.(i) + . bin.(i - 1)
    done
  done;
bin

```

```
let mq = ref 3
```

```
let bin = ref (pascal !mq)
```

A função *bernstein* (*i*, *n*) *t* calcula $B_{i,n}(t)$.

```

let bernstein (i, n) t =
  if (Array.length !bin) ≠ (n + 1)
  then bin := pascal n;

```

```

let bin = !bin.(i)

and i = float_of_int i

and n = float_of_int n in

  bin *. t ** i *. (1.0 -. t) ** (n -. i)

```

2. Parametrização

Dada a importância da propriedade 2 dos polinômios de Bernstein, vista na subseção 5.1.1, impõe-se que as abscissas variem entre 0 e 1. Uma forma de alcançar isto é parametrizando h_i por meio de t_i . Pode-se criar estes parâmetros com as seguintes especificidades:

1. $0 \leq t_i \leq 1$ sempre que $h_0 \leq h \leq h_p$
2. $t_0 < t_1 < \dots < t_p$

Para esse fim, faremos:

$$\begin{aligned}
 t_0 &= 0 \\
 t_i &= t_{i-1} + \frac{\delta_i}{\Delta}, \quad i = 1, 2, \dots, p
 \end{aligned}$$

em que

$\delta_i = |\mathbf{p}_i - \mathbf{p}_{i-1}|$ é comprimento da corda entre dois pontos consecutivos e

$\Delta = \sum_{i=1}^p \delta_i$ é o comprimento total da corda.

```

type datum = {mutable h :float; mutable v :float; mutable sigma :float}

```

Cria-se um vetor para receber os dados

```

let pt = ref (Array.create 1 {h = 0.0; v = 0.0; sigma = 0.0})

```

```

and p = ref 0 (* p = número de pontos de dados. *)

```

```

and t = ref (Array.create 1 0.0) (* t é um vetor de parâmetros *)

```

```

let initialize data =

  let dataLength = List.length data in

  let delta = Array.create dataLength 0.0

  and d = ref 0.0 in

  let rec fill i s =

    match s with

    [] → ()

    | ((v, h, sigma) :: xs) → (!pt.(i) ← {h = h; v = v; sigma = sigma}; fill (i+
1) xs)

  in

  p := dataLength - 1;

  pt := Array.create dataLength {h = 0.0; v = 0.0; sigma = 0.0};

  t := Array.create dataLength 0.0;


  fill 0 data;

  for i = 1 to !p do

    let h = !pt.(i).h - . !pt.(i - 1).h

    and v = !pt.(i).v - . !pt.(i - 1).v in

      delta.(i) ← sqrt(h *. h + . v *. v);

      d := !d + . delta.(i)

  done;

  !t.(0) ← 0.0;

  for i = 1 to !p do

    !t.(i) ← !t.(i - 1) + . delta.(i) /. !d

  done

```

3. Ajuste pelo método dos mínimos quadrados

Para resolver o problema posto, define-se a seguinte função

$$\chi^2 = \sum_{i=0}^p \left((h_i, \frac{v_i}{\sigma_i}) - \sum_{k=0}^n (a_{h_k}, \frac{a_{v_k}}{\sigma_i}) B_{k,n}(t_i) \right)^2 \quad (\text{B.2})$$

Para encontrar este mínimo desta equação, usam-se as equações normais. Antes, porém, mais algumas definições.

Seja **A** uma matriz, chamada *matriz de projeto*, cujos $(p+1) \times (n+1)$ elementos são constituídos das $n+1$ funções da base de Bernstein, avaliadas nos $p+1$ parâmetros t_i com seus respectivos $p+1$ erros σ_i , tal que:

$$A_{ij} = \frac{B_{j,n}(t_i)}{\sigma_i}$$

```
open Bigarray
```

```
let makeAsig n =
```

```
  let mat = Array2.create float64 c_layout (!p + 1) (n + 1) in
```

```
    for i = 0 to !p do
```

```
      for j = 0 to n do
```

```
        mat.{i,j} ← (bernstein (j, n) !t.(i)) /. !pt.(i).sigma
```

```
      done
```

```
    done;
```

```
  mat
```

```
let makeA n =
```

```
  let mat = Array2.create float64 c_layout (!p + 1) (n + 1) in
```

```

for  $i = 0$  to  $!p$  do
    for  $j = 0$  to  $n$  do
         $mat.\{i,j\} \leftarrow (bernstein(j, n) !t.(i))$ 
    done
done;
 $mat$ 

```

Define-se um vetor $\mathbf{B}$ de comprimento $p + 1$ por

$$\mathbf{b}_i = \frac{\mathbf{p}_i}{\sigma_i} \quad (\text{B.3})$$

O código para este vetor segue abaixo.

```

let  $makeB()$  =
    let  $vecH = Array1.create float64 c\_layout (!p + 1)$ 
    and  $vecV = Array1.create float64 c\_layout (!p + 1)$  in
        for  $i = 0$  to  $!p$  do
             $vecH.\{i\} \leftarrow !pt.(i).h; (* /. !pt.(i).sigma ; *)$ 
             $vecV.\{i\} \leftarrow !pt.(i).v /. !pt.(i).sigma$ 
        done;
    ( $vecH, vecV$ )

```

Denote um vetor $\mathbf{a}$ com seus $n + 1$ elementos dados pelos parâmetros a serem ajustados, $\mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_n$.

4. Suavização da curva de ajuste

Se se juntarem as equações de suavização do capítulo 5 ao sistema original $\mathbf{A} \cdot \mathbf{a} = \mathbf{B}$,

obter-se-á

$$\begin{pmatrix} \mathbf{A} \\ \mathbf{S} \end{pmatrix} \mathbf{a} = \begin{pmatrix} \mathbf{b} \\ \mathbf{0} \end{pmatrix} \quad (\text{B.4})$$

Este sistema é resolvido formando um sistema linear simétrico de equações normais.

Para aumentar o controle sobre o efeito das equações de suavização, ponderam-se os dois diferentes componentes por ω :

$$\underbrace{\begin{pmatrix} (1-\omega)\mathbf{A} \\ \omega\mathbf{S} \end{pmatrix}}_{\mathbf{N}} \mathbf{a} = \underbrace{\begin{pmatrix} (1-\omega)\mathbf{b} \\ \mathbf{0} \end{pmatrix}}_{\mathbf{Q}} \quad (\text{B.5})$$

Para completar o algoritmo, vai-se resolver a equação (B.5):

$$\begin{aligned} \mathbf{N} \cdot \mathbf{a} &= \mathbf{Q} \\ \mathbf{N}^T \cdot \mathbf{N} \cdot \mathbf{a} &= \mathbf{N}^T \cdot \mathbf{Q} \\ (\mathbf{N}^T \cdot \mathbf{N})^{-1} \cdot (\mathbf{N}^T \cdot \mathbf{N}) \cdot \mathbf{a} &= (\mathbf{N}^T \cdot \mathbf{N})^{-1} \cdot \mathbf{N}^T \cdot \mathbf{Q} \\ \mathbf{a} &= (\mathbf{N}^T \cdot \mathbf{N})^{-1} \cdot \mathbf{N}^T \cdot \mathbf{Q} \end{aligned}$$

A codificação da matriz $\mathbf{N}$, em conformidade com a equação (B.5) é feita pelo código a seguir.

```
open Matrix
```

```
let makeN n omega =
```

```
    let matN = Array2.create float64 c_layout (!p + n) (n + 1)
```

```
    and matA = makeA n
```

and $k = \text{ref } 0$

in

Array2.fill matN 0.0; (* preenche a matriz $\mathbf{N}$ com zeros *)

multE (1.0 - . omega) matA; (* $(1 - \omega)\mathbf{A}$ *)

(* copia $\mathbf{A}$ nos primeiros elementos de $\mathbf{N}$ *)

for $i = 0$ to $!p$ do

for $j = 0$ to n do

$\text{matN}\{i, j\} \leftarrow \text{matA}\{i, j\}$

done;

done;

(* completa $\mathbf{N}$ com (ωS) *)

for $i = !p + 1$ to $!p + n - 1$ do

$\text{matN}\{i, !k\} \leftarrow \text{omega} * 1.0;$

$\text{matN}\{i, !k + 1\} \leftarrow \text{omega} * - (2.0);$

$\text{matN}\{i, !k + 2\} \leftarrow \text{omega} * 1.0;$

$k := !k + 1$

done;

matN

let *makeNsig n omega* =

let *matN* = *Array2.create float64 c_layout (!p + n) (n + 1)*

and *matA* = *makeAsig n*

and $k = \text{ref } 0$

in

Array2.fill matN 0.0; (* preenche a matriz $\mathbf{N}$ com zeros *)

```

multE (1.0 - . omega) matA;                                (* (1 -  $\omega$ )A *)
                                                                (* copia A em nos primeiros elementos de N *)

for i = 0 to !p do
    for j = 0 to n do
        matN.{i,j} ← matA.{i,j}
    done;
done;

                                                                (* completa N com ( $\omega S$ ) *)

for i = !p + 1 to !p + n - 1 do
    matN.{i, !k} ← omega * . 1.0;
    matN.{i, !k + 1} ← omega * . -(2.0);
    matN.{i, !k + 2} ← omega * . 1.0;
    k := !k + 1
done;

matN

```

Agora, implementa-se a matriz **Q**.

```

let makeQ n omega =

    let (bH, bV) = makeB ()

    and vecH = Array1.create float64 c_layout (!p + n)

    and vecV = Array1.create float64 c_layout (!p + n)

    and weight = 1.0 - . omega in

    Array1.fill vecH 0.0;

    Array1.fill vecV 0.0;

    for i = 0 to !p do                                (* Preenche a primeira parte com (1 -  $\omega$ )B *)

```

```

     $vecH.\{i\} \leftarrow weight * .bH.\{i\} ;$ 

     $vecV.\{i\} \leftarrow weight * .bV.\{i\}$ 

done;

( $vecH, vecV$ )

```

Agora, calculam-se os produtos $\mathbf{N}^T\mathbf{N}$ e $\mathbf{N}^T\mathbf{Q}$.

```

let  $nEquations$   $n\ omega$  =

    let  $matN$  =  $makeN$   $n\ omega$ 

    and  $matNsig$  =  $makeNsig$   $n\ omega$  in

    let  $matNT$  =  $transpose$   $matN$ 

    and  $matNTsig$  =  $transpose$   $matNsig$  in

    let  $matNTN$  =  $mult$   $matNT$   $matN$ 

    and  $matNTNsig$  =  $mult$   $matNTsig$   $matNsig$ 

    and ( $qH, qV$ ) =  $makeQ$   $n\ omega$  in

    let  $nTqH$  =  $multV$   $matNT$   $qH$ 

    and  $nTqV$  =  $multV$   $matNTsig$   $qV$  in

    ( $matNTN, matNTNsig, nTqH, nTqV$ )

```

Usando a decomposição LU para resolver o sistema, tem-se

```

open  $Lu$ 

```

```

let  $findCoefs$   $n\ omega$  =

    let ( $matNTN, matNTNsig, matQH, matQV$ ) =  $nEquations$   $n\ omega$ 

    and  $indx$  =  $Array1.create$   $int$   $c\_layout$  ( $n + 1$ ) (* Aloca espaço para o vetor  $indx$  *)

    and  $d$  =  $ref$  0 in

```

Aplicando a decomposição LU a $\mathbf{N}^T\mathbf{N}$

```
d := lu_dcmp matNTN (n + 1) indx;
```

Aplicando a substituição regressiva para encontrar-se-á os coeficientes a_h . Estes coeficientes ficarão em *matQH*.

```
lu_bksb matNTN (n + 1) indx matQH;
```

Aplicando a decomposição LU a $\mathbf{N}^T\mathbf{N}$

```
d := lu_dcmp matNTNsig (n + 1) indx;
```

Aplicando a substituição regressiva para encontrar os coeficientes a_v . Estes coeficientes estão em *matQV*.

```
lu_bksb matNTNsig (n + 1) indx matQV;
```

```
(matQH, matQV)
```

Para um exemplo de uso deste módulo, utilize-se a função *test*, como na última linha deste apêndice.

```
let teste n omega =  
  let (a_h, a_v) = findCoefs n omega  
  and matA = makeA n  
  and matAsig = makeAsig n in  
  let h = multV matA a_h  
  and v = multV matA a_v  
  and s = ref [] in
```

```
print_newline(); (* Imprime H e V *)

print_string "H░░░░░░░░░░He░░░░░░░░░░V░░░░░░░░░░Ve░░░░░░░░░░Erro\n";

for i = 0 to !p do

    print_newline();

    print_float !pt.(i).h ; print_string "\t"; print_float h.{i}; print_string "\t";

    print_float !pt.(i).v ; print_string "\t"; print_float v.{i}; print_string "\t";

    print_float !pt.(i).sigma

done;

print_newline(); (* Fim imprime H e V *)

for i = 0 to !p do

    s := ((v.{i}, h.{i}) :: s.contents);

done;

List.rev s.contents

if !Sys.interactive then () else begin teste 6 0.1 ; exit 0 end;;
```

Apêndice C - Código para a superfície de Bézier

O código mostrado na próxima seção implementa as superfícies de Bézier para o caso de duas variáveis de entrada.

C.1 Bernstein3D.ml

```
(* Parametrização 2D – Método do comprimento da corda *)
open Bigarray;;

open Matrix;;
open Data;;
open Lu;;

let norm (x1,y1,z1) (x2,y2,z2) =
  let x = x2 -. x1
  and y = y2 -. y1
  and z = z2 -. z1 in
  sqrt(x*x +. y*y +. z*z)
;;

let param2D d =
  let m = d.m (* número de linhas *)
  and n = d.n (* número de colunas *)
  let u = Array2.create float64 c_layout (m+1) (n+1)
  and v = Array2.create float64 c_layout (m+1) (n+1)
  and l = ref 0.0 in
```

```

(* Preenche com zeros os vetores e matrizes de parâmetros *)
Array1.fill d.s 0.0;
Array1.fill d.t 0.0;
Array2.fill u 0.0;
Array2.fill v 0.0;

(* Calcula s0, ..., sm *)
for j = 0 to n do (* coluna j *)
  (* Calcula um conjunto de m+1 parâmetros
    u0j, u1j, ..., umj *)
    l := 0.0;
    (* Método do comprimento da corda *)
    for i = 1 to m do
      l := !l +. norm (d.x.{i,j}, d.y.{i,j}, d.z.{i,j})
                      (d.x.{i-1,j}, d.y.{i-1,j}, d.z.{i-1,j});
      u.{i,j} <- !l
    done;
    for i = 1 to m do
      u.{i,j} <- u.{i,j} /. !l
    done;
  done;
  for i = 0 to m do
    for j = 0 to n do
      d.s.{i} <- d.s.{i} +. u.{i,j}
    done;
    d.s.{i} <- d.s.{i} /. (float_of_int n +. 1.0)
    (* si = (ui0 + ui1 + ... + uin)/(n+1) *)
  done; (* Os parâmetros para a direção u estão disponíveis *)

(* Calcula t0, ..., tn *)
for i = 0 to m do (* linha i *)
  (* Calcula um conjunto de n+1 parâmetros
    vi0, vi1, ..., vin *)
    l := 0.0;
    (* Método do comprimento da corda *)
    for j = 1 to n do
      l := !l +. norm (d.x.{i,j}, d.y.{i,j}, d.z.{i,j})
                      (d.x.{i,j-1}, d.y.{i,j-1}, d.z.{i,j-1});
      v.{i,j} <- !l
    done;
    for j = 1 to n do
      v.{i,j} <- v.{i,j} /. !l
    done;
  done;
  for j = 0 to n do
    for i = 0 to m do
      d.t.{j} <- d.t.{j} +. v.{i,j}
    done;
  done;

```

```

    d.t.{j} <- d.t.{j} /. (float_of_int m +. 1.0)
    (* tj = (v0j + v1j + ... + vnj)/(m+1) *)
done; (* Os parâmetros para a direção v estão disponíveis *)
;;

(* 3D Bézier curve approximation *)
let rec bernstein(i, n) u =
  if i < 0 then 0.0 else
  if n = 0 then 1.0
  else (1.0 -. u) *. (bernstein(i,n-1) u) +. u *. bernstein(i-1,n-1) u
;;

(* cria a matriz A, onde
   q é o grau do polinômio de Bernstein
   t é o vetor de parâmetros
*)
let bernsteinMatrix q t =
  let n = Array1.dim t - 1 in
  let mat = Array2.create float64 c_layout (n + 1) (q + 1) in
  for i = 0 to n do
    for j = 0 to q do
      mat.{i,j} <- bernstein (j, q) t.{i}
    done
  done;
  mat
;;

let bernsteinMatrixSigma n t sigma =
  let m = Array1.dim t - 1 in
  let mat = Array2.create float64 c_layout (m + 1) (n + 1) in
  for i = 0 to m do
    for j = 0 to n do
      mat.{i,j} <- (bernstein (j, n) t.{i}) /. sigma.{i}
    done
  done;
  mat
;;

(* Cria a matriz 
$$N = \begin{vmatrix} (1-w)A & \\ & wS \end{vmatrix}$$

   Recebe:
   q é o grau do polinômio
   omega é o peso que multiplicará as matrizes
   a é a matriz A
*)

```

```

let makeN q omega a =
  let m = Array2.dim1 a - 1
  and n = Array2.dim2 a - 1 in
  let matN = Array2.create float64 c_layout (m + q) (n + 1)
  and k = ref 0
  in
    Array2.fill matN 0.0; (*r preenche a matriz  $\mathbf{N}$  com zeros *)
    multE (1.0 -. omega) a; (*r  $(1-\omega) \mathbf{A}$  *)
    (*r copia  $\mathbf{A}$  em nos primeiros elementos de  $\mathbf{N}$  *)
    for i = 0 to m do
      for j = 0 to n do
        matN.{i,j} <- a.{i,j}
      done;
    done;
    (*r completa N com  $(\omega S)$  *)
    for i = m + 1 to m + q - 1 do
      matN.{i, !k} <- omega *. 1.0;
      matN.{i, !k + 1} <- omega *. -(2.0);
      matN.{i, !k + 2} <- omega *. 1.0;
      k := !k + 1
    done;

    matN

;;

(* Cria a matriz  $Q = \begin{vmatrix} (1-w)B \\ 0 \end{vmatrix}$  *)

Recebe:
  q é o grau do polinômio
  omega é o peso que multiplicará as matrizes
  b é a matriz B

*)
let makeQ q omega b =
  let m = Array2.dim1 b - 1
  and n = Array2.dim2 b - 1 in
  let matQ = Array2.create float64 c_layout (m + 1) (n + q)
  and weight = 1.0 -. omega in
    Array2.fill matQ 0.0;
    for i = 0 to m do
      for j = 0 to n do
        (*r Fill the first part with  $(1-\omega) \mathbf{B}$  *)
        matQ.{i,j} <- weight *. b.{i,j}
      done;
    done;

    matQ

;;

```

```

let makeQv q omega b =
  let m    = Array2.dim1 b - 1
  and n    = Array2.dim2 b - 1 in
  let matQ = Array2.create float64 c_layout (m + q) (n + 1)
  and weight = 1.0 -. omega in
  Array2.fill matQ 0.0;
  for i = 0 to m do
    for j = 0 to n do
      (*r Fill the first part with  $(1 - \omega) \{ \bf B \}$  *)
      matQ.{i,j} <- weight *. b.{i,j}
    done;
  done;

  matQ
;;

```

(*

Input: $(m+1) \times (n+1)$ data points D_{ij}

degree (p, q)

Output: $(p+1) \times (q+1)$ matrix P_{ij} of control points from Bézier

surface of degree (p, q) that approximates the data points;

Algorithm:

Compute parameters in the u-direction $s_0, s_1, \dots, s_m$;

Compute parameters in the v-direction $t_0, t_1, \dots, t_n$;

for $c := 0$ to m do / for row c of D 's */*

begin / computing "intermediate data points" Q 's */*

Apply curve approximation to row c of data points

(i.e., $D_{c0}, D_{c1}, \dots, D_{cn}$) using

** degree q*

** parameters $t_0, t_1, \dots, t_n$*

The result is row c of the "intermediate data points"

$Q_{c0}, Q_{c1}, \dots, Q_{cq}$

/ Q 's form a $(m+1) \times (q+1)$ matrix */*

end

for $d := 0$ to q do / for column d of Q 's */*

begin / computing the desired control points P 's */*

Apply curve approximation to column d of the Q 's

(i.e., $Q_{0d}, Q_{1d}, \dots, Q_{md}$) using

** degree p*

** parameters $s_0, s_1, \dots, s_m$*

The result is column d of the desired control points

$P_{0d}, P_{1d}, \dots, P_{pd}$

/ P 's form a $(p+1) \times (q+1)$ matrix */*

end

The computed $(p+1) \times (q+1)$ control points P_{ij} , degree (p, q) define an approximation Bézier surface of degree (p, q) for the given data points.

**)*

```

let bezierSurface data (p,q) alpha beta =
  let r = ref 0      in
  (* Compute parameters in the u-direction s0, s1, ..., sm;
     Compute parameters in the v-direction t0, t1, ..., tn; *)
  param2D data;

  let n    = makeN q alpha (bernsteinMatrix q data.t) in
  let nT   = transpose n                               in
  let qx   = makeQ q alpha data.x
  and qy   = makeQ q alpha data.y
  and qz   = makeQ q alpha data.z                     in
  let qxn  = mult qx n
  and qyn  = mult qy n
  and qzn  = Array2.create float64 c_layout (data.m+1) (q+1) in
  let nTn  = mult nT n
  and idxV = Array1.create int c_layout (q + 1)
  and idxVsig = Array1.create int c_layout (q + 1) in

  (* Apply LU decomposition to nTn, destructively *)
  r := lu_dcmp nTn (q+1) idxV;

  for c = 0 to data.m do (* for row c of D's *)
    (* computing "intermediate data points" Q's *)
    (* Apply curve approximation to row c of data points
       (i.e., Dc0, Dc1, ... Dcn) using
       * degree q
       * parameters t0, t1, ..., tn *)

    (* Resolva o sistema (VtV)P=Qc para P *)
    (* Apply back substitution to find Pi coefficients. Estes
       coeficientes estão em Qc.*)
    lu_bksb nTn (q+1) idxV (Array2.slice_left qxn c);
    lu_bksb nTn (q+1) idxV (Array2.slice_left qyn c);

  let dz = Array2.slice_left qz c
  and ds = Array2.slice_left data.sd c in
  let ns = makeN q alpha (bernsteinMatrixSigma q data.t ds) in
  for k = 0 to data.n do
    dz.{k} <- dz.{k} /. ds.{k}
  done;

```

```

let nsT    = transpose ns    in
let nsTns  = mult nsT ns
and nsTdz  = multV nsT dz in

  (* Apply LU decomposition to vsTvs, destructively *)
  r := lu_dcmp nsTns (q+1) idxVsig;

  (* Apply back substitution to find Pzi coefficients.
     Estes coeficientes estão em vsTdz. *)
  lu_bksb nsTns (q+1) idxVsig nsTdz;
  Array1.blit nsTdz (Array2.slice_left qzn c);

  (* As coordenadas do ponto de controle Pci são:
     (qxn_ci, qyn_ci, qzn_ci) *)

  (* The result is row c of the "intermediate data points"
     Qc0, Qc1, ..., Qcq *)
  (* Q's form a (m+1)x(q+1) matrix *)
done;

let w      = makeN p beta (bernsteinMatrix p data.s)    in
let wT     = transpose w                                in
let px     = mult wT (makeQv p beta qxn)
and py     = mult wT (makeQv p beta qyn)
and pz     = mult wT (makeQv p beta qzn)                in
let wTw    = mult wT w
and idxU   = Array1.create int c_layout (p + 1)
and pxT    = transpose px
and pyT    = transpose py
and pzT    = transpose pz                                in

  (* Apply LU decomposition to uTu, destructively *)
  r := lu_dcmp wTw (p+1) idxU;
  for d = 0 to q do
    (* Resolva o sistema (UtU)P=Pc para P *)
    (* Apply back substitution to find Pi coefficients.
       Estes coeficientes estão em Pc. *)
    lu_bksb wTw (p+1) idxU (Array2.slice_left pxT d);
    lu_bksb wTw (p+1) idxU (Array2.slice_left pyT d);
    lu_bksb wTw (p+1) idxU (Array2.slice_left pzT d);
  done;

  let x = transpose pxT
  and y = transpose pyT
  and z = transpose pzT    in
    (* The result is column d of the desired control points

```

```

    P0d, P1d, ..., Ppd *)
    (* P's form a (p+1)x(q+1) matrix *)

    (* The computed (p+1)x(q+1) control points Pij, degree (p,q)
       define an approximation Bézier surface of degree (p,q)
       for the given data points. *)
    (x,y,z)
;;

let s (u,v) (p,q) (x,y,z) =
let somaX = ref 0.0
and somaY = ref 0.0
and somaZ = ref 0.0 in
for i = 0 to p do
  for j = 0 to q do
    let bip = bernstein (i,p) u
    and bjq = bernstein (j,q) v in
    let bb = bip *. bjq in
    let px = x.{i,j} *. bb
    and py = y.{i,j} *. bb
    and pz = z.{i,j} *. bb in
    somaX := !somaX +. px;
    somaY := !somaY +. py;
    somaZ := !somaZ +. pz;
  done;
done;
(!somaX,!somaY,!somaZ)
;;

let imprime_pontos_de_controle () =
  let dados = d()
  and p = 3
  and q = 11
  and alpha = 0.01
  and beta = 0.01 in

  let (x,y,z) = bezierSurface dados (p,q) alpha beta in
  for i = 0 to p do
    for j = 0 to q do
      Printf.printf "(%E, %E, %E)\t" x.{i,j} y.{i,j} z.{i,j};
    done;
    print_newline()
  done;;

let imprime_pontos_da_superfície () =
  let dados = d()
  and graus = (3,11)
  and alpha = 0.00

```

```
and beta  = 0.00 in

let pc = bezierSurface dados graus alpha beta
and u   = ref 0.0
and v   = ref 0.0 in
while (!u <= 1.0) do
  v := 0.0;
  while (!v <= 1.0) do
    let (x,y,z) = s (!u,!v) graus pc in
    Printf.printf "%E %E %E\n" x y z;
    v := !v +. 0.01
  done;
  print_newline();
  u := !u +. 0.1
done
;;

let _ = imprime_pontos_da_superfície();;
```


Apêndice D - Código para as ondaletas

Neste apêndice é mostrado o código usado nos experimentos com as ondaletas. Este código é uma reimplementação in OCaml da biblioteca LIFTPACK¹, escrita em C.

O primeiro passo para usar este código é compilá-lo. Para isto siga os passos abaixo

```
ocamlopt -inline 2 -c util.mli
ocamlopt -inline 2 -c util.ml
ocamlopt -inline 2 -c lu.mli
ocamlopt -inline 2 -c lu.ml
ocamlopt -inline 2 -c neville.mli
ocamlopt -inline 2 -c neville.ml
ocamlopt -inline 2 -c filter.mli
ocamlopt -inline 2 -c filter.ml
ocamlopt -inline 2 -c lift.mli
ocamlopt -inline 2 -c lift.ml
ocamlopt -inline 2 -c mallat.mli
```

¹Disponível em <http://www.cs.dartmouth.edu/~sp/liftpack/>

```
ocamlopt -inline 2 -c mallat.ml

ocamlopt -inline 2 -c flwt.mli

ocamlopt -inline 2 -c flwt.ml

ocamlopt -inline 2 -a -o libflwt.cmxa util.cmx lu.cmx neville.cmx filter.cmx \
lift.cmx mallat.cmx flwt.cmx
```

Após a execução deste passo será criada a biblioteca `libflwt.cmxa` que contém a implementação das ondaletas. Esta biblioteca pode ser usado com o módulo `ondaleta.ml`, que fornece um exemplo de uso. Este último módulo pode ser compilado usando-se:

```
ocamlopt -o ondaleta.exe bigarray.cmxa unix.cmxa libflwt.cmxa ondaleta.ml
```

D.1 Util.mli

```
type direction = Col of int | Lin of int

val coord : int -> direction -> int * int
val odd : int -> bool
val ceil2 : int -> int -> int
val zero : float -> bool
val logBaseN : float -> float -> float
```

D.2 Util.ml

```
(*
 * $Id: util.c,v 1.4 1996/08/16 17:25:54 fernande Exp $
 * $Source: /sgi.acct/sweldens/cvs/liftpack/Util/util.c,v $
 * Author: Gabriel Fernandez
 * Modifier: Alexsandro Soares 2003/10/24 (Ocaml version)
 *
 * This file contains the definition of useful mathematical and
 * general comparison functions.
```

```

    *)
    (* do not edit anything above this line *)

type direction =
    Col of int
  | Lin of int
;;

let coord z dir =
    match dir with
    Col c -> (z, c)
  | Lin l -> (l, z)
;;

let odd x = (x land 0x01) = 1
and ceil2 num den = (num+den-1) / den ;;

let zero x = (-.epsilon_float <= x) && (x <= epsilon_float) ;;

let logBaseN x n = (log10 x) /. log10 n;;

```

D.3 Lu.mli

exception Singular_Matrix

```

val lu_dcmp :
    (float, 'a, 'b) Bigarray.Array2.t -> int -> (int, 'c, 'd)
    Bigarray.Array1.t -> int
val lu_bksb :
    (float, 'a, 'b) Bigarray.Array2.t -> int -> (int, 'c, 'd)
    Bigarray.Array1.t -> (float, 'e, 'f) Bigarray.Array1.t -> unit

```

D.4 Lu.ml

```

(*
 * -- Mode: ANSI C --
 * $Id: lu.c,v 1.6 1996/11/20 02:38:51 fernande Exp $
 * $Source: /sgi.acct/sweldens/cvs/liftpack/Lifting/lu.c,v $
 * Author: Gabriel Fernandez
 * Modifier: Alexsandro Soares 2003/30/10 (Ocaml version)
 *)

```

```

*  Definition of the functions used to perform the LU
*  decomposition of matrix a. Function LUdcmp() performs the LU
*  operations on a[][] and function LUbksb() performs the
*  backsubstitution giving the solution in b[];
*)

```

```

open Bigarray

```

```

open Util

```

```

exception Singular_Matrix;;

```

```

let tiny = 1.0e-20;; (* A small number *)

```

```

(*
*  LUdcmp function: Given a matrix a [0..n-1][0..n-1], this
*  routine replaces it by the LU decomposition of a rowwise
*  permutation of itself. a and n are input. a is output,
*  arranged with L and U in the same matrix; indx[1..n] is an
*  output vector that records the row permutation affected by
*  the partial pivoting; d is output as +/-1 depending on
*  whether the number of row interchanges was even or odd,
*  respectively. This routine is used in combination with
*  LUbksb() to solve linear equations or invert a matrix.
*)

```

```

let lu_dcmp a n indx =
  let vv    = Array1.create float64 c_layout n
  and d     = ref 1                      (* No row interchanges yet. *)
  and big   = ref 0.0
  and sum   = ref 0.0
  and imax  = ref 0
  and dum   = ref 0.0
  and temp  = ref 0.0 in
  for i = 0 to n - 1 do (* Loop over rows to get the implicit
                           scaling information. *)
    big := 0.0 ;
    for j = 0 to n - 1 do
      temp := abs_float a.{i,j};
      if !temp > !big
      then big := !temp;
    done;
    if !big = 0.0
    then raise Singular_Matrix;
    (* Nonzero largest element. *)
    vv.{i} <- 1.0 /. !big    (* Save the scaling. *)

```

```

done;

for j = 0 to n - 1 do  (* This is the loop over columns of
                        Crout's method. *)
  for i = 0 to j - 1 do  (* Sum form of a triangular matrix
                          except for i=j. *)
    sum := a.{i,j};
    for k = 0 to i - 1 do
      sum := !sum - a.{i,k}*a.{k,j}
    done;
    a.{i,j} <- !sum
  done;
  big := 0.0;          (* Initialize for the search for
                        largest pivot element. *)
  imax := 0;          (* Set default value for imax *)
  for i = j to n - 1 do  (* This is i=j of previous sum
                          and i=j+1...N *)
    sum := a.{i,j};    (* of the rest of the sum. *)
    for k = 0 to j - 1 do
      sum := !sum - a.{i,k}*a.{k,j}
    done;
    a.{i,j} <- !sum;
    dum := vv.{i} *. abs_float !sum;

    if !dum >= !big
    then begin
      (* Is the figure of merit for the pivot better than
         the best so far? *)
      big := !dum;
      imax := i
    end;
  done;
  if j <> !imax      (* Do we need to interchange rows? *)
  then begin        (* Yes, do so... *)
    for k = 0 to n - 1 do
      dum := a.{!imax,k};
      a.{!imax,k} <- a.{j,k};
      a.{j,k} <- !dum
    done;
    d := - (!d);    (* ...and change the parity of d. *)
    vv.{!imax} <- vv.{j} (* Also interchange the scale
                          factor. *)
  end;
  indx.{j} <- !imax;
  if a.{j,j} = 0.0
  then a.{j,j} <- tiny;

(* If the pivot element is zero the matrix is singular

```

```

        (at least to the precision of the algorithm). For some
        applications on singular matrices, it is desirable
        to substitute TINY for zero. *)
    if j <> (n-1) (* Now, finally divide by pivot element. *)
    then begin
        dum := 1.0/.a.{j,j};
        for i = j + 1 to n-1 do
            a.{i,j} <- a.{i,j} *. !dum
        done;
    end;
done; (* Go back for the next column in the reduction. *)
!d
;;

(*
*  LUbksb function: Solves the set of n linear equations  $A.X = B$ .
*  Here a[0..n-1][0..n-1] is input, not as the matrix A but
*  rather as its LU decomposition, determined by the routine
*  LUdcmp(). indx[1..n] is input as the permutation vector
*  returned by LUdcmp(). b[1..n] is input as the right hand
*  side vector B, and returns with the solution vector X. a,
*  n, and indx are not modified by this routine and can be
*  left in place for successive calls with different right-hand
*  sides b. This routine takes into account the possibility
*  that b will begin with many zero elements, so it is efficient
*  for use in matrix inversion.
*)

let lu_bksb a n indx b =
    let ii = ref (-1)
    and sum = ref 0.0
    and ip = ref 0 in
    for i = 0 to n-1 do (* When ii is set to a positive value, it *)
        ip := indx.{i}; (* will become the index of the first *)
        sum := b.{!ip}; (* nonvanishing element of b. We now do
                        the forward substitution. *))
        b.{!ip} <- b.{i}; (* The only new wrinkle is to unscramble *)
        if !ii >= 0 (* the permutation as we go. *)
        then
            for j = !ii to i-1 do
                sum := !sum -. a.{i,j} *. b.{j}
            done
        else if !sum <> 0.0 (* A nonzero element was encountered,
                        so from now on we will have to do *)
            then ii := i; (* the sums in the loop above *)
        b.{i} <- !sum

```

```

done;

for i = n - 1 downto 0 do (* Now we do the backsubstitution. *)
  sum := b.{i};
  for j = i + 1 to n - 1 do
    sum := !sum -. a.{i,j} *. b.{j}
  done;
  b.{i} <- !sum /. a.{i,i}; (* Store a component of the
                             solution vector X. *)
  if zero b.{i} (* Verify small numbers. *)
  then b.{i} <- 0.0;
done
;;

```

D.5 Neville.mli

```

exception Denon_Near_Zero
val neville :
  (float , 'a , 'b) Bigarray.Array1.t ->
  (float , Bigarray.float64_elt , Bigarray.c_layout)
  Bigarray.Array1.t -> float -> float

```

D.6 Neville.ml

```

(*
 *
 * (Polynomial Interpolation)
 * $Id: neville.ml,v 1.5 1996/09/17 16:10:41 fernande Exp $
 * $Source: /sgi.acct/sweldens/cvs/liftpack/Lifting/neville.c,v $
 * Author: Wim Sweldens, Gabriel Fernandez
 * Modifier: Alexsandro Soares 2003/10/24 (Ocaml version)
 *
 * Given n points of the form (x,f), this program uses the Neville
 * algorithm to find the value at xx of the polynomial of degree
 * (n-1) interpolating the points (x,f).
 * Ref: Stoer and Bulirsch, Introduction to Numerical Analysis,
 * Springer-Verlag.
 *)

open Bigarray
open Util

```

```

exception Denon_Near_Zero;;

let neville x f xx =
  let n = Array1.dim f in
  let y = Array1.create float64 c_layout n in
  Array1.blit f y;
  for i = 0 to n-1 do
    for j = i-1 downto 0 do
      let den = x.{i} -. x.{j} in
      if zero den
      then raise Denon_Near_Zero
      else y.{j} <- y.{j+1} +. (y.{j+1} -. y.{j}) *.
        (xx -. x.{i})
      /. den
    done;
  done;
  y.{0}
;;

```

D.7 Filter.mli

```

exception Not_Even
val getFilter :
  int ->
  (float , Bigarray.float64_elt , Bigarray.c_layout)
  Bigarray.Array1.t

```

D.8 Filter.ml

```

(*
 * $Id: filter.c,v 1.4 1996/08/16 17:25:47 fernande Exp $
 * $Source: /sgi.acct/sweldens/cvs/liftpack/Lifting/filter.c,v $
 * Author: Gabriel Fernandez
 * Modifier: Alexsandro Santos 2003/10/24 (Ocaml version)
 *
 * This module finds the filter coefficients used in the wavelet
 * transform to generate the wavelet coefficients (gamma) using
 * the contribution of the lambda values. The process uses the
 * Neville's algorithm (neville.c) for the polynomial interpolation
 * and a given value, N, to know how many filter coefficients are
 * needed. The results are returned in a Matrix of N/2+1 rows by
 * N columns in a linear representation.

```

```

*)
(* do not edit anyhting above this line *)

open Bigarray

open Util
open Neville

exception Not_Even;;

(*
 * GetFilter function: finds the filter coefficients used in the
 * Gamma coefficients prediction routine. The
 * Neville polynomial interpolation algorithm
 * is used to find all the possible values for
 * the filter coefficients. Thanks to this
 * process, the boundaries are correctly
 * treated without including artifacts.
 * Results are ordered in matrix as follows:
 * 0 in the left and N in the right
 * 1 in the left and N-1 in the right
 * 2 in the left and N-2 in the right
 * .
 * .
 * .
 * N/2 in the left and N/2 in the right
 * For symmetry, the cases from
 * N/2+1 in the left and N/2-1
 * to
 * N in the left and 0 in the right
 * are the same as the ones shown above, but
 * with switched sign.
 *)

let getFilter n =
  let xa = Array1.create float64 c_layout n
  and ya = Array1.create float64 c_layout n in

  (* Check for validity of n *)
  if (odd n) || (n == 0)
  then raise Not_Even;

  (* Number of cases for filter calculations *)
  let nrows = (n asr 1) + 1 in (* N/2 + 1 *)

  (* Allocate memory for filter matrix *)
  let filter = Array1.create float64 c_layout (nrows*n) in

```

```

(* Generate values of xa *)
xa.{0} <- 0.5 *.(1. -. (float n));      (* -N/2 + 0.5 *)
for col = 1 to n - 1 do
  xa.{col} <- xa.{col - 1} +. 1.0
done;

let i = ref ((nrows * n) - 1) in
(* index of last position in filter matrix *)

(* Find filter coefficient values *)
for row = 0 to nrows - 1 do
  let x = float row in
    for col = 0 to n - 1 do
      Array1.fill ya 0.0;
      ya.{col} <- 1.0;
      filter.{!i} <- neville xa ya x;
      i := !i - 1
    done;
done;
filter
;;

```

D.9 Lift.mli

```

exception No_Lifting_Matrix

val initMoment : int -> int -> int -> bool -> bool
val updateMoment :
  (float , 'a, 'b) Bigarray.Array2.t ->
  (float , 'c, 'd) Bigarray.Array1.t ->
  int -> int -> int -> int -> int -> unit
val updateLifting :
  (float , 'a, 'b) Bigarray.Array1.t ->
  (float , 'c, 'd) Bigarray.Array2.t -> int -> int -> int ->
  int -> unit
val getLifting :
  (float , 'a, 'b) Bigarray.Array1.t ->
  (float , 'c, 'd) Bigarray.Array1.t ->
  int -> int -> int -> int -> bool -> bool -> unit

```

D.10 Lift.ml

```

(*
 * $Id: lift.c,v 1.7 1996/11/20 02:38:51 fernande Exp $
 * $Source: /sgi.acct/sweldens/cvs/liftpack/Lifting/lift.c,v $
 * Author: Gabriel Fernandez
 * Modifier: Alexsandro Soares 2003/10/24 (Ocaml version)
 *
 * This module finds the lifting coefficients used in the wavelet
 * transform to update the lambda coefficients using the
 * contribution of the neighboring gamma values. The function
 * GetLifting() updates the values of the (integral, moment)
 * tuples. Then, calculates the corresponding lifting coefficients.
 *)

open Bigarray
open Printf

open Util
open Lu

exception No_Lifting_Matrix;;

(* Global static variables *)
let momentX = ref (Array2.create float64 c_layout 0 0)
and momentY = ref (Array2.create float64 c_layout 0 0) ;;

(*
 * InitMoment function: Initializes the values of the
 * Integral-Moments matrices. The moments are equal
 * to  $k^i$ , where  $k$  is the coefficient index and  $i$  is
 * the moment number (0,1,2,...). There is one matrix
 * for each direction in case the idmensions of  $X$  and  $Y$ 
 * are different.
 *)

let initMoment r sizeX sizeY print =
  if ( odd r || r < 0)
  then false
  else begin

    (* Allocate memory for the Integral-Moment matrices *)
    momentX := Array2.create float64 c_layout sizeX r;
    momentY := Array2.create float64 c_layout sizeY r;

    (* Initialize Integral and Moments *)
    (* Integral is equal to one at the beginning since all the *)

```

```

(* filter coefficients must add to one for each gamma *)
(* coefficient. 1st moment = k, 2nd moment = k^3,
   3rd moment = k^3, ... *)

for row = 0 to sizeY - 1 do    (* for the cols *)
  for col = 0 to r - 1 do
    if (row = 0 && col = 0 )
      then !momentY.{row,col} <- 1.0 (* pow() domain error *)
      else !momentY.{row,col} <- (float row) ** (float col)
    done;
  done;

for row = 0 to sizeX - 1 do    (* for the rows *)
  for col = 0 to r - 1 do
    if (row = 0 && col = 0 )
      then !momentX.{row,col} <- 1.0 (* pow() domain error *)
      else !momentX.{row,col} <- (float row) ** (float col)
    done;
  done;

(* Print Moment Coefficients *)

if print
then
  begin
    printf "Integral-Moments Coefficients for X (R=%2d)\n" r;
    for row = 0 to sizeX - 1 do
      printf "Coeff[%d] (%.20g" row !momentX.{row,0};
      for col = 1 to r - 1 do
        printf ",%.20g" !momentX.{row,col}
      done;
      printf ")\n"
    done;
    printf "Integral-Moments Coefficients for Y (R=%2d)\n" r;
    for row = 0 to sizeY - 1 do
      printf "Coeff[%d] (%.20g" row !momentY.{row,0};
      for col = 1 to r - 1 do
        printf ",%.20g" !momentY.{row,col}
      done;
      printf ")\n"
    done;

  end;
true

end
;;

```

```

(*
 * UpdateMoment function: calculates the integral-moment tuple for
 * the current level of calculations.
 *)

let updateMoment moment filter step2 noGammas len n r =
  (*****
  (* Update Integral-Moments information *)
  *****)

  (* Calculate number of iterations for each case *)
  let top1 = (n asr 1) - 1 (* L < R *)
  and top3 = (n asr 1) - (len land 0x01) in (* L > R *)
  let top2 = noGammas - (top1 + top3) in (* L = R *)

  (* Cases where nbr. left Lambdas < nbr. right Lambdas *)
  let idxF = ref 0 (* initialize to first filter row *)
  and idxG = ref (step2 asr 1) (* second coefficient is Gamma *)
  and idxL = ref 0 in
  for row = 1 to top1 do
    idxL := 0; (* the first Lambda is always the first coefficient *)
    idxF := !idxF + n; (* go to next filter row *)
    for col = 0 to n - 1 do
      (* Update (int, mom_1, mom_2, ...) *)
      for j = 0 to r - 1 do
        moment.{!idxL, j} <- moment.{!idxL, j} +.
          filter.{!idxF + col} *. moment.{!idxG, j}
      done;
      (* Jump to next Lambda coefficient *)
      idxL := !idxL + step2
    done;
    idxG := !idxG + step2 (* go to next Gamma coefficient *)
  done;

  (* Cases where nbr. left Lambdas = nbr. right Lambdas *)
  idxF := !idxF + n; (* go to last filter row *)
  for i = 0 to top2 - 1 do
    idxL := i * step2;
    for col = 0 to n - 1 do
      (* Update (int, mom_1, mom_2, ...) *)
      for j = 0 to r - 1 do
        moment.{!idxL, j} <- moment.{!idxL, j} +.
          filter.{!idxF + col} *. moment.{!idxG, j}
      done;
      (* Jump to next Lambda coefficient *)
      idxL := !idxL + step2
    done;

```

```

    idxG := !idxG + step2; (* go to next Gamma coefficient*)
    (* and stay in the same row of filter coefficients *)
done;

(* Cases where nbr. left Lambdas > nbr. right Lambdas *)
for row = top3 downto 1 do
    idxL := (top2-1)*step2; (* first Lambda is always in this place *)
    idxF := !idxF - n;      (* go to previous filter row *)
    for col = n - 1 downto 0 do
        (* Update (int, mom_1, mom_2, ...) *)
        for j = 0 to r - 1 do
            moment.{!idxL, j} <- moment.{!idxL, j} +.
            filter.{!idxF + col} *. moment.{!idxG, j}
        done;
        (* Jump to next Lambda coefficient and next filter row *)
        idxL := !idxL + step2
    done;
    idxG := !idxG + step2 (* go to next Gamma coefficient *)
done
;;

(*
* UpdateLifting function: calculates the lifting coefficients
* using the given integral-moment tuple.
*)

let updateLifting lifting moment len step2 noGammas r =
    (*****)
    (* Calculate Lifting Coefficients *)
    (*****)

    let lcIdx = ref 0 (* first element of lifting vector *)
    and idxL = ref 0 (* pointers to Lambda & Gamma coeffs, resp. *)
    and idxG = ref 0
    and indx = Array1.create int c_layout r
    (* Allocate space for the indx[] vector *)
    and d = ref 0
    and lift = Array2.create float64 c_layout r r
    (* temporary matrix to be solved *)
    and b = Array1.create float64 c_layout r
    (* temporary solution vector *)
    (* Calculate number of iterations for each case *)
    and top1 = (r asr 1) - 1 (* L < R *)
    and top3 = (r asr 1) - (len land 0x01) in (* L > R *)
    let top2 = noGammas - (top1 + top3) in (* L = R *)

```

```

(* Cases where nbr. left Lambdas < nbr. right Lambdas *)
idxG := step2 asr 1; (* second coefficient is Gamma *)
for i = 0 to top1 - 1 do
  idxL := 0; (* the first Lambda is always the first
              coefficient *)
  for col = 0 to r-1 do
    (* Load temporary matrix to be solved *)
    for row = 0 to r-1 do
      lift.{row, col} <- moment.{!idxL, row} (* matrix *)
    done;
    (* Jump to next Lambda coefficient *)
    idxL := !idxL + step2
  done;
  (* Apply LU decomposition to lift [[]] *)
  d := lu_dcmp lift r idx;
  (* Load independent vector *)
  for j = 0 to r - 1 do
    b.{j} <- moment.{!idxG, j} (* independent vector *)
  done;
  (* Apply back substitution to find lifting coefficients *)
  lu_bksb lift r idx b;
  for col = 0 to r-1 do (* save them in lifting vector *)
    lifting.{!lcIdx} <- b.{col};
    lcIdx := !lcIdx + 1
  done;
  idxG := !idxG + step2; (* go to next Gamma coefficient *)
done;

(* Cases where nbr. left Lambdas = nbr. right Lambdas *)
for i = 0 to top2 - 1 do
  idxL := i * step2;
  for col = 0 to r - 1 do
    (* Load temporary matrix to be solved *)
    for row = 0 to r - 1 do
      lift.{row, col} <- moment.{!idxL, row} (* matrix *)
    done;
    (* Jump to next Lambda coefficient *)
    idxL := !idxL + step2
  done;
  (* Apply LU decomposition to lift [[]] *)
  d := lu_dcmp lift r idx;

  (* Load independent vector *)
  for j = 0 to r - 1 do
    b.{j} <- moment.{!idxG, j} (* independent vector *)
  done;
  (* Apply back substitution to find lifting coefficients *)

```

```

    lu_bksb lift r indx b;

    for col = 0 to r-1 do (* save them in lifting vector *)
        lifting.{!lcIdx} <- b.{col};
        lcIdx := !lcIdx + 1
    done;
    idxG := !idxG + step2; (* go to next Gamma coefficient *)
done;

(* Cases where nbr. left Lambdas > nbr. right Lambdas *)
for i = 0 to top3 - 1 do
    idxL := (top2-1)*step2; (* first Lambda is always in this place *)
    for col = 0 to r-1 do
        (* Load temporary matrix to be solved *)
        for row = 0 to r-1 do
            lift.{row,col} <- moment.{!idxL,row} (* matrix *)
        done;
        (* Jump to next Lambda coefficient *)
        idxL := !idxL + step2
    done;
    (* Apply LU decomposition to lift [][ ] *)

    d := lu_dcmp lift r indx;
    (* Load independent vector *)
    for j=0 to r-1 do
        b.{j} <- moment.{!idxG,j} (* independent vector *)
    done;
    (* Apply back substitution to find lifting coefficients *)
    lu_bksb lift r indx b;
    for col = 0 to r-1 do (* save them in lifting vector *)
        lifting.{!lcIdx} <- b.{col};
        lcIdx := !lcIdx + 1
    done;
    idxG := !idxG + step2; (* go to next Gamma coefficient *)
done;
;;

(*
* GetLifting function: Updates corresponding moment matrix
* X if dir is FALSE and Y if dir is TRUE) and calculates
* the lifting coefficients using the new moments. This
* function is used for the forward transform and uses the
* original filter coefficients.
*)

let getLifting lifting filter step n r l dir tipo =

```

```

    (*****)
    (* Check for boolean variables *)
    (*****)

    (* Check for the lifting matrix *)
    if (Array1.dim lifting = 0)
    then raise No_Lifting_Matrix;

    (* Select the appropriate moment matrix *)
    let moment = if (not dir)
                  then !momentX    (* X direction *)
                  else !momentY    (* Y direction *)
    in
    (*****)
    (* Initialize important constants *)
    (*****)

    (* Calculate number of coefficients and step size *)
    let len = ceil2 l step (* number of coefficients in this level *)
    and step2 = step lsl 1 (* step size between same coefficients *)
    in

    (* Check for number of Gamma coefficients *)
    let noGammas = len asr 1 in
    (* number of Gamma coeffs = floor(len/2) *)

    (*****)
    (* Decide order of operations *)
    (*****)
    if tipo = false
    then begin (* Direct transform *)
        updateMoment moment filter step2 noGammas len      n r ;
        updateLifting lifting moment len      step2      noGammas r
    end
    else begin (* Inverse transform *)
        updateLifting lifting moment len      step2      noGammas r ;
        updateMoment moment filter step2 noGammas len      n r
    end
end
;;

```

D.11 Mallat.mli

```

val inplaceChange1D :
  (float , 'a, 'b) Bigarray.Array2.t ->

```

```

Util.direction -> int -> bool -> unit
val flwtChangeFormat :
  (float , 'a , 'b) Bigarray.Array2.t ->
  int -> int -> int -> int -> int -> bool -> unit

```

D.12 Mallat.ml

```

(*
 * $Id: mallat.c,v 1.5 1996/10/06 09:24:54 fernande Exp $
 * $Source: /sgi.acct/sweldens/cvs/liftpack/Lifting/mallat.c,v $
 * Author: Gabriel Fernandez, Senthil Periaswamy
 * Modifier: Alexsandro Soares 2003/29/10 (Ocaml version)
 *
 * This file contains routines to manipulate the results of an
 * in-place transform in order to re-arrange the coefficients in
 * Mallat format.
 *)

open Bigarray

open Util

(*
 * InplaceChange1D function: arranges the wavelet coefficients
 * in the given vector to put them in Mallat or Lifting
 * format, depending on the value of the inverse variable.
 *)

let inplaceChange1D data dir len inverse =
  (* Allocate memory for buffer vector *)
  let buffer = Array1.create float64 c_layout len
  and idxL = ref 0
  and idxG = ref 0
  and i = ref 0
  (* Number of Gamma coefficients *)
  and noGammas = len asr 1 in
  let noLambdas = len - noGammas in

  (* Check whether to do Lifting format or Mallat format *)
  if not inverse
  then begin
    (* Set lambda and gamma pointers *)
    idxL := 0; (* point at the beginning of vector *)
    idxG := noLambdas; (* point at half of the vector *)

```

```

(* Move all lambdas to the left and all gammas to the right *)
i:=0;
while !i < len - 2 do
  let (x0,y0) = coord !i dir
  and (x1,y1) = coord (!i+1) dir in
    buffer.{!idxL} <- data.{x0,y0};
    buffer.{!idxG} <- data.{x1,y1};
    idxL := !idxL + 1;
    idxG := !idxG + 1;
  i := !i + 2
done;
if odd len
then begin
  i := len - 1;
  let (x0,y0) = coord !i dir in
    buffer.{!idxL} <- data.{x0,y0}
end
else begin
  i := len - 2;
  let (x0,y0) = coord !i dir
  and (x1,y1) = coord (!i+1) dir in
    buffer.{!idxL} <- data.{x0,y0};
    buffer.{!idxG} <- data.{x1,y1}
end
end
else begin
  (* Set lambda and gamma pointers *)
  idxL := 0; (* point at the beginning of vector *)
  idxG := noLambdas; (* point at half of the vector *)

  (* Alternate lambdas and gammas *)
  i:=0;
  while !i < len - 2 do
    let (x0,y0) = coord !idxL dir
    and (x1,y1) = coord !idxG dir in
      buffer.{!i} <- data.{x0,y0};
      buffer.{!i+1} <- data.{x1,y1};
      idxL := !idxL + 1;
      idxG := !idxG + 1;
    i := !i + 2
  done;
  if odd len
  then begin
    i := len - 1;
    let (x0,y0) = coord !idxL dir in
      buffer.{!i} <- data.{x0,y0}
    end
  else begin

```

```

        i := len - 2;
        let (x0,y0) = coord !idxL dir
        and (x1,y1) = coord !idxG dir in
            buffer.{!i} <- data.{x0,y0};
            buffer.{!i+1} <- data.{x1,y1}
        end;
    end;
    (* Write results in original vector *)
    idxL := 0;
    for z = 0 to len - 1 do
        let (x0,y0) = coord z dir in
            data.{x0,y0} <- buffer.{!idxL};
            idxL := !idxL + 1
        done;
    ;;

(*
* FLWTChangeFormat function: changes a given image from the
* Lifting format to the Mallat format, and viceversa.
*)

let flwtChangeFormat data width height n nTilde levels inverse =
    let nX = ref 0
    and nY = ref 0
    and lenX = ref 0
    and lenX2 = ref 0
    and lenY = ref 0
    and lenY2 = ref 0 in
        (* Calculate number of iterations n. It is the maximum *)
        (* value such that the following relation is satisfied. *)
        (*  $(2^n) * (N-1) \leq L < (2^{(n+1)}) * (N-1) + 1$  *)
        (* Where  $L = \max(\text{signal's length in } X\text{-}Y \text{ direction})$ . *)
        (* and  $N = \max(\# \text{ dual vanish mom } \& \# \text{ real vanish mom})$  *)
        (* Hence, solving for n, we have the following equation *)
        (* for all the cases:  $n = \text{floor}(\log_2((L-1)/(N-1)))$  *)

        if n = 1 || nTilde = 1 (* Haar wavelets *)
        then begin
            (* Iterations in X *)
            nX := if (width = 1) then 0
            else (int_of_float (ceil (logBaseN (float width) 2.0)));
            (* Iterations in Y *)
            nY := if (height = 1) then 0
            else (int_of_float (ceil (logBaseN (float height) 2.0)));
        end
        else begin (* Biorthogonal wavelets *)

```

```

    (* Max vanishing moments *)
    let maxN = (max n nTilde) - 1 in
    (* Iterations in X *)
    nX := if (width = 1) then 0
    else (int_of_float (logBaseN (float ((width-1)/maxN)) 2.0));
    (* Iterations in Y *)
    nY := if (height = 1) then 0
    else (int_of_float (logBaseN (float ((height-1)/maxN)) 2.0));
end;
(* Find lowest levels *)
nX := if (!nX < 0) then 0 else min levels !nX;
nY := if (!nY < 0) then 0 else min levels !nY;
(* find lowest level *)
(* Total number of iterations *)
let n' = max !nX !nY in (* find minimum number of iterations *)
if not inverse
then begin
    (* Initial lengths *)
    lenX := width;
    lenX2 := width;
    lenY := height;
    lenY2 := height;
    (* Apply Mallat ordering to rows and columns *)
    for i = 0 to n' - 1 do
        (* For the rows *)
        if !nX > 0
        then begin
            nX := !nX - 1;
            for y = 0 to !lenY2 - 1 do
                inplaceChange1D data (Lin y) !lenX inverse
            done;
            lenX := !lenX - (!lenX asr 1);
            lenY2 := !lenY2 - (!lenY2 asr 1)
        end
        else nX := !nX - 1;
        (* For the columns *)
        if !nY > 0
        then begin
            nY := !nY - 1;
            for x = 0 to !lenX2 - 1 do
                inplaceChange1D data (Col x) !lenY inverse
            done;
            lenY := !lenY - (!lenY asr 1);
            lenX2 := !lenX2 - (!lenX2 asr 1)
        end
        else nY := !nY - 1;
    done;
end
end

```

```

else begin (* inverse *)

    (* Apply Lifting ordering to rows and columns *)
    for i = n' - 1 downto 0 do
        (* Initial lengths *)
        lenX := width;
        lenX2 := width;
        lenY := height;
        lenY2 := height;

        (* For the columns *)
        if !nY > 0
        then begin
            nY := !nY - 1;
            (* Calculate new length *)
            for j = 0 to i - 1 do
                lenY := !lenY - (!lenY asr 1);
                lenX2 := !lenX2 - (!lenX2 asr 1)
            done;
            (* Reorganize coeffs *)
            for x = 0 to !lenX2 - 1 do
                inplaceChange1D data (Col x) !lenY inverse
            done;
        end
        else nY := !nY - 1;

        (* For the rows *)
        if !nX > 0
        then begin
            nX := !nX - 1;
            (* Calculate new length *)
            for j = 0 to i - 1 do
                lenX := !lenX - (!lenX asr 1);
                lenY2 := !lenY2 - (!lenY2 asr 1)
            done;
            (* Reorganize coeffs *)
            for y = 0 to !lenY2 - 1 do
                inplaceChange1D data (Lin y) !lenX inverse
            done;
        end
        else nX := !nX - 1;

    done;
end
;;

```

D.13 Flwt.mli

```

exception No_Filter_Vector

val getDual : int -> int -> int -> int -> bool -> int -> bool
val getReal : int -> int -> int -> int -> bool -> int -> unit
val flwt1D_Predict :
  (float , 'a, 'b) Bigarray.Array2.t ->
  Util.direction -> int -> int -> int -> unit
val flwt1D_Update :
  (float , 'a, 'b) Bigarray.Array2.t ->
  Util.direction ->
  int -> int -> int -> (float , 'c, 'd) Bigarray.Array1.t
  -> unit
val flwt2D :
  (float , 'a, 'b) Bigarray.Array2.t ->
  int -> int -> int -> int -> int -> bool -> unit
val flwt1D :
  (float , 'a, 'b) Bigarray.Array2.t ->
  int -> int -> int -> int -> bool -> unit
val flwt_Haar :
  (float , 'a, 'b) Bigarray.Array2.t ->
  Util.direction -> int -> int -> unit
val iflwt_Haar :
  (float , 'a, 'b) Bigarray.Array2.t ->
  Util.direction -> int -> int -> unit
val flwt2D_Haar :
  (float , 'a, 'b) Bigarray.Array2.t -> int -> int -> int
  -> bool -> unit
val flwt1D_Haar :
  (float , 'a, 'b) Bigarray.Array2.t -> int -> int -> bool
  -> unit

```

D.14 Flwt.ml

```

(*
 * $Id: flwt.c,v 1.18 1997/04/03 15:08:04 fernande Exp $
 * $Source: /sgi.acct/sweldens/cvs/liftpack/Lifting/flwt.c,v $
 * Author: Gabriel Fernandez, Senthil Periaswamy
 * Modifier: Alexsandro Soares 2003/30/10 (Ocaml version)
 *
 * >>> Fast Lifted Wavelet Transform on the Interval <<<
 *
 * .-.

```



```

        printf "\nFilter Coefficients (N=%d)\n" n;
        for row = 0 to nrow - 1 do
            printf "For %d in the left and %d in the right\n[ "
                row (n-row);
            for col = 0 to n - 1 do
                printf "%.18g " !filter.{!i};
                i := !i + 1
            done;
            printf "]\n"
        done;
    end;

    (* Allocate lifting coefficient matrices *)
    let maxN      = (max n nTilde) - 1 (* max vanishing moments *)
    and noGammas = width asr 1 in (* number of Gammas in X direction *)
    if (noGammas > 0)
    then begin
        let nX = (int_of_float (logBaseN (float ((width-1)/maxN))
                                         2.0 )) in

        (* iterations in X *)
        let nX = if (nX < 0) then 0 else min levels nX in
        (* find lowest level *)
        liftingX := Array2.create float64 c_layout nX
                                         (noGammas*nTilde);
        Array2.fill !liftingX 0.0;
    end;

    let noGammas = height asr 1 in (* number of Gammas in Y
                                     direction *)
    if (noGammas > 0)
    then begin
        let nY = (int_of_float (logBaseN (float ((height-1)/maxN))
                                         2.0 )) in

        (* iterations in Y *)
        let nY = if (nY < 0) then 0 else min levels nY in
        (* find lowest level *)
        liftingY := Array2.create float64 c_layout nY
                                         (noGammas*nTilde);
        Array2.fill !liftingY 0.0;
    end;

    (* Initialize Integral-Moments matrices *)
    initMoment nTilde width height print

;;

(*

```

```

* GetReal function: Calculates the moments and lifting coefficients
* needed for the wavelet transform. These coefficients preserve
* the moments of the Real Wavelet function.
*)

```

```

let getReal width height n nTilde print levels =
  (* Check for filter vector *)
  if (Array1.dim !filter) = 0
  then raise No_Filter_Vector;

  (* Calculate number of iterations n'. It is the maximum *)
  (* value such that the following relation is satisfied. *)
  (*  $(2^{n'}) * (n-1) \leq L < (2^{(n'+1)}) * (n-1) + 1$  *)
  (* Where  $L = \max(\text{signal's length in } X\text{-}Y \text{ direction})$ . *)
  (* and  $n = \max(\# \text{ dual vanish mom } \& \# \text{ real vanish mom})$  *)
  (* Hence, solving for  $n'$ , we have the following equation *)
  (* for all the cases:  $n' = \text{floor}(\log_2((L-1)/(n-1)))$  *)

  let maxN = (max n nTilde) - 1 in (* max vanishing moments *)
  (* Iterations in X *)
  let nX = if (width = 1) then 0
  else (int_of_float (logBaseN (float ((width-1)/maxN)) 2.0 )) in
  let nX = ref (if (nX < 0) then 0 else min levels nX)
  (* find lowest level *)
  (* Iterations in Y *)
  and nY = if (height = 1) then 0
  else (int_of_float (logBaseN (float ((height-1)/maxN)) 2.0 )) in
  let nY = ref (if (nY < 0) then 0 else min levels nY) in
  (* find lowest level *)
  (* Total number of iterations *)
  let n' = max !nX !nY in (* find minimum number of iterations *)
  let max_step = if (n' = 0) then 0 else (1 lsl n') (* maximum step *)
  and level = ref 0
  and step = ref 1 in

  (* Find lifting coefficients for all levels *)
  while !step < max_step do
    if !nX > 0
    then begin
      nX := !nX - 1;
      (* Get lifting coefficients for rows *)
      getLifting (Array2.slice_left !liftingX !nX) !filter
        !step n nTilde width false false;
      if print
      then begin
        printf "\nLifting Coefficients for Rows (Level=%d)\n"
          !level;
        let liftPtr = Array2.slice_left !liftingX !nX

```

```

        and i = ref 0
        for x = 0 to (width asr 1) - 1 do
            let buf = ref (sprintf "%.20g" liftPtr.{!i}) in
            i := !i + 1;
            for k = 1 to nTilde - 1 do
                buf := sprintf "%s,%.20g" !buf liftPtr.{!i};
                i := !i + 1
            done;
            printf "Gamma[%d] (%s)\n" x !buf
        done;
    end;
end
else nX := !nX - 1;

if !nY > 0
then begin
    nY := !nY - 1;
    (* Get lifting coefficients for columns *)
    getLifting (Array2.slice_left !liftingY !nY) !filter
        !step n nTilde height true false;
    if print
    then begin
        printf "\nLifting Coefficients for Columns (Level=%d)\n"
            !level;
        let liftPtr = Array2.slice_left !liftingY !nY
        and i = ref 0
        for y = 0 to (height asr 1) - 1 do
            let buf = ref (sprintf "%.20g" liftPtr.{!i}) in
            i := !i + 1;
            for k = 1 to nTilde - 1 do
                buf := sprintf "%s,%.20g" !buf liftPtr.{!i};
                i := !i + 1
            done;
            printf "Gamma[%d] (%s)\n" y !buf
        done;
    end;
end
else nY := !nY - 1;

level := !level + 1;    (* go to next level *)

step := !step lsl 1
done
;;

```

(* CODE FOR THE IN-PLACE FLWT (INTERPOLATION CASE, N EVEN) *)

```

(*)
* FLWT1D-Predict function: The Gamma coefficients are found as an
* average interpolation of their neighbors in order to find
* the "failure to be linear" or "failure to be cubic" or the
* failure to be the order given by N-1. This process uses the
* filter coefficients stored in the filter vector and predicts
* the odd samples based on the even ones storing the difference
* in the gammas. By doing so, a Dual Wavelet is created.
*)

let flwt1D-Predict vect dir size step n =
  (*****)
  (* Calculate values of some important variables *)
  (*****)
  let len = ceil2 size step (* # of coefficients at current level *)
  and stepIncr = step lsl 1 in (* step size between coefficients *)
  (*****)
  (* Calculate number of iterations for each case *)
  (*****)
  let j      = ref (len land 0x01) in
  let stop1  = ref (n asr 1)      in
  let stop3  = ref (!stop1 - !j)  (* L > R *)
  and stop2  = ref ((len asr 1) - n + 1 + !j) in (* L = R *)
  stop1 := !stop1 - 1;              (* L < R *)

  (*****)
  (* Predict Gamma (wavelet) coefficients (odd guys) *)
  (*****)
  let idxF = ref n (* position filter pointer *)
  and idxG = ref (stepIncr asr 1) (* second coefficient is Gamma *)
  (* Cases where nbr. left Lambdas < nbr. right Lambdas *)
  and idxL = ref 0 in
  while !stop1 <> 0 do
    stop1 := !stop1 - 1;
    idxL := 0; (* first coefficient is always first Lambda *)
    j := n;
    (* Gamma update (Gamma - predicted value) *)
    let (x0,y0) = coord !idxG dir
    and (x1,y1) = coord !idxL dir in
    vect.{x0,y0} <- vect.{x0,y0} -. vect.{x1,y1} *. !filter.{!idxF};
    idxF := !idxF + 1;
    idxL := !idxL + stepIncr; (* jump to next Lambda coefficient *)
    j := !j - 1;
    while !j <> 0 do (* use all N filter coefficients *)
      let (x0,y0) = coord !idxG dir
      and (x1,y1) = coord !idxL dir in
      vect.{x0,y0} <- vect.{x0,y0} -.

```

```

                                vect.{x1,y1} *. !filter.{!idxF});
    idxF := !idxF + 1;
    idxL := !idxL + stepIncr;  (* jump to next Lambda coefficient *)
    j := !j - 1;
done;
(* Go to next Gamma coefficient *)
idxG := !idxG + stepIncr;

done;

(* Cases where nbr. left Lambdas = nbr. right Lambdas *)
let soi = ref 0
and fP = ref 0 in
while !stop2 <> 0 do
    stop2 := !stop2 - 1;
    idxL := !soi; (* first Lambda to be read *)
    fP := !idxF; (* filter stays in same values for this cases *)
    j := n;
    (* Gamma update (Gamma - predicted value) *)
    let (x0,y0) = coord !idxG dir
    and (x1,y1) = coord !idxL dir in
    vect.{x0,y0} <- vect.{x0,y0} -. vect.{x1,y1} *. !filter.{!fP};
    fP := !fP + 1;
    idxL := !idxL + stepIncr;  (* jump to next Lambda coefficient *)
    j := !j - 1;
    while !j <> 0 do (* use all N filter coefficients *)
        (* Gamma update (Gamma - predicted value) *)
        let (x0,y0) = coord !idxG dir
        and (x1,y1) = coord !idxL dir in
        vect.{x0,y0} <- vect.{x0,y0} -. vect.{x1,y1} *. !filter.{!fP};
        fP := !fP + 1;
        idxL := !idxL + stepIncr;  (* jump to next Lambda coefficient *)
        j := !j - 1;
    done;
    (* Move start point for the Lambdas *)
    soi := !soi + stepIncr;

    (* Go to next Gamma coefficient *)
    idxG := !idxG + stepIncr;

done;

(* Cases where nbr. left Lambdas > nbr. right Lambdas *)
fP := !idxF;  (* start going backwards with the filter coefficients *)
let vet = !soi-stepIncr in (* first Lambda is always in this place *)
while !stop3 <> 0 do
    stop3 := !stop3 - 1;
    idxL := vet;  (* position Lambda pointer *)

```

```

j := n;
(* Gamma update (Gamma - predicted value) *)
fP := !fP - 1;
let (x0,y0) = coord !idxG dir
and (x1,y1) = coord !idxL dir in
vect.{x0,y0} <- vect.{x0,y0} -. vect.{x1,y1} *. !filter.{!fP};
idxL := !idxL + stepIncr; (* jump to next Lambda coefficient *)
j := !j - 1;
while !j <> 0 do (* use all N filter coefficients *)
  (* Gamma update (Gamma - predicted value) *)
  fP := !fP - 1;
  let (x0,y0) = coord !idxG dir
  and (x1,y1) = coord !idxL dir in
  vect.{x0,y0} <- vect.{x0,y0} -.
    vect.{x1,y1} *. !filter.{!fP};
  idxL := !idxL + stepIncr; (* jump to next Lambda coefficient *)
  j := !j - 1;
done;

(* Go to next Gamma coefficient *)
idxG := !idxG + stepIncr;

done
;;

(*
* FLWT1D_Update: the Lambda coefficients have to be "lifted" in
* order to find the real wavelet coefficients. The new Lambdas
* are obtained by applying the lifting coefficients stored in
* the lifting vector together with the gammas found in the
* prediction stage. This process assures that the moments of
* the wavelet function are preserved.
*)

let flwt1D_Update vect dir size step nTilde lc =
  (*****)
  (* Calculate values of some important variables *)
  (*****)
  let len = ceil2 size step (* number of coefficients at current
    level *)
  and stepIncr = step lsl 1 in (* step size between coefficients *)
  let noGammas = len asr 1 (* number of Gamma coefficients *)
  (*****)
  (* Calculate number of iterations for each case *)
  (*****)
  and j = ref (len land 0x01) in
  let stop1 = ref (nTilde asr 1) in

```

```

let stop3    = ref (!stop1 - !j)                (* L > R *)
and stop2    = ref (noGammas - nTilde + 1 + !j) in (* L = R *)
    stop1 := !stop1 - 1;                          (* L < R *)

    (*****)
    (* Lift Lambda values (even guys) *)
    (*****)
    let lcPtr = ref 0      (* position lifting pointer *)
    (* Cases where nbr. left Lambdas < nbr. right Lambdas *)
    and vG = ref (stepIncr asr 1) (* second coefficient is Gamma *)
    and vL = ref 0          in
    while !stop1 <> 0 do
        stop1 := !stop1 - 1;
        vL := 0; (* lambda starts always in first coefficient *)
        j := nTilde;
        let (x0,y0) = coord !vL dir
        and (x1,y1) = coord !vG dir in
        vect.{x0,y0} <- vect.{x0,y0} +.
            vect.{x1,y1} *. lc.{!lcPtr}; (* lift
            Lambda (Lambda + lifting value) *)
        lcPtr := !lcPtr + 1;
        vL := !vL + stepIncr; (* jump to next Lambda coefficient *)
        j := !j - 1;
        while !j <> 0 do (* use all nTilde lifting coefficients *)
            let (x0,y0) = coord !vL dir
            and (x1,y1) = coord !vG dir in
            vect.{x0,y0} <- vect.{x0,y0} +.
                vect.{x1,y1} *. lc.{!lcPtr};
                (* lift Lambda (Lambda + lifting value) *)
            lcPtr := !lcPtr + 1;
            vL := !vL + stepIncr; (* jump to next Lambda coefficient *)
            j := !j - 1;
        done;
        (* Go to next Gamma coefficient *)
        vG := !vG + stepIncr;

    done;

    (* Cases where nbr. left Lambdas = nbr. right Lambdas *)
    let soi = ref 0 in

    while !stop2 <> 0 do
        stop2 := !stop2 - 1;
        vL := !soi; (* first Lambda to be read *)
        j := nTilde;
        let (x0,y0) = coord !vL dir
        and (x1,y1) = coord !vG dir in
        vect.{x0,y0} <- vect.{x0,y0} +. vect.{x1,y1} *. lc.{!lcPtr};

```

```

(* lift Lambda (Lambda + lifting value) *)
lcPtr := !lcPtr + 1;
vL := !vL + stepIncr; (* jump to next Lambda coefficient *)
j := !j - 1;
while !j <> 0 do      (* use all nTilde lifting coefficients *)
  let (x0,y0) = coord !vL dir
  and (x1,y1) = coord !vG dir in
    vect.{x0,y0} <- vect.{x0,y0} +. vect.{x1,y1} *. lc.{!lcPtr};
    (* lift Lambda (Lambda + lifting value) *)
    lcPtr := !lcPtr + 1;
    vL := !vL + stepIncr; (* jump to next Lambda coefficient *)
    j := !j - 1;
done;
(* Go to next Gamma coefficient *)
vG := !vG + stepIncr;
(* Move start point for the Lambdas *)
soi := !soi + stepIncr;

done;

(* Cases where nbr. left Lambdas = nbr. right Lambdas *)
let vet = !soi - stepIncr in (* first Lambda is always in
                               this place *)

while !stop3 <> 0 do
  stop3 := !stop3 - 1;
  vL := vet; (* position Lambda pointer *)
  j := nTilde;
  let (x0,y0) = coord !vL dir
  and (x1,y1) = coord !vG dir in
    vect.{x0,y0} <- vect.{x0,y0} +. vect.{x1,y1} *. lc.{!lcPtr};
    (* lift Lambda (Lambda + lifting value) *)
    lcPtr := !lcPtr + 1;
    vL := !vL + stepIncr; (* jump to next Lambda coefficient *)
    j := !j - 1;
  while !j <> 0 do (* use all nTilde lifting coefficients *)
    let (x0,y0) = coord !vL dir
    and (x1,y1) = coord !vG dir in
      vect.{x0,y0} <- vect.{x0,y0} +. vect.{x1,y1} *. lc.{!lcPtr};
      (* lift Lambda (Lambda + lifting value) *)
      lcPtr := !lcPtr + 1;
      vL := !vL + stepIncr;
      (* jump to next Lambda coefficient *)
      j := !j - 1;
  done;
  (* Go to next Gamma coefficient *)
  vG := !vG + stepIncr;
  (* Move start point for the Lambdas *)

```

```

done
;;

(*
* FLWT2D function: This is a 2D Direct/Inverse Fast Lifted Wavelet
* Trasform that uses the "Square 2D" method to find the Gamma
* coefficients of the wavelet function Phi in the Direct case
* and the Lambda coefficients of the original signal in the
* Inverse case. The "Square 2D" method applies the transform
* first to all the rows and then to all the columns. This creates
* squared window transforms for the process. Since the Lifting
* Scheme is used, the final coefficients are organized
* in-place in the original matrix Data[[]]. In file mallat.c
* there are functions provided to change this organization to
* the Isotropic format originally established by Mallat.
*)

let flwt2D data width height n nTilde levels inverse =
  (* Verify for existence of filter and lifting coefficients *)
  if (Array1.dim !filter) = 0
  then raise No_Filter_Vector;

  (* O código abaixo verifica se os vetores de lifting foram
  inicializados. Depois, encontrar uma maneira de fazer isto e
  também de filter.

  if ( !liftingX && !liftingY ) {
    fprintf( stderr,
              "\nFLWT2D(): lifting coefficients are not initialized."
              "\nUse GetDual() and GetReal() before calling this function.\n");
    exit( INPUT_ERROR );
  }
  *)

  (* Calculate number of iterations n. It is the maximum *)
  (* value such that the following relation is satisfied. *)
  (*  $(2^n) * (N-1) \leq L < (2^{(n+1)}) * (N-1) + 1$  *)
  (* Where  $L = \max$  (signal's length in X-Y direction). *)
  (* and  $N = \max$  (# dual vanish mom & # real vanish mom) *)
  (* Hence, solving for n, we have the following equation *)
  (* for all the cases:  $n = \text{floor}(\log_2((L-1)/(N-1)))$  *)
  let maxN = (max n nTilde) - 1 in (* max vanishing moments *)
  (* Iterations in X *)
  let nX = if (width = 1) then 0
  else (int_of_float (logBaseN (float ((width-1)/maxN)) 2.0 )) in
  let nX = ref (if (nX < 0) then 0 else min levels nX)
  (* find lowest level *)

```

```

(* Iterations in Y *)
and nY = if (height = 1) then 0
else (int_of_float (logBaseN (float ((height-1)/maxN)) 2.0 )) in
let nY = ref (if (nY < 0) then 0 else min levels nY) in
(* find lowest level *)
(* Total number of iterations *)
let n' = ref (max !nX !nY) in (* find minimum number of iterations *)

if not inverse    (* Forward transform *)
then begin
  (* Calculate wavelet coefficients *)
  let max_step = if (!n' = 0) then 0 else (1 lsl !n') (* maximum step *)
  and step = ref 1 in

  while !step < max_step do
    if !nX > 0
    then begin
      nX := !nX - 1;
      (* Apply forward transform to the rows *)
      let y = ref 0 in
      while !y < height do
        flwt1D_Predict data (Lin !y) width !step n;
        flwt1D_Update data (Lin !y) width !step nTilde
          (Array2.slice_left !liftingX !nX);
        y := !y + !step
      done;
    end
    else nX := !nX - 1;

    if !nY > 0
    then begin
      nY := !nY - 1;
      (* Apply forward transform to the columns *)
      let x = ref 0 in
      while !x < width do
        flwt1D_Predict data (Col !x) height !step n;
        flwt1D_Update data (Col !x) height !step nTilde
          (Array2.slice_left !liftingY !nY);
        x := !x + !step
      done;
    end
    else nY := !nY - 1;

    step := !step lsl 1
  done;
end
else begin    (* Inverse transform *)
  (* Switch sign of filter & lifting coefficients *)

```

```

for i = 0 to n * (1+(n asr 1)) - 1 do
    !filter.{i} <- -. !filter.{i} (* negate filter *)
done;
for x = 0 to !nX - 1 do
    for k = 0 to (width asr 1)*nTilde - 1 do
        !liftingX.{x,k} <- -. !liftingX.{x,k}
        (* negate lifting in X *)
    done;
done;
for y = 0 to !nY - 1 do
    for k = 0 to (height asr 1)*nTilde - 1 do
        !liftingY.{y,k} <- -. !liftingY.{y,k}
        (* negate lifting in Y *)
    done;
done;

(* Find original dataset *)
let max_step = if (!n' = 0) then 0
else (1 lsl (!n' - 1)) in (* initial step *)
let step = ref max_step in

while !step >= 1 do
    if !n' <= !nY
    then begin
        (* Apply inverse transform to the columns *)
        let x = ref 0 in
        while !x < width do
            flwt1D_Update data (Col !x) height !step nTilde
                        (Array2.slice_left !liftingY (!nY - !n'));
            flwt1D_Predict data (Col !x) height !step n;
            x := !x + !step
        done;
    end;

    if !n' <= !nX
    then begin
        (* Apply inverse transform to the rows *)
        let y = ref 0 in
        while !y < height do
            flwt1D_Update data (Lin !y) width !step nTilde
                        (Array2.slice_left !liftingX (!nX - !n'));
            flwt1D_Predict data (Lin !y) width !step n;
            y := !y + !step
        done;
    end;

    n' := !n' - 1; (* one less level to go *)

```

```

        step := !step asr 1
    done;

    (* Switch sign of filter & lifting coefficients again *)
    for i = 0 to n*(1+(n asr 1)) - 1 do
        !filter.{i} <- -. !filter.{i} (* negate filter *)
    done;
    for x = 0 to !nX - 1 do
        for k = 0 to (width asr 1)*nTilde - 1 do
            !liftingX.{x,k} <- -. !liftingX.{x,k}
            (* negate lifting in X *)
        done;
    done;
    for y = 0 to !nY - 1 do
        for k = 0 to (height asr 1)*nTilde - 1 do
            !liftingY.{y,k} <- -. !liftingY.{y,k}
            (* negate lifting in Y *)
        done;
    done;

end

;;

(*
* FLWT1D function: This is a 1D Direct/Inverse Fast Lifted Wavelet
* Transform. Since the Lifting Scheme is used, the final
* coefficients are organized in-place in the original
* vector Data[]. In file mallat.c there are
* functions provided to change this organization to
* the Isotropic format originally established by Mallat.
*)

let flwt1D data width n nTilde levels inverse =

(* Encontrar outra maneira!!!!!!!!!!!!!!
/* Verify for existence of filter and lifting coefficients */
if (!filter) {
    Error ("FLWT1D", NO_FILTER_VECTOR, ABORT);
}

if ( !liftingX && !liftingY ) {
fprintf( stderr,
        "\nFLWT1D(): lifting coefficients are not initialized."
        "\nUse GetDual() and GetReal() before calling this "
        "function.\n");
exit( INPUT_ERROR );

```

```

    }
*)
(* Calculate number of iterations n. It is the maximum *)
(* value such that the following relation is satisfied. *)
(*  $(2^n) * (N-1) \leq L < (2^{(n+1)}) * (N-1) + 1$  *)
(* Where  $L = \max$  (signal's length in X-Y direction). *)
(* and  $N = \max$  (# dual vanish mom & # real vanish mom) *)
(* Hence, solving for n, we have the following equation *)
(* for all the cases:  $n = \text{floor}(\log_2((L-1)/(N-1)))$  *)

let maxN = (max n nTilde) - 1 in (* max vanishing moments *)
(* Iterations in X *)
let nX = if (width = 1) then 0
else (int_of_float (logBaseN (float ((width-1)/maxN)) 2.0 )) in
let nX = ref (if (nX < 0) then 0 else min levels nX) in
(* find lowest level *)

if not inverse (* Forward transform *)
then begin
  (* Calculate wavelet coefficients *)
  let max_step = if (!nX = 0) then 0 else (1 lsl !nX)
  (* maximum step *)
  and step = ref 1
  and s = ref (!nX - 1) in

  while !step < max_step do
    (* Apply forward transform to the vector *)
    flwt1D_Predict data (Lin 0) width !step n;
    flwt1D_Update data (Lin 0) width !step nTilde
    (Array2.slice_left !liftingX !s) ;

    step := !step lsl 1;
    s := !s - 1
  done;
end
else begin (* Inverse transform *)
  (* Switch sign of filter & lifting coefficients *)
  for i = 0 to n*(1+(n asr 1)) - 1 do
    !filter.{i} <- -. !filter.{i} (* negate filter *)
  done;
  for x = 0 to !nX - 1 do
    for k = 0 to (width asr 1)*nTilde - 1 do
      !liftingX.{x,k} <- -. !liftingX.{x,k}
      (* negate lifting in X *)
    done;
  done;
done;

(* Find original dataset *)

```

```

    let max_step = if (!nX = 0) then 0
    else (1 lsl (!nX - 1)) in (* initial step *)
    let step = ref max_step
    and s = ref 0 in

    while !step >= 1 do
      (* Apply inverse transform to the vector *)
      flwt1D_Update data (Lin 0) width !step nTilde
                    (Array2.slice_left !liftingX !s);
      flwt1D_Predict data (Lin 0) width !step n;

      step := !step asr 1;
      s := !s + 1
    done;

    (* Switch sign of filter & lifting coefficients again *)
    for i = 0 to n*(1+(n asr 1)) - 1 do
      !filter.{i} <- -. !filter.{i} (* negate filter *)
    done;
    for x = 0 to !nX - 1 do
      for k = 0 to (width asr 1)*nTilde - 1 do
        !liftingX.{x,k} <- -. !liftingX.{x,k}
        (* negate lifting in X *)
      done;
    done;
  end
;;

(* CODE FOR THE IN-PLACE FLWT (AVERAGE INTERPOLATION CASE, N ODD) *)

(*
* FLWT_Haar function: finds the Haar and Lambda coefficients for
* the given 1D signal. Boundary cases are not
* considered yet. Thus, all signals must be
* of even lengths.
*)

let flwt_Haar vect dir size step =
  (*****)
  (* Calculate values of some important variables *)
  (*****)
  let len = ceil2 size step (* number of coefficients at current
                             level *)
  and stepIncr = step lsl 1 in (* step size between coefficients *)
  (*****)
  (* Calculate number of iterations *)
  (*****)
  let nG = ref (len asr 1) in

```

```

let nL = len - !nG
and lAve = ref 0.0 in

  (*****)
  (* Calculate Lambda average *)
  (*****)
  if odd len
  then begin
    let n = ref len
    and s = stepIncr asr 1
    and t = ref 0.0
    and l = ref 0 in
    while !n <> 0 do
      n := !n - 1;
      let (x0,y0) = coord !l dir in
      t := !t +. vect.{x0,y0};
      l := !l + s;
    done;
    lAve := !t /. (float len) (* average we've to maintain *)
  end;

  (*****)
  (* Calculate Gamma (Haar wavelet) coefficients (odd guys) *)
  (* and coarser Lambda coefficients (even guys) *)
  (*****)
  let idxG = ref (stepIncr asr 1) (* second coefficient is Gamma *)
  and idxL = ref 0 (* first Lambda to be read *)
  and lSum = ref 0.0 in
  while !nG <> 0 do
    nG := !nG - 1;
    (* New Gamma (Haar) coefficient *)
    let (x0,y0) = coord !idxG dir
    and (x1,y1) = coord !idxL dir in
    vect.{x0,y0} <- vect.{x0,y0} -. vect.{x1,y1};
    (* New Lambda coefficient *)
    vect.{x1,y1} <- vect.{x1,y1} +. vect.{x0,y0} /. 2.0;
    lSum := !lSum +. vect.{x1,y1};

    (* Go to next Gamma coefficient *)
    idxG := !idxG + stepIncr;

    (* Go to next Lambda coefficient *)
    idxL := !idxL + stepIncr;

  done;

  (*****)
  (* If ODD(len), then we have one additional case for *)

```

```

(* the Lambda calculations. This is a boundary case *)
(* and, therefore, has different filter values. *)
(*****)
  if odd len
  then begin
    let (x1,y1) = coord !idxL dir in
    vect.{x1,y1} <- !lAve *. (float nL) -. !lSum;
    lSum := !lSum +. vect.{x1,y1}
  end
;;

(*
* IFLWT_Haar function: applies the inverse transform to find
*   Lambda coefficients for the given 1D signal. Boundary
*   cases are not considered yet. Thus, all signals
*   must be of even lengths.
*)

let iflwt_Haar vect dir size step =
  (*****)
  (* Calculate values of some important variables *)
  (*****)
  let len = ceil2 size step (* number of coefficients at current
                             level *)
  and stepIncr = step lsl 1 in (* step size between coefficients *)

  (*****)
  (* Calculate number of iterations *)
  (*****)
  let nG = ref (len asr 1) in
  let nL = len - !nG
  and lAve = ref 0.0 in

  (*****)
  (* Calculate Lambda average *)
  (*****)
  if odd len
  then begin
    let n = ref nL
    and s = stepIncr
    and t = ref 0.0
    and l = ref 0 in
    while !n <> 0 do
      n := !n - 1;
      let (x0,y0) = coord !l dir in
      t := !t +. vect.{x0,y0};
      l := !l + s;
    end
  end
end

```

```

    done;
    lAve := !t /. (float nL)    (* average we've to maintain *)
end;

(*****)
(* Calculate Gamma (Haar wavelet) coefficients (odd guys) *)
(* and coarser Lambda coefficients (even guys) *)
(*****)
let idxG = ref (stepIncr asr 1) (* second coefficient is Gamma *)
and idxL = ref 0                (* first Lambda to be read *)
and lSum = ref 0.0              in
while !nG <> 0 do
    nG := !nG - 1;
    (* New Lambda coefficient *)
    let (x0,y0) = coord !idxL dir
    and (x1,y1) = coord !idxG dir    in
    vect.{x0,y0} <- vect.{x0,y0} -. vect.{x1,y1} /. 2.0;
    lSum := !lSum +. vect.{x0,y0};
    (* New Gamma (Haar) coefficient *)
    vect.{x1,y1} <- vect.{x1,y1} +. vect.{x0,y0};
    lSum := !lSum +. vect.{x1,y1};

    (* Go to next Gamma coefficient *)
    idxG := !idxG + stepIncr;
    (* Go to next Lambda coefficient *)
    idxL := !idxL + stepIncr;

done;

(*****)
(* If ODD(len), then we have one additional case for *)
(* the Lambda calculations. This is a boundary case *)
(* and, therefore, has different filter values. *)
(*****)
if odd len
then begin
    let (x0,y0) = coord !idxL dir in

    vect.{x0,y0} <- !lAve *. (float len) -. !lSum;
    lSum := !lSum +. vect.{x0,y0}
end
;;

(*
* FLWT2D_Haar function: calculates the Haar wavelet coefficients
* for a two dimensional signal. This case corresponds to

```

```

*          average interpolation with N=1 and wavelets with
*          nTilde = 1.
*)

let flwt2D_Haar data width height levels inverse =
  (* Number of iterations *)
  let nX = if (width = 1) then 0
  else (int_of_float (ceil (logBaseN (float width) 2.0 ))) in
  let nX = ref (if (nX < 0) then 0 else min levels nX)
  (* find lowest level *)
  (* Iterations in Y *)
  and nY = if (height = 1) then 0
  else (int_of_float (ceil (logBaseN (float height) 2.0 ))) in
  let nY = ref (if (nY < 0) then 0 else min levels nY) in
  (* find lowest level *)
  (* Total number of iterations *)
  let n' = ref (max !nX !nY) in
  (* find minimum number of iterations *)

  if not inverse (* Forward transform *)
  then begin
    (* Calculate wavelet coefficients *)
    let max_step = if (!n' = 0) then 0 else (1 lsl !n') (* maximum step *)
    and step = ref 1 in

    (* Find lifting coefficients for all levels *)
    while !step < max_step do
      if !nX > 0
      then begin
        nX := !nX - 1;
        (* Apply forward transform to the rows *)
        let y = ref 0 in
        while !y < height do
          flwt_Haar data (Lin !y) width !step;
          y := !y + !step
        done;
      end
      else nX := !nX - 1;

      if !nY > 0
      then begin
        nY := !nY - 1;
        (* Apply forward transform to the columns *)
        let x = ref 0 in
        while !x < width do
          flwt_Haar data (Col !x) height !step;
          x := !x + !step
        done;
      end
    end
  end

```

```

        end
        else nY := !nY - 1;
        step := !step lsl 1
    done;
end
else begin      (* Inverse transform *)
    (* Find original dataset *)
    let max_step = if (!n' = 0) then 0 else (1 lsl (!n' - 1)) in
    (* initial step *)
    let step = ref max_step in

    while !step >= 1 do
        if !n' <= !nY
        then begin
            (* Apply inverse transform to the columns *)
            let x = ref 0 in
            while !x < width do
                iflwt_Haar data (Col !x) height !step;
                x := !x + !step
            done;
        end;

        if !n' <= !nX
        then begin
            (* Apply inverse transform to the rows *)
            let y = ref 0 in
            while !y < height do
                iflwt_Haar data (Lin !y) width !step;
                y := !y + !step
            done;
        end;

        n' := !n' - 1; (* one less level to go *)

        step := !step asr 1
    done;

end

;;

(*
* FLWT1D_Haar function: calculates the Haar wavelet coefficients
* for a one dimensional signal. This case corresponds
* to average interpolation with N=1 and wavelets
* with nTilde = 1.
*)

```

```

let flwt1D_Haar data width levels inverse =
  (* Number of iterations *)
  let nX = if (width = 1) then 0
  else (int_of_float (ceil (logBaseN (float width) 2.0))) in
  let nX = if (nX < 0) then 0 else min nX levels in
  (* find lowest level *)

  if not inverse (* Forward transform *)
  then begin
    (* Calculate wavelet coefficients *)
    for s = 0 to nX - 1 do
      (* Apply forward transform to the vector *)
      flwt_Haar data (Lin 0) width (1 lsl s)
    done
  end
  else begin (* Inverse transform *)
    (* Find original dataset *)
    for s = nX - 1 downto 0 do
      (* Apply inverse transform to the vector *)
      iflwt_Haar data (Lin 0) width (1 lsl s)
    done
  end
end
;;

```

D.15 Ondaleta.ml

```

(*****
(*
(*      Transformada discreta wavelet CDF(2,2)
(*
(*      Autor : Alessandro Soares e-mail: a_s_soares@yahoo.com.br
(*      Criado em: 17/10/2003   17:00
(*      Modificado:
(*
(*
(*****)

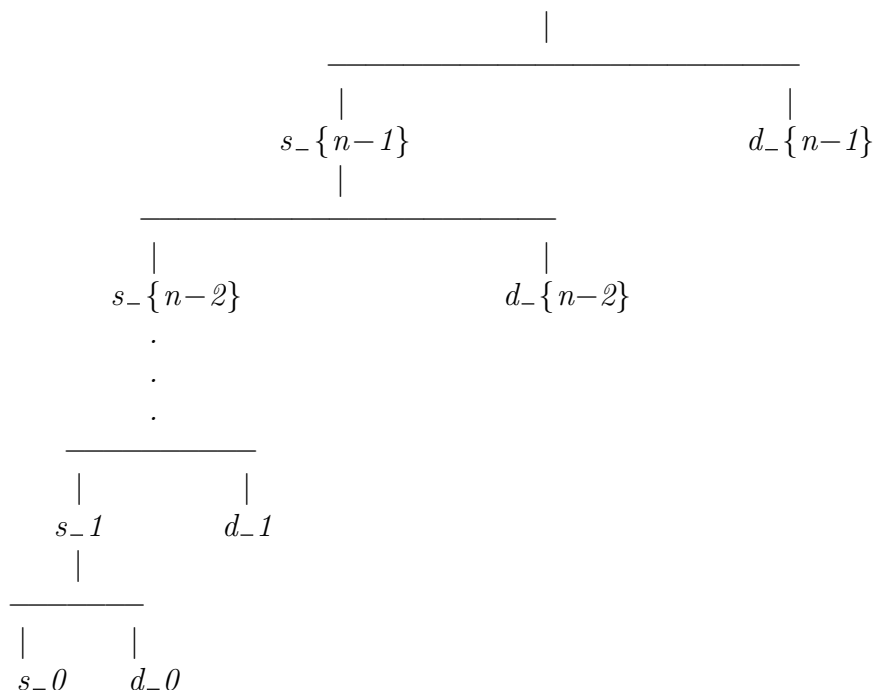
(* O sinal s_n possui 2^n pontos s_n, l:

```

$$s_n = \{s_n, l \mid 0 \leq l < 2^n\}$$

Assim, podemos decompor o sinal em até n níveis, sendo n o nível do sinal original e 0 o nível de resolução mais fino (coarsest level).

s_n



A transformada implementada neste exemplo é a wavelet linear, também chamada de waveletbiortogonal $(2,2)$ de Cohen–Daubechies–Feauveau, abreviada $CDF(2,2)$.

Para lidar com as bordas do sinal, assume-se que ele é periódico. Desta forma, se a sequência do sinal original for

$$\{a_0, a_1, \dots, a_n\}$$

o programa assume

$$\{\dots, a_0, a_1, \dots, a_n, a_0, a_1, \dots, a_n, a_0, a_1, \dots\}.$$

Outras soluções são possíveis e serão acrescentadas, se necessário.

O número máximo de pontos na sequência é 65536 (2^{16}). Se valores maiores forem necessários, acrescentar as potências de 2 respectivas ao vetor *incr*.

Exemplos de uso:

```
# let vet = [| 4.0; 6.0; 10.0;12.0;8.0;6.0;2.0;4.0|];;
```

Para achar todos os coeficientes da wavelet até o nível de resolução mais fino, use:

```
# dwt vet;;
```

O resultado modificará o vetor *vet*. Assim,

```
# vet;;
```

```
- : float array = [| 6.5; -1.; 4.; 3.; 5.; 1.; -4.; 1.|]
```

Se desejar obter todos os coeficientes da wavelet na ordem $\{s_0, d_0, d_1, \dots, d_{n-1}\}$ faça:

```
# getAllCoefs vet;; (* esta operação cria um novo
                      vetor e não altera vet *)
- : float array = [|6.5; 5.; 4.; -4.; -1.; 3.; 1.; 1.|]
```

Se quiser retornar ao sinal original:

```
# idwt vet;;
# vet;;
- : float array = [|4.; 6.; 10.; 12.; 8.; 6.; 2.; 4.|]
```

Suponha que se deseje decompor o sinal até determinado nível. Para o sinal do exemplo, desejamos decompô-lo até o nível 2, a partir do nível do sinal, que é 3. Isto pode ser feito por:

```
# waveAnalysis vet 3 2;;
# vet;;
- : float array = [|4.; -1.; 10.5; 3.; 9.; 1.; 2.5; 1.|]
```

Se desejar separar a parte resultante do filtro passa-baixa no nível 2 (s_2), use:

```
# let s2 = getLowPass vet 2;; (* esta operação cria um novo
                              vetor e não altera vet *)
val s2 : float array = [|4.; 10.5; 9.; 2.5|]
```

O mesmo pode ser feito para o passa-alta (d_2):

```
# let d2 = getHighPass vet 2;;
val d2 : float array = [| -1.; 3.; 1.; 1.|]
```

Se desejar recompor o sinal a partir do nível 2 para o nível 3, use:

```
# waveSintthesis vet 2 3;;
# vet;;
- : float array = [|4.; 6.; 10.; 12.; 8.; 6.; 2.; 4.|]
```

Desejando analisar nível a nível os valores resultantes dos filtros passa-baixa e passa-alta, faça o seguinte:

```
# dwtByLevel vet;;
```

Isto fará com o sinal seja analisado até o próximo nível, que no caso do exemplo é o nível 2.

```
# vet;;
- : float array = [|4.; -1.; 10.5; 3.; 9.; 1.; 2.5; 1.|]
```

Separando-se o resultado como fizemos anteriormente:

```
# let s2 = getLowPass vet 2;;
val s2 : float array = [|4.; 10.5; 9.; 2.5|]
```

```

# let d2 = getHighPass vet 2;;
val d2 : float array = [| -1.; 3.; 1.; 1. |]

Se quisermos avaliar o próximo nível (nível 1) precisamos usar s2:
# dwfByLevel s2;;

que separado produz:
# let s1 = getLowPass s2 1;;
val s1 : float array = [| 4.; 9. |]
# let d1 = getHighPass s2 1;;
val d1 : float array = [| 4.; -4. |]

Analogamente para o nível 0:
# dwfByLevel s1;;

# let s0 = getLowPass s1 0;;
val s0 : float array = [| 6.5 |]
# let d0 = getHighPass s1 0;;
val d0 : float array = [| 5. |]

*)

let maxLevel = ref 0 ;;

let log2 x = int_of_float (log10 (float x) /. log10 2.) ;;
let pow2 x = int_of_float (2.0 ** (float x));;

let incr = [| 1; 2; 4; 8; 16; 32; 64; 128; 256; 512; 1024; 2048;
              4096; 8192; 16384; 32768; 65536 |]

let dAdress(j,l) =
  let k = (l + incr.(j)) mod incr.(j) in
  (*      offset      +      increment      *)
  incr.(!maxLevel - (j+1)) + k*incr.(!maxLevel - j)

(* to handle wrap around:
   number of data in level j is 2^j
   (already available in INCR)
*)
let sAdress(j,l) =
  let k = (l + incr.(j)) mod incr.(j) in
  k*incr.(!maxLevel - j) (* increment *)

let p s (j,i) =
  (* linear interpolation (order = 2) *)
  (s.(sAdress(j-1,i)) +. s.(sAdress(j-1,i+1))) /. 2.0

```

(O argumento j representa o nível de resolução mais fino *)*

```

let predictor s j =
  for i = 0 to incr.(j-1) - 1 do
    let d = dAddress(j-1,i) in
    s.(d) <- s.(d) -. p s (j,i)
  done
;;

let u s (j,i) =
  (* linear interpolation (order = 2) *)
  0.25 *. s.(dAddress(j-1,i-1)) +. 0.25 *. s.(dAddress(j-1,i))

let update s j =
  for i = 0 to incr.(j-1) - 1 do
    let s' = sAddress(j-1,i) in
    s.(s') <- s.(s') +. u s (j,i)
  done
;;

let undoUpdate s j =
  for i = 0 to incr.(j-1) - 1 do
    let s' = sAddress(j-1,i) in
    s.(s') <- s.(s') -. u s (j,i)
  done
;;

```

```

let undoPredict s j =
  for i = 0 to incr.(j-1) - 1 do
    let d = dAddress(j-1,i) in
    s.(d) <- s.(d) +. p s (j,i)
  done
;;

```

(Aplica a transformada wavelet direta um nível por vez *)*

```

let dwtByLevel s =
  maxLevel := log2 (Array.length s);
  predictor s !maxLevel ;
  update      s !maxLevel
;;

```

(Aplica a transformada wavelet inversa um nível por vez *)*

```

let idwtByLevel s =

```

```

    maxLevel := log2 (Array.length s);
    predictor s 1;
    update s 1
;;

(* Aplica a transformada wavelet direta até o nível 1.
   Assume que os dados possuem 2^maxLevel pontos
   *)
let dwt s =
    maxLevel := log2 (Array.length s);
    for j = !maxLevel downto 1 do
        predictor s j;
        update s j
    done
;;

(* Aplica a transformada wavelet inversa do nível 1 até o nível
   máximo (o sinal reconstruído).
   Assume que os dados possuem 2^maxLevel pontos
   *)
let idwt s =
    maxLevel := log2 (Array.length s);
    for j = 1 to !maxLevel do
        undoUpdate s j;
        undoPredict s j;
    done
;;

(* Aplica a transformada wavelet ao sinal s no nível n até o nível
   n1. *)
let waveAnalysis s n n1 =
    maxLevel := log2 (Array.length s);
    for j = n downto (n1+1) do
        predictor s j;
        update s j
    done
;;

(* Aplica a transformada wavelet inversa ao sinal s no nível n1
   até o nível n. *)
let waveSynthesis s n1 n =
    maxLevel := log2 (Array.length s);
    for j = n1+1 to n do
        undoUpdate s j;
        undoPredict s j;
    done

```

```
;;
```

```
(* Pega os coeficientes de detalhes da wavelet no nível j
em ordem *)
```

```
let getDetailCoef s j =
  let ndata = pow2 j      in
  let y      = Array.create ndata 0.0 in
    for i = 0 to ndata - 1 do
      let d = dAdress(j,i) in
        y.(i) <- s.(d)
    done;
  y
```

```
;;
```

```
(* Pega os coeficientes do alisamento da wavelet no nível j
em ordem *)
```

```
let getSmoothCoef s j =
  let ndata = pow2 j      in
  let y      = Array.create ndata 0.0 in
    for i = 0 to ndata - 1 do
      let s' = sAdress(j,i) in
        y.(i) <- s.(s')
    done;
  y
```

```
;;
```

```
(* Pega todos os coeficientes da wavelet em ordem *)
```

```
let getAllCoefs s =
  let ndata = incr.(!maxLevel) in
  let y = Array.create ndata 0.0
  and k = ref 1                in
    y.(0) <- s.(0) ;
    for j = 0 to !maxLevel - 1 do
      for i = 0 to (incr.(j)) - 1 do
        let d = dAdress(j,i) in
          y.(!k) <- s.(d) ;
          k := !k + 1
        done;
      done;
    done;
  y
```

```
;;
```

```
(* As duas funções abaixo deveriam ser chamadas após dwfByLevel s,
onde s é o sinal *)
```

```
(* Pega a saída passa-baixa do filtro wavelet no nível j *)  
let getLowPass s j = getSmoothCoef s j;;  
  
(* Pega a saída passa-alta do filtro wavelet no nível j *)  
let getHighPass s j = getDetailCoef s j;;
```

