UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE COMUNICAÇÕES

TESE

submetida como parte

dos requisitos para a obtenção do

Título de Doutor em Engenharia Elétrica

SOBRE A ESTABILIDADE NUMÉRICA DOS ALGORITMOS DE MÍNIMOS QUADRADOS RÁPIDOS

AUTOR: JOSÉ ROBERTO BOLLIS GIMENEZ

ORIENTADOR: JOÃO MARCOS TRAVASSOS ROMANO

JANEIRO DE 1995

RESUMO

O trabalho apresentado nesta tese é dedicado ao entendimento e busca de soluções para o problema de instabilidade numérica inerente dos algoritmos de mínimos quadrados rápidos. A ênfase do tratamento é dirigida a viabilizar e aperfeiçoar a aplicação destes algoritmos em sistemas de processamento digital de sinais, especialmente na área de transmissão de dados, onde os fenômenos de ocorrência de eco e interferência inter-simbólica exigem procedimentos de filtragem adaptativa em tempo real.

Os métodos de mínimos quadrados clássico e recursivo são tratados no Capítulo 1, juntamente com os processos de filtragem digital adaptativa aos quais eles se aplicam. Os algoritmos rápidos são introduzidos no Capítulo 2, onde se procurou elaborar um texto que reuna as principais informações existentes sobre o assunto na literatura. Estes dois primeiros capítulos utilizam uma abordagem clara e ao mesmo tempo inédita do funcionamento dos algoritmos de mínimos quadrados, representando uma contribuição no campo didático.

Contribuições importantes para o entendimento do problema da instabilidade numérica em vários algoritmos de mínimos quadrados são apresentadas no Capítulo 3. Diversas formas de análise são desenvolvidas para explicar a origem e a propagação do fenômeno da instabilidade numérica, incluindo um modelo que descreve o comportamento do processo corruptivo.

No Capítulo 4 são discutidos os principais métodos de estabilização existentes. Também é apresentado um método novo, que representa uma significativa evolução técnica na área, sobretudo quando se deseja atender um compromisso entre complexidade computacional e estabilidade numérica. Resultados de simulações e comparações do novo método com outros existentes são apresentados no Capítulo 5, com o objetivo de validar o novo método como um instrumento adequado ao controle da instabilidade numérica nos algoritmos rápidos.

ABSTRACT

The work described in this thesis is concerned with the understanding and solution of the numerical instability problem presented by the fast least square algorithm. The emphasis is on the application and performance achieved by the algorithms in digital signal processing systems, specially in data transmission, where the echo phenomena and intersymbol interference effects makes the on-line adaptive filtering process a mandatory technique.

The traditional and recursive least square methods are presented in Chapter 1, so are their applications on the adaptive digital filtering process. The fast least square algorithms are introduced in Chapter 2. In this chapter we gathered the important information found in the technical literature about the topic. These two first chapters represent a contribution in the didactic area, as they present a clear and original approach to the least square algorithm operation.

In Chapter 3, we present important contributions to the understanding of the numerical instability problem in various least square algorithms. Also, we develop several kinds of theoretical and experimental analysis in order to explain the origin and growing of the numerical instability phenomenon, including a mathematical model that describes the behaviour of the corruptive process.

In Chapter 4, we discusse the existing stabilization methods. Also, we propose a new method, which represents a significant technical improvement in the area, mainly if the goal is a tradeoff between computational complexity and numerical stability. A series of simulation results and comparisons between the new method and the existing ones are presented in Chapter 5 with the objective of validating the proposed method as an adequate tool to prevent the numerical instability in the fast least square algorithms.

PREFÁCIO

Imagine-se uma pequena fábrica onde um funcionário manufatura um único tipo de peça, a qual exige que se tenha um molde para sua confecção. O funcionário, no entanto, prefere utilizar como amostra a última peça fabricada, trocando o modelo a cada nova reprodução. Copiando cada detalhe da peça, ele consegue manter um razoável controle de qualidade, garantindo a perfeita reprodução de cada peça. Obviamente, uma cópia sempre guarda uma pequena diferença de seu original, que pode mesmo ser imperceptível, dependendo da qualidade empregada no processo de reprodução.

Imagine-se agora que esta fábrica tenha passado anos trabalhando, e já tenha feito muitos milhares daquelas peças, usando sempre o mesmo procedimento. Ou seja, o funcionário toma como modelo a peça recém fabricada. Parece claro que se for comparada uma peça da fabricação atual com a primeira que foi feita, serão encontradas diferenças enormes. Apesar de funcionar satisfatoriamente para uma única cópia, este método de reprodução não se presta para fazer um número grande de cópias. Isto porque as imperfeições que ocorrem em cada nova cópia se acumulam ao ponto de produzir distorções aparentes. Logicamente, o resultado seria outro, caso o funcionário reservasse uma peça exclusivamente para servir de modelo. A fidelidade da reprodução estaria garantida para todas as peças fabricadas.

Analogamente ao processo de produção descrito acima, existem cálculos matemáticos que se utilizam da técnica de recursividade, tomando como variáveis de trabalho resultados recém obtidos. E assim, continuamente, substituem as variáveis de entrada pelos novos números que acabam de ser calculados. Da mesma forma que o processo de cópia introduz pequenas imperfeições na nova peça, os resultados de um cálculo computacional também apresentam pequenos erros, devido ao limite de precisão imposto pela máquina de calcular. Os pequenos erros, quando realimentados continuamente podem alcançar valores significativos. Este caso

acontece especificamente com os algoritmos de mínimos quadrados rápidos. A conseqüência do acúmulo de erros nestes algoritmos é um fenômeno de descontrole que se manifesta de uma forma brusca. Por esta razão, eles são tidos como numericamente instáveis.

Os algoritmos de mínimos quadrados rápidos representam um tipo de solução para uma classe de problemas de otimização conhecida como método dos mínimos quadrados. Esta solução apresenta a vantagem de ser implementada com um número mínimo de operações matemáticas, sendo, por isso, indicada para situações que requerem processamento em tempo real. O maior entrave para a aplicação prática destes algoritmos reside na sua inerente característica de instabilidade numérica.

Este trabalho tem por objetivo analisar e propor soluções ao problema de instabilidade numérica dos algoritmos rápidos. Uma vez estabilizados, estes algoritmos encontram aplicação imediata em processos de: estimação e predição de sinais, identificação de sistemas, cancelamento de ecos, eliminação de interferência inter-simbólica, e demais processos que envolvam filtragem adaptativa.

Boa parte destes processos estão potencialmente relacionados com a área de transmissão digital de sinais. O interesse por temas nessa área foi, sem dúvida, a maior motivação para a realização deste trabalho.

Nesse sentido, quero expressar aqui meus sinceros agradecimentos ao Prof. João Marcos T. Romano pela proposição do problema e pelo constante incentivo e orientação que muito marcou a realização desta tese .

Gostaria também de agradecer a UNICAMP, pelo saudável ambiente de pesquisa que ela sempre proporcionou; a CAPES, pelo auxílio financeiro; o Leonardo, cujo auxílio em algumas simulações me permitiram não cair em conclusões errôneas; a Lila, pela digitação de alguns capítulos; os meus pais, por todo o apoio e dedicação dispensados em minha formação; e os meus colegas, amigos e familiares, cuja simpatia e compreensão sempre me foram muito preciosas.

Finalmente, quero agradecer de uma forma especial a Marysol e a Andrea, pelas incontáveis horas que abdicaram de nosso agradável convívio familiar em favor da realização deste trabalho. É a elas que eu o dedico.

José Roberto B. Gimenez

ÍNDICE

CAPÍTULO 1

O CRITÉRIO DE MÍNIMOS QUADRADOS

Neste capítulo vamos fazer uma introdução ao método dos mínimos quadrados, ou método LS (Least Square). Este método se aplica a inúmeros problemas de estimação encontrados nos mais diversos campos da ciência. Vamos nos ater unicamente ao âmbito da transmissão de informação na forma discreta. Mais especificamente, estaremos interessados em casos que envolvam filtragem adaptativa, como cancelamento de eco, eliminação da interferência inter-simbólica e identificação de sistemas. Estes casos serão tratados aqui de uma forma genérica, importando unicamente a solução fornecida pelo método LS. Uma abordagem mais elaborada dos problemas relacionados com a ocorrência de ecos e interferência inter-simbólica é feita por este autor em [1].

Um sério inconveniente da literatura concernente à filtragem adaptativa é a falta de uma notação padronizada para a representação matemática dos sinais, filtros e demais variáveis envolvidas. O tratamento exige uma grande quantidade de símbolos, sendo que, cada autor os define de uma forma diferente. Procurando não criar ainda outra linha de notação, adotaremos com poucas modificações a usada em [2]. Esta escolha se deve não apenas à simplicidade que ela oferece ao tratamento, não sobrecarregando a representação dos símbolos com o uso excessivo de sub-índices, diacríticos e caracteres em negrito, mas principalmente à facilidade de associação na maneira de representar os elementos.

Sempre que possível procuraremos generalizar o tratamento, mantendo os sinais no campo das variáveis complexas. Este cuidado é especialmente importante na medida em que permite a análise de sinais modulados. O emprego de variáveis complexas é a forma usualmente adotada na representação temporal de sinais do tipo passa-faixas. Para isso existe toda uma teoria de modulação, bem estabelecida, que determina a relação existente, entre os sinais modulados e sua correspondente representação complexa em banda-básica [3].

1.1 – O MÉTODO LS

Vamos introduzir o critério LS revendo um conhecido exemplo onde esse método é usado: a determinação dos coeficientes da série de Fourier.

Exemplo 1.1

Sejam $x_1(t), x_2(t), \ldots, x_N(t)$ funções ortogonais no intervalo de tempo $t_1 \leq t \leq t_2$, ou seja

$$\int_{t_1}^{t_2} x_k(t)\, x_j^*(t)\, dt = \begin{cases} 0 & \text{se } k \neq j \\ S_k & \text{se } k = j \end{cases} \tag{1.1}$$

onde S_k, $k = 1, 2, \ldots, N$, é a energia do sinal $x_k(t)$ no intervalo de tempo considerado e o superíndice * denota a operação conjugado de uma variável complexa. Queremos aproximar a função d(t) pela combinação linear

$$y(t) = \sum_{k=1}^{N} h_k\, x_k(t) \tag{1.2}$$

onde os h_k são os coeficientes da série de Fourier que desejamos estimar pelo método LS. Definimos então a função de erro

$$\varepsilon(t) = d(t) - y(t) \tag{1.3}$$

e o parâmetro a ser minimizado

$$E_N = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} |\varepsilon(t)|^2 dt \tag{1.4}$$

$$= \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \left| d(t) - \sum_{k=1}^{N} h_k\, x_k(t) \right|^2 dt \tag{1.5}$$

Nota-se que E_N representa a potência do erro no intervalo de tempo $[t_1, t_2]$. É possível mostrar que a função descrita por este parâmetro no espaço multidimensional das variáveis complexas h_k é convexa. Em outras palavras: a função descrita por E_N em termos dos h_k apresenta um único ponto de mínimo. Este fato decorre diretamente da função módulo ao quadrado em (1.5). A unicidade deste ponto de mínimo permite que façamos

$$\frac{\partial E_N}{\partial h_k} = 0 \qquad k = 1, 2, \ldots, N \tag{1.6}$$

A solução de (1.6) fornece os valores ótimos para os h_k. Isto é, os valores que minimizam E_N. Separando as partes reais e imaginárias dos h_k na forma

$$h_k = a_k + jb_k \qquad k = 1, 2, \ldots, N \qquad (1.7)$$

podemos reescrever (1.5) como

$$E_N = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} [d(t) - \sum_{m=1}^{N} (a_m + j\, b_m)\, x_m(t)]\, [d^*(t) - \sum_{i=1}^{N} (a_i - j\, b_i)\, x_i^*(t)]\, dt \qquad (1.8)$$

assim

$$\frac{\partial E_N}{\partial a_k} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \{- x_k(t)\, [d^*(t) - \sum_{i=1}^{N} h_i^*\, x_i^*(t)] - x_k^*(t)\, [d(t) - \sum_{m=1}^{N} h_m\, x_m(t)]\}\, dt \qquad (1.9a)$$

e

$$\frac{\partial E_N}{\partial b_k} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \{- j\, x_k(t)\, [d^*(t) - \sum_{i=1}^{N} h_i^*\, x_i^*(t)] + j\, x_k^*(t)\, [d(t) - \sum_{m=1}^{N} h_m\, x_m(t)]\}\, dt \qquad (1.9b)$$

Combinando as equações acima, usando (1.1) e (1.6), obtemos finalmente

$$h_k = \frac{\int_{t_1}^{t_2} x_k^*(t)\, d(t)\, dt}{\int_{t_1}^{t_2} |x_k(t)|^2\, dt} \qquad k = 1, 2, \ldots, N \qquad (1.10)$$

Outro importante resultado da teoria dos mínimos quadrados pode ser derivado substituindo (1.2), (1.3) e (1.6) nas equações (1.9a) e (1.9b):

$$\int_{t_1}^{t_2} \varepsilon(t)\, x_k^*(t)\, dt = 0 \qquad k = 1, 2, \ldots, N \qquad (1.11)$$

Este resultado é também conhecido como "princípio da ortogonalidade". Ele estabelece que a função de erro no método dos mínimos quadrados seja ortogonal a cada uma das componentes da combinação linear quando os h_k assumem os valores ótimos. Esse aspecto do problema ficará melhor esclarecido na Seção 2.7, com a introdução de conceitos geométricos.

Na análise clássica de Fourier, as funções $x_k(t)$ são exponenciais complexas e $(t_2 - t_1) \rightarrow \infty$, de forma que esse conjunto de funções sempre atende a condição de ortogonalidade imposta por (1.1). Essa condição é a responsável pela simplicidade do resultado em (1.10). Vale notar que neste exemplo os h_k dependem unicamente da função desejada, d(t), e da respectiva componente, $x_k(t)$. A seguir analisaremos um segundo exemplo, com um outro tipo de restrição quanto à correlação entre as componentes $x_k(t)$, que não a ortogonalidade. Antes porém, vamos fazer algumas considerações sobre a representação dos sinais.

As variáveis acima definidas podem estar representando sinais contínuos no tempo, como por exemplo: o nível de tensão em determinado ponto de um sistema de transmissão ou recepção. Para nossos propósitos, no entanto, é mais conveniente o uso de sinais amostrados ao invés de sinais contínuos; pela facilidade que oferecem à aplicação em sistemas computacionais. Assim, passaremos agora a representar os sinais por seqüências de amostras complexas, como em $\{x(n)\}$, onde x(n) denota a amostra do sinal x(t) no instante $t = nT$, e T representa o período de amostragem do sinal.

Exemplo 1.2

Vamos determinar os coeficientes $h_k^*(n)$ tal que a seqüência $\{d(i)\}$ seja aproximada pela combinação linear

$$y(i,n) = \sum_{k=0}^{N-1} h_k^*(n)\, x(i-k) \qquad 1 \le i \le n \qquad (1.12)$$

Este problema constitui o tema central de nosso estudo e é semelhante ao visto no Exemplo 1.1. Desta vez, porém, não estamos impondo que as componentes da combinação linear sejam ortogonais. Por outro lado, elas agora são deslocamentos de uma mesma seqüência, $\{x(i)\}$. Também a janela de tempo, definida no Exemplo 1.1 como $[t_1, t_2]$, foi aqui modificada para os instantes de tempo i tais que $1 \le i \le n$. Ou seja, a janela de tempo depende do particular instante n que está sendo considerado. Por esse motivo, estamos associando o índice n em parênteses aos coeficientes e à seqüência estimada. Dessa forma $h_k(n)$, $k = 0, \ldots, N-1$, é o conjunto de coeficientes que otimiza a estimação pelo critério LS da seqüência de amostras d(i), $i = 1, \ldots, n$, para as n amostras consideradas; assim como y(i,n) representa a estimativa da amostra d(i) usando os coeficientes hk(n). Analogamente ao exemplo anterior, definimos

$$\varepsilon(i,n) = d(i) - y(i,n) \qquad (1.13)$$

A somatória em (1.12) pode ser colocada em uma forma mais compacta, utilizando-se multiplicação vetorial. Para isso, definimos os vetores

$$H^T(n) = [h_0(n)\ \ h_1(n)\ \ \ldots\ \ h_{N-1}(n)] \qquad (1.14)$$

e

$$X^T(n) = [x(n)\ x(n-1)\ \ldots\ x(n-N+1)] \tag{1.15}$$

Assim, de (1.12) e (1.13), podemos escrever

$$\varepsilon(i,n) = d(i) - H^H(n)X(i) \tag{1.16}$$

onde o superíndice ^{H} denota a dupla operação de transposição e conjugação de um vetor ou matriz. Queremos então utilizar o critério LS para minimizar a função custo

$$E_N(n) = \sum_{i=1}^{n} w^{n-i} |\varepsilon(i,n)|^2 \tag{1.17}$$

onde w é o fator de esquecimento, uma constante real escolhida entre 0 e 1, cuja função é ponderar os termos da somatória, de forma a produzir uma maior participação das amostras mais recentes. $E_N(n)$ pode ser reescrito, usando (1.16), como

$$E_N(n) = \sum_{i=1}^{n} w^{n-i} [|d(i)|^2 - d^*(i)H^H(n)X(i) - d(i)X^H(i)H(n) + H^H(n)X(i)X^H(i)H(n)] \tag{1.18}$$

Novamente é possível mostrar que $E_N(n)$ é uma função convexa, de forma que o vetor de coeficientes ótimos pode ser obtido fazendo-se

$$\nabla E_N(n) = 0_N \tag{1.19}$$

onde $\nabla E_N(n)$ representa o gradiente de $E_N(n)$ e 0_N é o vetor nulo de dimensão N. Para a determinação de $\nabla E_N(n)$, e também para facilitar a obtenção de outros resultados futuros, é conveniente estabelecermos algumas propriedades da diferenciação de um escalar em relação a um vetor.

Sejam V e X vetores Nx1 e seja Q uma matriz NxN. Então valem as seguintes propriedades:

Propriedade 1.1.1

$$\frac{d}{dV}[X^H V] = 0_N \tag{1.20a}$$

Propriedade 1.1.2

$$\frac{d}{dV}[V^H X] = 2X \tag{1.20b}$$

Propriedade 1.1.3

$$\frac{d}{dV}[V^H Q V] = 2QV \tag{1.20c}$$

cujas demonstrações estão feitas no Apêndice 1A

Aplicando as Propriedades acima em (1.18), obtemos o gradiente de $E_N(n)$

$$\nabla E_N(n) = -2\sum_{i=1}^{n} w^{n-i}\,[d^*(i)X(i) - X(i)X^H(i)H(n)] \tag{1.21}$$

Usando (1.19), temos finalmente

$$H(n) = R^{-1}(n)\, r_{dx}(n) \tag{1.22}$$

onde

$$R(n) = \sum_{i=1}^{n} w^{n-i}\, X(i)X^H(i) \tag{1.23}$$

é a matriz de autocorrelação determinística do sinal x(n), e

$$r_{dx}(n) = \sum_{i=1}^{n} w^{n-i}\, d^*(i)X(i) \tag{1.24}$$

é o vetor de correlação cruzada entre {d(n)} e {x(n)}.

À primeira vista, a solução dada por (1.22) parece exigir um esforço computacional muito grande. Contudo, as situações práticas que motivam a proposição do Exemplo 1.2 são tais que requerem o cálculo do vetor de coeficientes a cada período de amostragem. Com isso, no cálculo de H(n) podem ser utilizados os resultados intermediários envolvidos no cálculo de H(n-1), o que permite reduzir consideravelmente o esforço computacional. Usando essa filosofia, podemos obter R(n) e $r_{dx}(n)$ recursivamente através das relações

$$R(n) = w\,R(n-1) + X(n)X^H(n) \tag{1.25}$$

e

$$r_{dx}(n) = w\, r_{dx}(n-1) + d^*(n)X(n) \tag{1.26}$$

As equações (1.22), (1.25) e (1.26) estão reproduzidas na Tabela 1.1 na forma de um algoritmo, denominado Algoritmo LS. O número de operações envolvidas em cada iteração deste algoritmo é ainda muito grande devido à necessidade de inversão da matriz R(n) [algo proporcional a N^3], o que o torna inadequado para aplicações que exigem processamento em tempo

Tabela 1.1 - Algoritmo LS - N^3

disponível, do instante n-1

$r_{dx}(n-1)$, $R(n-1)$

recebido no instante n

$x(n)$, $d(n)$

atualizações

$$R(n) = w\,R(n-1) + X(n)X^H(n)$$

$$r_{dx}(n) = w\,r_{dx}(n-1) + d^*(n)\,X(n)$$

$$P(n) = R^{-1}(n)$$

$$H(n) = P(n)r_{dx}(n)$$

real. Veremos adiante que essa complexidade pode ser reduzida substancialmente com a introdução do Algoritmo RLS (Recursive Least Square), que evita a inversão de matrizes e diminui a quantidade de operações para um número proporcional a N^2. No Capítulo 2 apresentaremos os algoritmos RLS rápidos, que reduzem ainda mais o esforço computacional exigido na solução deste problema, abaixando-o para um número proporcional a N.

A matriz de autocorrelação, R(n), é um elemento de grande importância em todo decorrer deste trabalho. Por esse motivo vamos estudá-la com mais detalhes.

1.2 – A MATRIZ DE AUTOCORRELAÇÃO

Por razões que ficarão claras na próxima seção, vamos denominar {x(n)} como *sinal de entrada* ou *seqüência de entrada*. É de se esperar que a matriz de autocorrelação determinística deste sinal, R(n), definida em (1.23), traga informações acerca de suas características estatísticas, e vice-versa. De fato, este relacionamento se verifica, sobretudo quando x(n) assume um comportamento estacionário. Consideremos então algumas das propriedades da matriz de autocorrelação determinística.

Propriedade 1.2.1

Se {x(n)} é um processo estacionário discreto e $r_{ij}(n)$ o j-ésimo elemento da i-ésima linha de sua matriz de autocorrelação determinística, então para $n \rightarrow \infty$, temos

$$E[r_{ij}(n)] = \frac{1}{1-w}\, r(j-i) \qquad (1.27)$$

onde

$$r(k) = E[x(n)x^*(n-k)] \tag{1.28}$$

é a função de autocorrelação da seqüência x(n) e E[.] denota a operação estatística da média, ou esperança matemática.

Essa propriedade segue diretamente da aplicação da média na definição (1.23) para o limite com $n \to \infty$. O fator 1/(1–w) surge como resultado da soma da série geométrica infinita com razão w.

O fator 1/(1–w) tem também uma outra interpretação, em termos da inércia do processo de esquecimento. A função custo, dada por (1.17), tem suas parcelas ponderadas pela seqüência exponencial decrescente w^n, $n \geq 0$. Assim, a grosso modo, podemos considerar como significativas na otimização as parcelas que apresentam uma ponderação w^n maior que e^{-1}. Ou seja, as parcelas que pertencem à janela correspondente a uma constante de tempo de decaimento. Essa janela tem um comprimento n_o tal que

$$w^{n_o} = \frac{1}{e} \tag{1.29}$$

Ou, o que queríamos mostrar

$$n_o \approx \frac{1}{1-w} \tag{1.30}$$

onde usamos a aproximação $\ln(w) \approx w-1$, justificada pelo fato de que, em aplicações práticas, o fator de esquecimento, w, tem sempre um valor próximo de 1

Propriedade 1.2.2

A matriz de autocorrelação R(n) é Hermitiana. Ou seja,

$$R^H(n) = R(n) \tag{1.31}$$

Esta propriedade decorre trivialmente da definição (1.23)

Propriedade 1.2.3

A matriz de autocorrelação R(n) é não negativa definida. Ou seja,

$$V^H R(n) V \geq 0 \tag{1.32}$$

para V, um vetor Nx1 qualquer.

Usando (1.23) podemos reescrever a forma Hermitiana como

$$V^H R(n) V = \sum_{i=1}^{n} w^{n-i} V^H X(i) X^H(i) V \tag{1.33}$$

$$= \sum_{i=1}^{n} w^{n-1} [V^H X(i)] [V^H X(i)]^* \tag{1.34}$$

$$= \sum_{i=1}^{n} w^{n-1} |V^H X(i)|^2 \tag{1.35}$$

de onde resulta a inequação (1.32)

Propriedade 1.2.4

Se a forma Hermitiana é nula para algum vetor não nulo V, isto é,

$$V^H R(n) V = 0 \tag{1.36}$$

então a matriz de autocorrelação R(n) é singular.

Dizemos que uma matriz é singular quando suas linhas (ou colunas) são linearmente dependentes. Ou equivalentemente, quando existe uma combinação linear de suas linhas (ou colunas) que produza o vetor nulo. A condição (1.36) somente é atendida quando todos os produtos internos em (1.35) são iguais a zero, ou seja,

$$V^H X(i) = 0 \qquad i = 1, 2, \ldots, n \tag{1.37}$$

Notando, pela definição (1.23), que R(n) tem suas colunas formadas por combinações lineares dos vetores X(i), temos que (1.37) implica em

$$V^H R(n) = 0_N^T \tag{1.38}$$

Como V é não nulo, temos que R(n) tem suas linhas linearmente dependentes, e portanto R(n) é singular se valer a equação (1.36).

Quando uma matriz é singular, sua inversa não existe. Este fato poderia causar algum transtorno à solução do problema descrito no Exemplo 1.2. Entretanto, de (1.37), vemos que somente para casos muito particulares da seqüência {x(n)} teríamos uma matriz R(n) singular. Esses casos ocorrem essencialmente quando x(n) é um sinal constituído por uma soma de senóides, onde o número de senóides é inferior a N. Sendo essa uma situação muito peculiar, podemos afirmar que a matriz de autocorrelação R(n) é quase sempre não singular e, pelas

Propriedades 1.2.3 e 1.2.4, que é quase sempre positiva definida [4]. Uma Matriz R(n), positiva definida, tem sempre uma inversa que, pode-se mostrar, é também Hermitiana e positiva definida.

Propriedade 1.2.5

Os autovalores da matriz de autocorrelação R(n) são todos reais e não negativos.

Pela definição de autovalores, se λ_i é um autovalor de R(n), então

$$\lambda_i U_i = R(n)U_i \qquad i = 1, 2, \ldots, N \qquad (1.39)$$

onde U_i é um autovetor associado a λ_i. Pré-multiplicando ambos os membros de (1.39) por U_i^H, obtemos:

$$\lambda_i = \frac{U_i^H R(n) U_i}{U_i^H U_i} \qquad i = 1, 2, \ldots, N \qquad (1.40)$$

O denominador do segundo membro de (1.40) é a norma Euclidiana ao quadrado do autovetor U_i, e portanto, é um escalar real e positivo. De (1.32) temos que o nominador é também real e não negativo. Daí conclui-se que os autovalores de R(n) são todos reais e não negativos.

Propriedade 1.2.6

Se $U_1, U_2, \ldots, U_N$ são autovetores associados respectivamente aos autovalores distintos de R(n): $\lambda_1, \lambda_2, \ldots, \lambda_N$, então $U_1, U_2, \ldots, U_N$ são todos ortogonais entre si. Ou seja

$$U_i^H U_j = 0 \qquad i \neq j \qquad (1.41)$$

Da definição de autovalores, temos

$$R(n)U_i = \lambda_i U_i \qquad (1.42)$$

e

$$R(n)U_j = \lambda_j U_j \qquad (1.43)$$

Levando em conta que $R(n) = R^H(n)$ e λ_j é real (Propriedades 1.2.2 e 1.2.5), podemos reescrever (1.43) como

$$U_j^H R(n) = \lambda_j U_j^H \qquad (1.44)$$

Pré-multiplicando ambos os membros de (1.42) por U_j^H e pós-multiplicando ambos os membros de (1.44) por U_i, vem

$$\lambda_i U_j^H U_i = \lambda_j U_j^H U_i \qquad (1.45)$$

ou

$$[\lambda_i - \lambda_j]\, U_j^H U_i = 0 \tag{1.46}$$

Como estamos assumindo $\lambda_i \neq \lambda_j$, temos forçosamente como válida a equação (1.41).

Propriedade 1.2.7

Se $\lambda_1, \lambda_2, \ldots, \lambda_N$ são os autovalores da matriz de autocorrelação R(n), então $1/\lambda_1$, $1/\lambda_2, \ldots, 1/\lambda_N$ são os autovalores de $R^{-1}(n)$.

Esta propriedade é comprovada pré-multiplicando ambos os membros da definição (1.39) por $R^{-1}(n)$.

Podemos agora definir o fator de condicionamento da matriz de autocorrelação determinística, ou fator de espalhamento dos autovalores de R(n)

$$\chi[R(n)] = \frac{\lambda_{max}}{\lambda_{min}} \tag{1.47}$$

onde λ_{max} e λ_{min} são respectivamente o maior e o menor autovalor de R(n). Este fator pode ser melhor entendido se utilizarmos o conceito de transformação linear. Denotando por C^N o espaço vetorial complexo de N dimensões, podemos afirmar que R(n) caracteriza uma transformação linear de C^N em C^N. É interessante analisar esta transformação através dos autovetores de R(n). A Propriedade 1.2.6 sugere que é possível obter uma base ortonormal de autovetores distintos para C^N, pois os autovetores associados a autovalores distintos são ortogonais. De fato, se R(n) é não singular, é possível obter uma base ortonormal de autovetores para C^N ainda que os autovalores de R(n) não sejam distintos. Da definição (1.39), vemos que os autovetores de R(n) são elementos de C^N para os quais a aplicação da transformação linear R(n) resultam em um vetor de mesma "direção", porém com o "comprimento" alterado de acordo com o respectivo autovalor (um escalar real e não negativo). Com isso, o fator de condicionamento $\chi[R(n)]$ representa a distorção que a transformação linear R(n) provoca em C^N. De (1.47) e da Propriedade 1.2.5, vemos que

$$\chi[R(n)] \geq 1 \tag{1.48}$$

e, da Propriedade 1.2.7, que

$$\chi[R^{-1}(n)] = \chi[R(n)] \tag{1.49}$$

Podemos intuir, da discussão acima, que matrizes com $\chi[R(n)]$ muito altos possuem também uma dispersão muito grande nos valores de seus elementos. Esse fato pode trazer problemas computacionais na solução de sistemas de equações envolvendo R(n) [ou $R^{-1}(n)$]. Por este motivo, dizemos que uma matriz com $\chi[R(n)]$ grande é uma matriz "mal condicionada".

Como deve ser esperado, o fator de condicionamento traz também alguma informação sobre o sinal de entrada. Em [4] é mostrado que

$$\chi[R(n)] \leq \frac{S_{max}}{S_{min}} \tag{1.50}$$

onde S_{max} e S_{min} são respectivamente o máximo e o mínimo valor absoluto da densidade espectral de potência do processo estocástico $\{x(n)\}$.

1.3 – FILTRO TRANSVERSAL ADAPTATIVO

A idéia de filtragem adaptativa emerge de um contexto onde os parâmetros do filtro são obtidos através de operações matemáticas sobre o sinal que está sendo processado. Presume-se que o sinal tenha características variantes no tempo e o filtro deva sofrer constantes correções em seus parâmetros, de forma a se adaptar às novas condições do sinal. Dentre os esquemas propostos para filtragem adaptativa, o que utiliza o filtro transversal é certamente o mais simples. É também o único que iremos considerar neste trabalho. Este filtro é constituído de uma linha de atrasadores com N derivações, as quais são aplicadas cada uma a um multiplicador h_k, $k = 0, \ldots, N-1$, e a seguir somadas, como mostra a Figura 1.1. Nesta figura os multiplicadores têm valores fixos, e o filtro transversal é dito "não adaptativo".

Um filtro não adaptativo pode ser plenamente caracterizado por sua resposta impulsiva: um vetor H, de dimensão N, cujos elementos são os multiplicadores h_k, ou por sua resposta em freqüência

$$H(e^{j\omega}) = \sum_{k=0}^{N-1} h_k e^{-j\omega k} \tag{1.51}$$

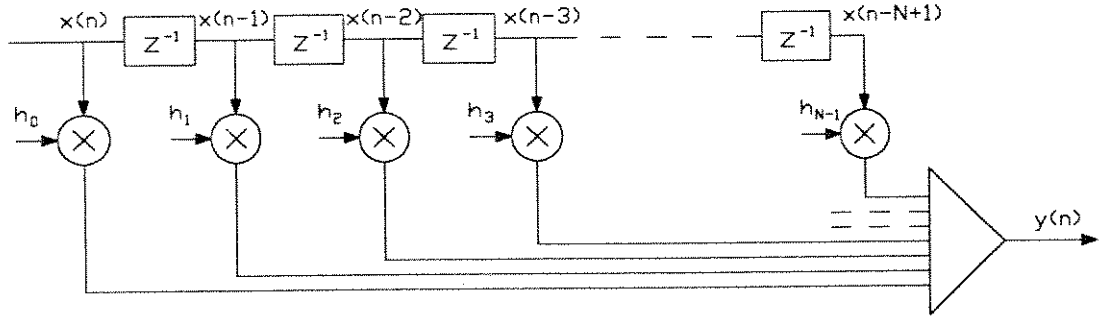


Figura 1.1 - Estrutura do Filtro Transversal

A resposta deste filtro ao sinal de entrada x(n) é dada pela convolução de x(n) com a resposta impulsiva do filtro

$$y(n) = H^T X(n) \tag{1.52}$$

Um outro aspecto prático do filtro transversal é que, para uma ordem N suficientemente grande, qualquer resposta impulsiva finita pode ser implementada. Além disso, como a Transformada Z dessa resposta não apresenta pólos senão na origem, temos que todos os pólos se encontram no interior do círculo de raio unitário no plano Z, o que garante a estabilidade da resposta do filtro.

Em aplicações onde o filtro transversal é mantido fixo, os multiplicadores podem ser calculados a partir da resposta em freqüência desejada; usando-se a Transformada Inversa de Fourier

$$h_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} F(e^{j\omega})\, e^{j\omega k}\, d\omega \qquad k = 0, \ldots, N-1 \tag{1.53}$$

onde $F(e^{j\omega})$ é a resposta em freqüência desejada e o período de amostragem foi normalizado para T = 1. Este procedimento, que utiliza um tratamento no domínio da freqüência, tem alguma similaridade com aquele descrito no Exemplo 1.1. Em [5] podem ser encontradas diversas técnicas de projeto de filtros digitais com coeficientes fixos. Não queremos nos estender sobre elas, uma vez que nosso objetivo é o processamento de sinais utilizando a filtragem adaptativa.

O tratamento para o filtro adaptativo apresenta algumas dificuldades pois, a rigor, não se pode falar em resposta em freqüência, já que os elementos do filtro não se mantém fixos. Neste sentido, é conveniente que os valores dos multiplicadores provenham de um processo de otimização no domínio do tempo, e não no domínio da freqüência. A minimização descrita no Exemplo 1.2 se encaixa perfeitamente neste caso. O vetor H(n) representa então, o filtro que, no instante n, minimiza a função custo definida em (1.17). A cada instante de amostragem os coeficientes do filtro são atualizados, de forma a considerar os novos dados recebidos. Se por acaso, as estatísticas dos sinais forem mantidas estacionárias, devemos esperar que os coeficientes convirjam para valores determinados, e que não sofram grandes variações em torno destes valores. Por outro lado, se os sinais forem não estacionários, os coeficientes ficarão se reajustando continuamente, dependendo de dois fatores: da rapidez com que se dá este processo, e do fator de esquecimento w. Contudo, é bom notar que, estando o filtro em fase de adaptação ou não, os coeficientes serão sempre ótimos, no sentido de que atendem, em cada instante, o critério LS.

No n-ésimo instante de tempo, quando uma nova amostra x(n) chega ao filtro, este é encontrado com os coeficientes obtidos no instante n-1. Dessa forma, duas operações de filtragem são possíveis neste intervalo de tempo: a primeira, com o filtro desatualizado, H(n-1); e a segunda, com os multiplicadores já atualizados, H(n). Estas duas situações estão ilustradas na Figura 1.2, assim como os sinais gerados a partir delas. Muito embora tais sinais possam ser referidos por $\varepsilon(n,n-1)$ e $\varepsilon(n,n)$ conforme estabelecido em (1.16), preferimos adotar uma notação mais simplificada para a distinção destes sinais, definindo

$$e(n) = d(n) - H^H(n-1)X(n) \tag{1.54}$$

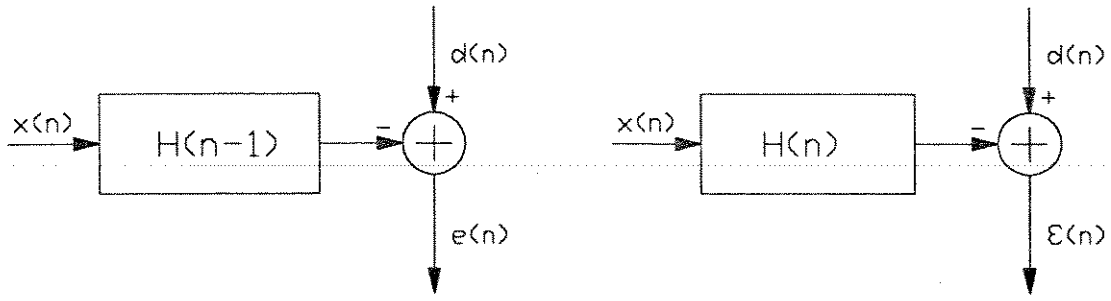


Figura 1.2 - Duas situações do Filtro Transversal Adaptativo

como sendo o erro a priori, e

$$\varepsilon(n) = d(n) - H^H(n)X(n) \tag{1.55}$$

como sendo o erro residual, ou erro a posteriori.

É natural que o erro a posteriori, $\varepsilon(n)$, caracteriza melhor o processo de aproximação que estamos considerando, uma vez que ele é resultante da operação com o filtro já atualizado. Entretanto, o erro a priori, e(n), tem também sua utilidade. Veremos a seguir que ele é utilizado na obtenção de um algoritmo de mínimos quadrados recursivo para a adaptação do filtro transversal.

1.4 – ALGORITMO RLS

O algoritmo que iremos descrever nesta seção é um refinamento do Algoritmo LS apresentado na Tabela 1.1. Enquanto naquele, a matriz R(n) e o vetor $r_{dx}(n)$ são primeiramente atualizados para o posterior cálculo de H(n), neste o vetor H(n) é atualizado diretamente a partir de H(n-1). Veremos que este procedimento evita também a indesejável inversão da matriz de autocorrelação, o que reduz significativamente a complexidade, ou o número de operações exigidas.

Tomando inicialmente as equações (1.22) e (1.26) podemos escrever

$$H(n) = R^{-1}(n)\,[w\, r_{dx}(n-1) + d^*(n)X(n)] \tag{1.56}$$

De (1.22) e (1.25), temos

$$r_{dx}(n-1) = R(n-1)H(n-1) \tag{1.57}$$

$$= \frac{1}{w}\,[R(n) - X(n)X^H(n)]\,H(n-1) \tag{1.58}$$

Substituindo (1.58) em (1.56) vem

$$H(n) = H(n-1) + R^{-1}(n)X(n)\,[d^*(n) - X^H(n)H(n-1)] \tag{1.59}$$

$$= H(n-1) + R^{-1}(n)X(n)e^*(n) \tag{1.60}$$

$$= H(n-1) + e^*(n)\,G(n) \tag{1.61}$$

Onde, na última passagem, definimos o vetor de ganho de adaptação

$$G(n) = R^{-1}(n)X(n) \tag{1.62}$$

também conhecido por vetor de Kalman.

Precisamos agora encontrar uma forma de atualizar a matriz inversa, $R^{-1}(n)$, a partir de $R^{-1}(n-1)$. Usaremos para isso uma identidade algébrica conhecida como Lema de Inversão de Matrizes

Lema 1.1

Sejam A, B, C e D matrizes de dimensões compatíveis tais que

$$A = B + CDC^H \tag{1.63}$$

onde as matrizes quadradas A, B e D são positivas definidas. Então

$$A^{-1} = B^{-1} - B^{-1}C\,[D^{-1} + C^HB^{-1}C]^{-1}\,C^HB^{-1} \tag{1.64}$$

Uma demonstração do Lema 1.1 é dada no Apêndice 1B. Podemos aplicar este lema à equação (1.25) fazendo

$$A = R(n) \tag{1.65a}$$

$$B = w\,R(n-1) \tag{1.65b}$$

$$C = X(n) \tag{1.65c}$$

$$D = 1 \tag{1.65d}$$

de onde obtemos

$$R^{-1}(n) = \frac{1}{w}\left[R^{-1}(n-1) - \frac{1}{w + X^H(n)R^{-1}(n-1)X(n)}\,R^{-1}(n-1)X(n)X^H(n)R^{-1}(n-1)\right] \tag{1.66}$$

É notável a redução computacional fornecida por (1.66). A inversão da matriz R(n) é artificiosamente substituída por somas e multiplicações matriciais e, à rigor, uma única divisão por escalar. Essa redução fica realçada se considerarmos que alguns produtos são redundantes no membro direito de (1.66) e podem ser evitados usando-se variáveis auxiliares.

Substituindo (1.66) em (1.62) e fazendo as devidas simplificações, temos

$$G(n) = \frac{1}{w + X^H(n)P(n-1)X(n)} P(n-1)X(n) \tag{1.67}$$

onde, por comodidade, definimos

$$P(n) = R^{-1}(n) \tag{1.68}$$

Dessa forma, podemos reescrever (1.66) como

$$P(n) = \frac{1}{w} [P(n-1) - G(n)X^H(n)P(n-1)] \tag{1.69}$$

Na Tabela 1.2 sintetizamos o Algoritmo RLS (Recursive Least Square) a partir de (1.54), (1.61), (1.67) e (1.69). A demanda computacional por iteração é de $\frac{3}{2} N^2 + \frac{5}{2} N$ multiplicações, $2N^2 + 5N$ adições e 1 divisão. Estes números foram obtidos aproveitando a característica Hermitiana de P(n) [somente são computados os $(N^2+N)/2$ elementos que figuram na diagonal principal e acima dela], usando um vetor auxiliar para produto P(n-1)X(n) e ignorando o caráter genericamente complexo das variáveis.

A inicialização do Algoritmo RLS exige a existência de P(0). Este inconveniente pode ser evitado admitindo-se uma modificação na definição de R(n)

$$R(n) = \sum_{i=1}^{n} w^{n-i} X(i)X^H(i) + w^n \delta I_N \tag{1.70}$$

onde δ é um escalar bem pequeno e I_N é a matriz identidade NxN. Dessa forma

$$P(0) = \frac{1}{\delta} I_N \tag{1.71}$$

A modificação imposta por (1.70) não altera a estrutura do algoritmo. Graças ao baixo valor de δ e ao decaimento exponencial gerado por w, sua influência na estimação dos coeficientes é praticamente desprezível, sobretudo após decorrido um período de tempo inicial [4].

Também para H(n) é necessário o estabelecimento de um valor de partida. Uma escolha natural quando não se tem uma estimativa a priori é

$$H(0) = 0_N \tag{1.72}$$

Tabela 1.2 - Algoritmo RLS

disponível, do instante n-1

$X(n-1), H(n-1), G(n-1), P(n-1)$

recebido no instante n

$x(n), d(n)$

atualizações

$$G(n) = \frac{1}{w + X^H(n)P(n-1)X(n)} P(n-1)X(n)$$

$$P(n) = \frac{1}{w}[P(n-1) - G(n)X^H(n)P(n-1)]$$

$$e(n) = d(n) - H^H(n-1)X(n)$$

$$H(n) = H(n-1) + e^*(n)\,G(n)$$

Cabe agora algumas observações sobre o algoritmo descrito acima. Ele fornece exatamente o mesmo resultado (exceto pelas modificações exigidas na inicialização) do Algoritmo LS básico, visto na Seção 1.1, e apresenta uma complexidade moderada para valores baixos de N. O artifício utilizado para a redução da complexidade reside no emprego do vetor de ganho, G(n), que atualiza diretamente a matriz inversa, P(n), assim como o vetor de coeficientes, H(n).

Vale destacar que o Algoritmo RLS não faz absolutamente uso da característica de deslocamento do vetor de entrada X(n). Ou seja, não aproveita o fato de que os N-1 últimos elementos de X(n) são meros deslocamentos dos N-1 primeiros elementos de X(n-1). Dessa forma, podemos vislumbrar uma redução ainda maior na complexidade do método LS. De fato, a propriedade de deslocamento de X(n), estabelecida na definição (1.15), é uma imposição matemática desnecessária para o problema de minimização descrito no Exemplo 1.2. Ela está presente no caso específico da filtragem transversal devido à natureza desta estrutura, mas é dispensável sob o ponto de vista funcional do Algoritmo RLS. O aproveitamento da propriedade de deslocamento de X(n) para a redução da complexidade computacional será discutido no Capítulo 2 com a introdução dos algoritmos rápidos.

1.5 – ALGORITMO LMS

Nesta seção iremos descrever brevemente o Algoritmo LMS (Least Mean Square). Ainda que não tenha uma relação direta com o objeto principal de nosso estudo, achamos conveniente discuti-lo, devido a sua grande importância no contexto da filtragem adaptativa, sobretudo quando associada ao filtro transversal.

Muito embora o Algoritmo LMS seja usualmente derivado a partir de uma abordagem estocástica, procuraremos fazê-lo de uma outra forma. No intuito de abreviar o tratamento e propor um desenvolvimento alternativo, vamos apresentá-lo como uma simplificação do Algoritmo RLS. Tomamos para isso a equação (1.60)

$$H(n) = H(n-1) + e^{*}(n)\, R^{-1}(n) X(n) \tag{1.73}$$

Se admitirmos como válida a aproximação

$$R(n) = \frac{1}{\mu} I_N \tag{1.74}$$

onde μ é uma constante real e positiva, temos

$$H(n) = H(n-1) + \mu\, e^{*}(n)\, X(n) \tag{1.75}$$

Dessa forma, o Algoritmo LMS é representado simplesmente pelas equações (1.54) e (1.75), conforme disposto na Tabela 1.3.

Para o caso de um sinal estacionário, com amostras descorrelacionadas, a aproximação (1.74) é razoável, o que garante o funcionamento do algoritmo. Mostraremos, entretanto, que, mesmo para o caso de sinais estacionários em geral, onde essa aproximação é grosseira, uma escolha apropriada do parâmetro μ faz com que os coeficientes do filtro convirjam.

Ainda que o critério de otimalidade usado no Algoritmo LMS seja diferente do usado no método LS; para o caso específico de sinais estacionários, os elementos de H(n) convergem praticamente para os mesmos valores em ambos os algoritmos. Não querendo prolongar excessivamente essa discussão, admitiremos simplesmente que seja H_∞ o vetor para o qual o filtro H(n) deve convergir em determinada situação de estacionaridade. Assim, podemos definir

$$\varepsilon(n,\infty) = d(n) - H_\infty^{H} X(n) \tag{1.76}$$

como sendo o erro produzido no instante n pelo filtro transversal na condição ótima. Definimos também o vetor

$$C(n) = H(n) - H_\infty \tag{1.77}$$

e o escalar

$$J(n) = C^{H}(n) C(n) \tag{1.78}$$

O valor de J(n) representa um bom parâmetro para se avaliar a convergência do filtro. Ele é o módulo ao quadrado da diferença entre o vetor atual e o vetor desejado. Subtraindo H_∞ nos dois membros de (1.75), temos

$$C(n) = C(n-1) + \mu\, e^{*}(n)\, X(n) \tag{1.79}$$

Tabela 1.3 - Algoritmo LMS

disponível, do instante n-1

$X(n-1), H(n-1)$

recebido no instante n

$x(n), d(n)$

atualizações

$$e(n) = d(n) - H^H(n-1)X(n)$$
$$H(n) = H(n-1) + \mu\, e^*(n)\, X(n)$$

Usando (1.79) em (1.78), vem

$$J(n) = C^H(n-1)C(n-1) + 2\,\mu\, Re[e^*(n)\, C^H(n-1)X(n)] + \mu^2\, |e(n)|^2\, X^H(n)X(n) \quad (1.80)$$

O produto vetorial no colchetes do segundo termo pode ser reescrito como

$$C^H(n-1)X(n) = H^H(n-1)X(n) - H_\infty^H\, X(n) \quad (1.81)$$

$$= d(n) - e(n) - H_\infty^H\, X(n) \quad (1.82)$$

$$= \varepsilon(n,\infty) - e(n) \quad (1.83)$$

onde usamos (1.77), (1.54) e (1.76). Substituindo este resultado acima, temos

$$J(n) = J(n-1) - 2\,\mu\, Re\{e^*(n)\, [\varepsilon(n,\infty) - e(n)]\} + \mu^2\, |e(n)|^2\, X^H(n)X(n) \quad (1.84)$$

Vamos agora considerar o filtro no estado inicial, quando os coeficientes encontram-se totalmente desajustados. Nessa situação, devemos esperar que e(n) assuma valores grandes quando comparados a $\varepsilon(n,\infty)$. Podemos então desprezar $\varepsilon(n,\infty)$ na expressão (1.84), resultando

$$J(n) = J(n-1) + 2\,\mu |e(n)|^2 + \mu^2\, |e(n)|^2\, X^H(n)X(n) \quad (1.85)$$

A convergência do filtro pode ser garantida impondo que

$$J(n) \leq J(n-1) \quad (1.86)$$

Esta imposição implica em

$$0 \leq \mu \leq \frac{2}{X^H(n)X(n)} \quad (1.87)$$

Lembrando que

$$E[X^H(n)X(n)] = N\,r(0) \tag{1.88}$$

onde r(0) é a potência média das amostras x(n), podemos delimitar a faixa de valores de μ para a qual o Algoritmo LMS converge [no sentido de minimizar J(n)] como sendo

$$0 \leq \mu \leq \frac{2}{N\,r(0)} \tag{1.89}$$

O resultado acima, derivado sob uma ótica predominantemente determinística, é em geral obtido na literatura através de uma abordagem estocástica [2], [4], [6].

O comportamento do Algoritmo LMS nos períodos de não estacionaridade dos sinais está de longe fora do critério LS. Nos muitos estudos comparativos entre os Algoritmos LMS e RLS, as conclusões mais significativas ressaltam que o tempo de convergência do primeiro é uma ordem de grandeza maior que o tempo de convergência do segundo; além disso, no Algoritmo LMS este tempo depende das características do sinal de entrada, x(n) [4]. Essa dependência é facilmente explicável, pois a correlação entre as amostras de $\{x(n)\}$ implicam em uma matriz de autocorrelação com um fator de condicionamento $\chi[R(n)] \gg 1$, o que diminui a validade da aproximação (1.74). O Algoritmo RLS, por outro lado, é totalmente indiferente às propriedades estatísticas do sinal de entrada, devido a própria forma como os dados são tratados, que mantém o algoritmo inerentemente dentro do critério LS.

Apesar de pouco eficiente quando utilizado com sinais não estacionários e/ou muito correlacionados, o Algoritmo LMS está consagrado devido à sua baixa complexidade computacional: apenas 2N+1 multiplicações e 2N adições por iteração. Graças a esta característica, são encontrados na prática, muitos sistemas que o empregam. Outro fator favorável ao uso deste algoritmo é sua reconhecida estabilidade numérica.

Para um tratamento mais apropriado e bastante detalhado do Algoritmo LMS gostaríamos de citar a referência [4].

1.6 – COMENTÁRIO

Neste capítulo expusemos a teoria de mínimos quadrados e vimos como o Algoritmo RLS, de complexidade proporcional a N^2, pode ser usado para atualizar os coeficientes de um filtro transversal adaptativo. Procuramos restringir o tratamento ao âmbito dos elementos necessários para o entendimento do critério LS, assim como sua aplicação à estrutura do filtro transversal. No próximo capítulo veremos que a tarefa de atualizar o filtro transversal pode também ser executada pelos algoritmos rápidos, que representam uma solução bastante atrativa, uma vez que oferecem a mesma resposta que o Algoritmo RLS, porém, com uma complexidade proporcional a N.

O Algoritmo LMS foi incluído devido ao seu consagrado uso nos sistemas adaptativos. Seu mecanismo de convergência foi demonstrado de uma forma inédita, usando-se um tratamento determinístico. O resultado confere com o obtido através do tratamento estocástico, usualmente encontrado na literatura.

Para uma abordagem mais aprofundada das questões tratadas neste capítulo, incluindo as estruturas LS do tipo lattice, sugerimos a referência [7].

APÊNDICE 1A

Neste apêndice vamos demonstrar as Propriedades 1.1.1, 1.1.2 e 1.1.3, que estabelecem algumas regras para a diferenciação de funções escalares em relação a vetores.

Seja f uma função escalar de um vetor V. Então, a derivada de f em relação a V é definida por

$$\frac{d}{dV} f = \begin{bmatrix} \frac{\partial f}{\partial a_1} + j \frac{\partial f}{\partial b_1} \\ \frac{\partial f}{\partial a_2} + j \frac{\partial f}{\partial b_2} \\ \vdots \\ \frac{\partial f}{\partial a_N} + j \frac{\partial f}{\partial b_N} \end{bmatrix} \tag{1.90}$$

onde a_i e b_i representam as partes real e imaginária do i-ésimo elemento do vetor V, ou seja

$$v_i = a_i + j\, b_i \qquad i = 1, 2, \ldots, N \tag{1.91}$$

Fazendo

$$f = X^H V \tag{1.92}$$

podemos escrever

$$f = \sum_{i=1}^{N} x_i^* v_i \tag{1.93}$$

$$= \sum_{i=1}^{N} x_i^* [a_i + j\, b_i] \tag{1.94}$$

Assim

$$\frac{\partial f}{\partial a_i} = x_i^* \qquad i = 1, 2, \ldots, N \tag{1.95a}$$

e

$$\frac{\partial f}{\partial b_i} = j\, x_i^* \qquad i = 1, 2, \ldots, N \qquad (1.95b)$$

Substituindo (1.95a) e (1.95b) em (1.90), temos

$$\frac{d}{dV}[X^H V] = 0 \qquad (1.96)$$

que corresponde à equação (1.20a), ou Propriedade 1.1.1.

Fazendo

$$f = V^H X \qquad (1.97)$$

temos

$$f = \sum_{i=1}^{N} v_i^* x_i \qquad (1.98)$$

$$= \sum_{i=1}^{N} [a_i - j\, b_i]\, x_i \qquad (1.99)$$

Assim

$$\frac{\partial f}{\partial a_i} = x_i \qquad i = 1, 2, \ldots, N \qquad (1.100a)$$

e

$$\frac{\partial f}{\partial b_i} = -j\, x_i \qquad i = 1, 2, \ldots, N \qquad (1.100b)$$

Substituindo (1.100a) e (1.100b) em (1.90), temos

$$\frac{d}{dV}[V^H X] = 2X \qquad (1.101)$$

que corresponde à Propriedade 1.1.2

Fazendo

$$f = V^H Q V \qquad (1.102)$$

podemos escrever

$$f = \sum_{i=1}^{N} \sum_{k=1}^{N} v_i^* v_k q_{ik} \tag{1.103}$$

onde q_{ik} é o k-ésimo elemento da i-ésima linha da matriz Q. Usando (1.91), temos

$$f = \sum_{i=1}^{N} \sum_{k=1}^{N} [a_i - jb_i][a_k + jb_k] q_{ik} \tag{1.104}$$

Efetuando as derivações parciais, vem

$$\frac{\partial f}{\partial a_i} = \sum_{k=1}^{N} v_k q_{ik} + v_k^* q_{ki} \qquad i = 1, 2, \ldots, N \tag{1.105a}$$

$$\frac{\partial f}{\partial b_i} = \sum_{k=1}^{N} -j v_k q_{ik} + j v_k^* q_{ki} \qquad i = 1, 2, \ldots, N \tag{1.105b}$$

De (1.105a) e (1.105b), podemos escrever

$$\frac{\partial f}{\partial a_i} + j \frac{\partial f}{\partial b_i} = 2 \sum_{k=1}^{N} v_k q_{ik} \qquad i = 1, 2, \ldots, N \tag{1.106}$$

Substituindo este resultado em (1.90), e notando que a somatória acima corresponde ao produto escalar da i-ésima linha da matriz Q pelo vetor V, temos finalmente

$$\frac{d}{dV} [V^H Q V] = 2 QV \tag{1.107}$$

que representa a Propriedade 1.1.3

APÊNDICE 1B

Neste apêndice, queremos demonstrar a validade do Lema 1.1. Para isso tomemos a igualdade matricial

$$I_N - [C^H B^{-1} C + D^{-1}] [C^H B^{-1} C + D^{-1}]^{-1} = 0 \tag{1.108}$$

onde B e D são matrizes quadradas positivas definidas e C é uma matriz de dimensões compatíveis com as operações. Desmembrando o primeiro colchetes, temos

$$I_N - C^H B^{-1} C \, [C^H B^{-1} C + D^{-1}]^{-1} - D^{-1} \, [C^H B^{-1} C + D^{-1}]^{-1} = 0_N \qquad (1.109)$$

Pré-multiplicando ambos os membros por CD e pós-multiplicando por $C^H B^{-1}$, vem

$$CDC^H B^{-1} - CDC^H B^{-1} C \, [C^H B^{-1} C + D^{-1}]^{-1} \, C^H B^{-1} - C \, [C^H B^{-1} C + D^{-1}]^{-1} \, C^H B^{-1} = 0_N \qquad (1.110)$$

Somando BB^{-1} no primeiro membro e I_N no segundo membro de (1.110), podemos escrever

$$I_N = BB^{-1} - C \, [C^H B^{-1} C + D^{-1}]^{-1} \, C^H B^{-1} + CDC^H B^{-1} - CDC^H B^{-1} C \, [C^H B^{-1} C + D^{-1}]^{-1} \, C^H B^{-1} \qquad (1.111)$$

Reagrupando os termos, vem

$$I_N = [B + CDC^H] \, [B^{-1} - B^{-1} C \, (C^H B^{-1} C + D^{-1})^{-1} \, C^H B^{-1}] \qquad (1.112)$$

De onde obtemos finalmente

$$[B + CDC^H]^{-1} = B^{-1} - B^{-1} C \, (C^H B^{-1} C + D^{-1})^{-1} \, C^H B^{-1} \qquad (1.113)$$

que corresponde à equação (1.64), ou, ao Lema de Inversão de Matrizes.

1.7 – REFERÊNCIAS

[1] Gimenez, J. R. B., "Estudos de processos adaptativos aplicados aos problemas de eliminação de interferência intesimbólica/cancelamento de ecos". Tese de mestrado, Unicamp, março de 1988.

[2] Bellanger, M. G., Adaptive Digital Filters and Signal Analysis, Marcel Dekker Inc., New York, 1987.

[3] Proakis, J. G., Digital Communications, Mc Graw-Hill, 1983.

[4] Haykin, S., Adaptive Filter Theory, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[5] Oppenheim, A. V., Schafer, R. W., Digital Signal Processing, Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[6] Cowan, C. F. N., Grant, P. M., Adaptive Filters, Prentice Hall, Englewood Cliffs, New Jersey, 1985.

[7] Giordano, A. A., Hsu, F. M., Least Square Estimation with applications to Digital Signal Processing, John Wiley & Sons, 1985.

CAPÍTULO 2

ALGORITMOS DE MÍNIMOS QUADRADOS RÁPIDOS

No capítulo anterior expusemos o método de mínimos quadrados e derivamos o Algoritmo RLS, o qual, como vimos, tem uma complexidade proporcional a N^2. Neste capítulo, apresentaremos os algoritmos rápidos, que solucionam exatamente o mesmo problema, mas com uma complexidade proporcional a N.

Segundo um breve histórico feito por Haykin [1], a origem da teoria de mínimos quadrados remonta ao início do século passado com um trabalho de Gauss. O filtro transversal foi primeiramente descrito em 1940 por Kallmann, e o Algoritmo RLS em 1950 por Plackett. Em 1969 um trabalho de Hastings-James e Sage empregava o Algoritmo RLS na identificação em tempo real de parâmetros de sistemas. Somente em 1978 é que se deu o surgimento do primeiro algoritmo rápido com o trabalho pioneiro de Ljung, Morf e Falconer [2]. Este algoritmo, designado por FK (Fast Kalman), será descrito primeiramente. A seguir, serão descritos os algoritmos FAEST e FTF, datados respectivamente de 1983 e 1984, e que representam uma sofisticação do primeiro.

Inicialmente precisamos fazer uma pequena restrição ao problema que estamos considerando. Vamos supor que as amostras x(n) sejam todas nulas para $n \leq 0$. Esta restrição, conhecida como "prewindowing method" [1], não impõe qualquer limitação prática ao problema, já que todo o processo físico deve ter necessariamente um começo, mas é um requisito importante no caso dos algoritmos rápidos.

Como dissemos anteriormente, os algoritmos rápidos exploram a natureza seqüencial do sinal de entrada. Ou seja, o vetor X(n) pode ser obtido, a menos do primeiro elemento, por um simples deslocamento nos elementos de X(n-1). Se existe correlação entre as amostras, é também possível fazer uma estimativa do primeiro elemento de X(n) a partir das amostras anteriores. Esta

estimativa é tanto melhor quanto maior for a correlação existente entre as amostras. O processo de estimar as amostras do sinal é também chamado de predição.

As duas seções que se seguem estão destinadas a entender o processo de predição que, como veremos, constitui a base do desenvolvimento dos algoritmos rápidos.

2.1 – PREDIÇÃO LINEAR PROGRESSIVA

Suponhamos que a amostra x(i) deva ser estimada pela combinação linear

$$\hat{x}(i,n) = A^H(n)X(i-1) \tag{2.1}$$

onde

$$A^T(n) = [a_1(n)\ a_2(n)\ \ldots\ a_N(n)] \tag{2.2}$$

é o vetor de coeficientes de predição progressiva (ou simplesmente, preditor progressivo), a ser obtido pelo critério LS. O termo "progressivo" ficará justificado logo mais adiante. Procedendo como no Exemplo 1.2, desejamos minimizar

$$E_a(n) = \sum_{i=1}^{n} w^{n-i} |\varepsilon_a(i,n)|^2 \tag{2.3}$$

onde fizemos

$$\varepsilon_a(i,n) = x(i) - \hat{x}(i,n) \tag{2.4}$$

$$= x(i) - A^H(n)X(i-1) \tag{2.5}$$

O parâmetro $E_a(n)$ representa um importante papel no contexto dos algoritmos rápidos e pode ser referido como sendo a "energia do erro de predição progressiva". De (2.3) e (2.5) temos

$$E_a(n) = \sum_{i=1}^{n} w^{n-i} [\, |x(i)|^2 - x^*(i)\, A^H(n)X(i-1) - x(i)\, X^H(i-1)A(n) + A^H(n)X(i-1)X^H(i-1)A(n)\,] \tag{2.6}$$

Usando as relações (1.20a), (1.20b) e (1.20c), temos para o gradiente de $E_a(n)$

$$\nabla E_a(n) = -2 \sum_{i=1}^{n} w^{n-i} [\, x^*(i)\, X(i-1) - X(i-1)X^H(i-1)A(n)\,] \tag{2.7}$$

Igualando a zero a expressão acima, temos

$$A(n) = R^{-1}(n-1)r_a(n) \tag{2.8}$$

onde

$$r_a(n) = \sum_{i=1}^{n} w^{n-i} x^*(i) X(i-1) \tag{2.9}$$

é o vetor de autocorrelação progressiva de $\{x(n)\}$.

Da mesma forma que fizemos no Capítulo 1 para o vetor H(n), podemos encontrar uma forma recursiva para a atualização do preditor progressivo, A(n). Usando (2.8) e (2.9), temos

$$A(n) = R^{-1}(n-1)\,[w\, r_a(n-1) + x^*(n)\, X(n-1)] \tag{2.10}$$

e de (2.8) e (1.25)

$$r_a(n-1) = R(n-2)A(n-1) \tag{2.11}$$

$$= \frac{1}{w}\,[R(n-1) - X(n-1)X^H(n-1)]\, A(n-1) \tag{2.12}$$

Substituindo (2.12) em (2.10), vem

$$A(n) = A(n-1) + R^{-1}(n-1)X(n-1)\,[x^*(n) - X^H(n-1)A(n-1)] \tag{2.13}$$

$$= A(n-1) + e_a^*(n)\, G(n-1) \tag{2.14}$$

Na última passagem usamos o vetor de ganho, definido em (1.62), e fizemos

$$e_a(n) = x(n) - A^H(n-1)X(n-1) \tag{2.15}$$

que pode ser referido como sendo o erro de predição a priori. Este erro é o resultado da predição sobre x(n) utilizando o preditor com os coeficientes obtidos no instante n-1. Contrastando, temos o erro de predição a posteriori

$$\varepsilon_a(n) = x(n) - A^H(n)X(n-1) \tag{2.16}$$

resultante da predição com os coeficientes atualizados para o instante n.

Veremos agora que é possível obter uma forma de recorrência para $E_a(n)$ que faz uso dos erros de predição. Substituindo (2.9) e (1.23) em (2.6), temos

$$E_a(n) = \rho(n) - A^H(n)r_a(n) - r_a^H(n)A(n) + A^H(n)R(n-1)A(n) \tag{2.17}$$

onde fizemos

$$\rho(n) = \sum_{i=1}^{n} w^{n-i} |x(i)|^2 \tag{2.18}$$

De (2.8) em (2.17), obtemos

$$E_a(n) = \rho(n) - r_a^H(n)A(n) \tag{2.19}$$

e de (2.14), vem

$$E_a(n) = \rho(n) - r_a^H(n)A(n-1) - e_a^*(n)\, r_a^H(n)G(n-1) \tag{2.20}$$

Usando as definições (2.18) e (2.9), podemos escrever

$$E_a(n) = w\,[\rho(n-1) - r_a^H(n-1)A(n-1)] + x(n)\,[x^*(n) - X^H(n-1)A(n-1)] - e_a^*(n)\, r_a^H(n)G(n-1) \tag{2.21}$$

Reconhecendo a expressão do primeiro colchete de (2.21) em (2.19) e a do segundo colchete em (2.15), temos

$$E_a(n) = w\, E_a(n-1) + e_a^*(n)\,[x(n) - r_a^H(n)G(n-1)] \tag{2.22}$$

Utilizando a propriedade Hermitiana de R(n), a equação (2.8) pode ser reescrita como

$$r_a^H(n) = A^H(n)R(n-1) \tag{2.23}$$

e de (1.62), vem

$$r_a^H(n)G(n-1) = A^H(n)X(n-1) \tag{2.24}$$

Substituindo este resultado em (2.22) e usando a definição (2.16), temos finalmente

$$E_a(n) = w\, E_a(n-1) + e_a^*(n)\varepsilon_a(n) \tag{2.25}$$

Vemos então que a energia do erro de predição progressiva, $E_a(n)$, pode ser atualizada a partir do produto dos erros de predição a priori e a posteriori. É interessante notar que este produto tem sempre um valor real, ou seja $e_a^*(n)\varepsilon_a(n) = e_a(n)\varepsilon_a^*(n)$. Além disso, temos as seguintes desigualdades:

$$e_a^*(n)\varepsilon_a(n) \geq 0 \tag{2.26}$$

$$|\varepsilon_a(n)| \leq |e_a(n)| \tag{2.27}$$

A validade das equações (2.26) e (2.27) será mostrada mais adiante, na Seção 2.4. O fator significativo de (2.27) é a comprovação de que a predição a posteriori fornece uma estimativa melhor ou igual à fornecida pela predição a priori.

Veremos agora um tipo semelhante de predição sobre as amostras de $\{x(n)\}$.

2.2 – PREDIÇÃO LINEAR REGRESSIVA

Na seção anterior vimos como uma dada amostra da seqüência $\{x(n)\}$ pode ser estimada a partir das N amostras precedentes. Queremos agora fazer justamente o contrário: estimar a amostra x(i-N) a partir das N amostras posteriores. Para isso usamos a combinação linear

$$\hat{x}(i\text{-}N,n) = B^H(n)X(i) \tag{2.28}$$

onde

$$B^T(n) = [b_1(n)\ b_2(n)\ \ldots\ b_N(n)] \tag{2.29}$$

é o vetor de coeficientes de predição regressiva (ou simplesmente, preditor regressivo). Definindo a energia do erro de predição regressiva como

$$E_b(n) = \sum_{i=1}^{n} w^{n-i} |\varepsilon_b(i,n)|^2 \tag{2.30}$$

onde

$$\varepsilon_b(i,n) = x(i-N) - \hat{x}(i-N,n) \tag{2.31}$$

$$= x(i-N) - B^H(n)X(i) \tag{2.32}$$

temos, após a minimização de $E_b(n)$

$$B(n) = R^{-1}(n)r_b(n) \tag{2.33}$$

onde

$$r_b(n) = \sum_{i=1}^{n} w^{n-i}\, x^*(i\text{-}N)\, X(i) \tag{2.34}$$

é o vetor de autocorrelação regressiva de $\{x(n)\}$.

Nesta seção estamos omitindo as passagens algébricas, pois são equivalentes às da seção anterior. Analogamente à equação (2.14), temos para o preditor regressivo, a seguinte forma de recorrência

$$B(n) = B(n-1) + e_b^*(n)\, G(n) \tag{2.35}$$

onde $e_b(n)$ é o erro de predição regressiva a priori

$$e_b(n) = x(n-N) - B^H(n-1)X(n) \tag{2.36}$$

O erro de predição regressiva a posteriori é então definido por

$$\varepsilon_b(n) = x(n-N) - B^H(n)X(n) \tag{2.37}$$

e, como feito na derivação de (2.25), pode ser usado para obter $E_b(n)$ recursivamente

$$E_b(n) = w\, E_b(n-1) + e_b(n)\varepsilon_b^*(n) \tag{2.38}$$

Na obtenção de (2.38) surgem também duas relações que serão utilizadas futuramente e, por isso, devem ser mencionadas:

$$E_b(n) = \rho(n-N) - r_b^H(n)B(n) \tag{2.39}$$

e

$$r_b^H(n)G(n) = B^H(n)X(n) \tag{2.40}$$

Estas relações são análogas às equações (2.19) e (2.24).

Uma forma interessante de "enxergar" a predição regressiva é ver o preditor regressivo atuando nas N amostras armazenadas de forma a gerar uma estimativa da amostra que acabou de sair do filtro. Nesse contexto, a predição regressiva faz tanto (ou tão pouco) sentido quanto a predição progressiva, pois ambas procuram estimar algo que já é conhecido. De fato, em nosso caso, não estamos interessados nas estimativas geradas pelos preditores, mais sim nos erros e nos coeficientes de predição. Estes parâmetros serão usados nas recurções dos algoritmos rápidos para reduzir a complexidade computacional exigida pelo Algoritmo RLS.

Podemos agora iniciar a discussão sobre os algoritmos rápidos, começando pelo Algoritmo FK.

2.3 – ALGORITMO FK

Vamos denotar por $X_{N+1}(n)$ o vetor cujos elementos são as N+1 últimas amostras da seqüência $\{x(n)\}$ relativas ao instante n. Assim

$$X_{N+1}^T(n) = [x(n)\ X^T(n-1)] \tag{2.41a}$$

ou

$$X_{N+1}^T(n) = [X^T(n)\ x(n-N)] \tag{2.41b}$$

Da mesma forma, podemos definir a matriz de autocorrelação estendida

$$R_{N+1}(n) = \sum_{i=1}^{n} w^{n-i} X_{N+1}(i) X_{N+1}^H(i) \tag{2.42}$$

Essa definição surge naturalmente como uma extensão da matriz de autocorrelação determinística, definida no Capítulo 1. Usando a forma (2.41a) em (2.42), constatamos que a matriz estendida pode ser particionada como

$$R_{N+1}(n) = \begin{bmatrix} \rho(n) & r_a^H(n) \\ r_a(n) & R(n-1) \end{bmatrix} \tag{2.43a}$$

Se, ao invés de (2.41a), usamos (2.41b), encontramos a partição

$$R_{N+1}(n) = \begin{bmatrix} R(n) & r_b(n) \\ r_b^H(n) & \rho(n-N) \end{bmatrix} \tag{2.43b}$$

Como vimos no Capítulo 1, a eficiência do Algoritmo RLS reside em se utilizar a inversa da matriz de autocorrelação, P(n), ao invés de R(n) nas atualizações. Assim sendo, é interessante obter a inversa de $R_{N+1}(n)$. No Apêndice 2A mostramos em poucas linhas que a inversa da matriz de autocorrelação estendida pode ser escrita como

$$R_{N+1}^{-1}(n) = \begin{bmatrix} 0 & 0_N^T \\ 0_N & R^{-1}(n-1) \end{bmatrix} + \frac{1}{E_a(n)} \begin{bmatrix} 1 & -A^H(n) \\ -A(n) & A(n)A^H(n) \end{bmatrix} \tag{2.44}$$

ou ainda como

$$R_{N+1}^{-1}(n) = \begin{bmatrix} R^{-1}(n) & 0_N \\ 0_N^T & 0 \end{bmatrix} + \frac{1}{E_b(n)} \begin{bmatrix} B(n)B^H(n) & -B(n) \\ -B^H(n) & 1 \end{bmatrix} \tag{2.45}$$

A primeira forma de $R_{N+1}^{-1}(n)$ resulta da partição (2.43a), enquanto que a segunda resulta de (2.43b).

Podemos agora definir o vetor de ganho estendido

$$G_{N+1}(n) = R_{N+1}^{-1}(n)X_{N+1}(n) \tag{2.46}$$

Substituindo as formas (2.41a) e (2.44) em (2.46) e usando a definição do erro de predição progressiva a posteriori, (2.16), temos

$$G_{N+1}(n) = \begin{bmatrix} 0 \\ G(n-1) \end{bmatrix} + \frac{\varepsilon_a(n)}{E_a(n)} \begin{bmatrix} 1 \\ -A(n) \end{bmatrix} \tag{2.47}$$

Se por outro lado, usamos as formas (2.41b), (2.45) e o erro de predição regressiva a posteriori, (2.37), obtemos

$$G_{N+1}(n) = \begin{bmatrix} G(n) \\ 0 \end{bmatrix} + \frac{\varepsilon_b(n)}{E_b(n)} \begin{bmatrix} -B(n) \\ 1 \end{bmatrix} \tag{2.48}$$

Uma breve análise de (2.47) e (2.48) permite entender o funcionamento dos algoritmos rápidos. Com algumas poucas manipulações é possível atualizar diretamente o vetor de ganho, G(n), evitando-se o uso explicito da matriz inversa, $R^{-1}(n)$. Uma forma prática de se conseguir isso consiste em fazer a partição

$$G_{N+1}(n) = \begin{bmatrix} M(n) \\ m(n) \end{bmatrix} \tag{2.49}$$

onde M(n) é um vetor de N elementos e m(n) é um escalar. De (2.48) temos

$$m(n) = \frac{\varepsilon_b(n)}{E_b(n)} \tag{2.50}$$

Contudo, é bom notar que esta não é a única forma de se calcular m(n). Ele pode também ser obtido como o último elemento do vetor $G_{N+1}(n)$ em (2.47). Usando novamente a definição (2.49) em (2.48), temos

$$G(n) = M(n) + m(n)\, B(n) \tag{2.51}$$

Esta equação não representa ainda uma solução de recorrência para o problema. Isto porque existe uma outra relação entre B(n) e G(n). Como vimos na Seção 2.2

$$B(n) = B(n\text{-}1) + e_b^*(n)\, G(n) \tag{2.52}$$

Substituindo (2.52) em (2.51), obtemos

$$G(n) = \frac{1}{1 - m(n)e_b^*(n)} [M(n) + m(n)\, B(n\text{-}1)] \tag{2.53}$$

Se, ao contrário , substituirmos (2.51) em (2.52), teremos

$$B(n) = \frac{1}{1 - m(n)e_b^*(n)} [B(n-1) + e_b^*(n)M(n)] \tag{2.54}$$

Podemos então resolver o problema de duas formas. A primeira delas, utiliza as equações (2.53) e (2.52) e atualiza o vetor G(n) antes de B(n). A segunda, usa (2.54) e (2.51) e atualiza B(n) antes de G(n).

Vamos agora atentar para o termo que aparece no denominador de (2.53) e (2.54). Usando a definição de m(n) temos

$$1 - m(n)e_b^*(n) = \frac{E_b(n) - \varepsilon_b(n)e_b^*(n)}{E_b(n)} \tag{2.55}$$

Lembrando que o produto $\varepsilon_b(n)e_b^*(n)$ é real, podemos substituir (2.38) em (2.55), obtendo

$$1 - m(n)e_b^*(n) = \frac{w\, E_b(n-1)}{E_b(n)} \tag{2.56}$$

Vemos então que o termo $1 - m(n)e_b^*(n)$ é a razão entre dois valores sucessivos da energia do erro de predição regressiva multiplicada por w, e tem um valor real e positivo. Além disso, sabemos de (2.26) que o produto $\varepsilon_b(n)e_b^*(n)$ é real e não negativo. Assim, de (2.55), temos que $1 - m(n)e_b^*(n)$ é menor que 1. O campo de variação desse termo é então dado por

$$0 < 1 - m(n)e_b^*(n) \leq 1 \tag{2.57}$$

Uma criteriosa seleção das equações apresentadas acima leva ao Algoritmo FK, o qual é exposto na Tabela 2.1. Este algoritmo requer 9N + 2 somas, 10N + 3 multiplicações e duas divisões por iteração. Esses números são obtidos através de um criterioso aproveitamento das operações envolvidas, evitando-se operações redundantes e, novamente, ignorando a natureza genericamente complexa das variáveis.

Visto que a complexidade computacional do Algoritmo FK aumenta linearmente com N (e não com N^2), observa-se um considerável ganho, em termos de esforço computacional, em relação ao Algoritmo RLS. Comparando-se os números de operações requeridas por cada um dos algoritmos, observa-se que o primeiro apresenta, de fato, vantagem computacional a partir de N = 6. Vantagem esta que é tanto mais válida quanto maior for a ordem do filtro transversal.

Veremos a seguir dois outros algoritmos rápidos, que executam a mesma função do Algoritmo FK, mas com uma complexidade computacional ainda inferior à deste.

Tabela 2.1 - Algoritmo FK

disponível, do instante n-1

$$X(n-1),\ H(n-1),\ A(n-1),\ B(n-1),\ G(n-1),\ E_a(n-1)$$

recebido no instante n

$$x(n),\ d(n)$$

atualização dos preditores

$$e_a(n) = x(n) - A^H(n-1)X(n-1)$$

$$A(n) = A(n-1) + e_a^*(n)\, G(n-1)$$

$$\varepsilon_a(n) = x(n) - A^H(n)X(n-1)$$

$$E_a(n) = w\, E_a(n-1) + e_a^*(n)\varepsilon_a(n)$$

$$\begin{bmatrix} M(n) \\ m(n) \end{bmatrix} = \begin{bmatrix} 0 \\ G(n-1) \end{bmatrix} + \frac{\varepsilon_a(n)}{E_a(n)} \begin{bmatrix} 1 \\ -A(n) \end{bmatrix}$$

$$e_b(n) = x(n-N) - B^H(n-1)X(n)$$

$$G(n) = \frac{1}{1 - m(n)e_b^*(n)}\, [M(n) + m(n)\, B(n-1)]$$

$$B(n) = B(n-1) + e_b^*(n)\, G(n)$$

atualização do filtro adaptativo

$$e(n) = d(n) - H^H(n-1)X(n)$$

$$H(n) = H(n-1) + e^*(n)\, G(n)$$

2.4 – ALGORITMO FTF

Nesta seção iremos apresentar o Algoritmo FTF (Fast Transversal Filter). Este algoritmo foi proposto por Cioffi e Kailath em 1984 [3], e consiste basicamente de modificações do Algoritmo FK. Este também é o caso do Algoritmo FAEST, proposto por Carayannis, Manolakis e Kalouptsidis em 1983 [4], que veremos na próxima seção.

Começaremos definindo um importante parâmetro

$$\gamma(n) = 1 - G^H(n)X(n) \tag{2.58}$$

Usando a definição do vetor de ganho e o fato de que R(n) é Hermitiana, podemos reescrever (2.58) como

$$\gamma(n) = 1 - X^H(n)R^{-1}(n)X(n) \tag{2.59}$$

Notando o surgimento da forma Hermitiana em (2.59), vemos que $\gamma(n)$ é um escalar de valor real. Substituindo a equação de recorrência (1.66) para a matriz inversa, $R^{-1}(n)$, em (2.59), temos, após as simplificações

$$\gamma(n) = \frac{w}{w + X^H(n)R^{-1}(n-1)X(n)} \tag{2.60}$$

Lembrando que a forma Hermitiana no denominador de (2.60) tem um valor não negativo, concluímos que os limites de $\gamma(n)$ são dados por

$$0 \leq \gamma(n) \leq 1 \tag{2.61}$$

Vamos agora analisar melhor a variável $\gamma(n)$. Tomemos para isso a definição (1.55), do erro a posteriori

$$\varepsilon(n) = d(n) - H^H(n)X(n) \tag{2.62}$$

Usando (1.61) e (1.54), temos

$$\varepsilon(n) = e(n)\,[1 - G^H(n)X(n)] \tag{2.63}$$

Notando que o termo em colchetes é a definição de $\gamma(n)$, podemos finalmente escrever

$$\gamma(n) = \frac{\varepsilon(n)}{e(n)} \tag{2.64}$$

Resultados semelhantes podem ser obtidos considerando-se os erros de predição. Para a predição progressiva temos

$$\gamma(n\text{-}1) = \frac{\varepsilon_a(n)}{e_a(n)} \tag{2.65}$$

e para a predição regressiva, temos

$$\gamma(n) = \frac{\varepsilon_b(n)}{e_b(n)} \tag{2.66}$$

Estas duas últimas equações são derivadas de forma análoga à (2.64). A interpretação de $\gamma(n)$ como uma razão entre os erros de predição introduz um fato surpreendente: os erros a posteriori podem ser obtidos antes mesmo do que os filtros que os determinam. Alguns autores, em virtude deste fato, denominam $\gamma(n)$ como *variável de antecipação*. Outro nome mais freqüente para $\gamma(n)$ é *variável de verossimilhança*.

Considerando a equação (2.64), e lembrando que $\gamma(n)$ é um escalar real, positivo e menor que 1, ficam provados dois argumentos usados anteriormente: $|\varepsilon(n)| \leq |e(n)|$ e $\varepsilon(n)e^*(n) \geq 0$. Argumentos que, similarmente, valem para os erros de predição progressiva e regressiva.

Vamos agora desenvolver alguns conceitos introduzidos no Capítulo 1. Como discutimos no Exemplo 1.2, uma seqüência de amostras {d(i)} pode ser aproximada por uma combinação linear de N vetores X(i-k), k = 0, 1, . . , N-1. Denotamos por H(n) o vetor cujos elementos são os coeficientes dessa combinação linear, e por ε(n) o erro na estimação da amostra d(n). Queremos agora obter H(n) e ε(n) para um caso particular, em que {d(i)} é a seqüência dada por

$$d(i) = \begin{cases} 1 & \text{se } i = n \\ 0 & \text{se } i \leq n \end{cases} \tag{2.67}$$

Das equações (1.22), (1.24) e (1.62) resulta, para esta seqüência particular, que H(n) = G(n). Ou seja, o vetor de coeficientes que minimiza o erro ao quadrado para a seqüência definida em (2.67) é precisamente o vetor de ganho, G(n). A utilização de H(n) = G(n) como um filtro para calcular o erro residual fornece, das definições (1.55) e (2.58), ε(n) = γ(n). Nesse sentido, G(n) pode ser entendido como sendo também um filtro transversal. Neste caso, γ(n) é o erro residual gerado por este filtro.

Alguns autores costumam descrever os algoritmos rápidos como sendo formado por um conjunto de quatro filtros transversais, H(n), A(n), B(n) e G(n), todos excitados pela mesma seqüência de entrada, {x(n)}. Essa idéia é mais clara no Algoritmo FTF, pois neste caso o parâmetro γ(n), que representa o erro relativo ao filtro G(n), é utilizado explicitamente, o que não acontece no Algoritmo FK.

Podemos agora definir o vetor de ganho dual, que será utilizado nos algoritmos rápidos FTF e FAEST

$$\bar{G}(n) = \frac{1}{\gamma(n)} G(n) \tag{2.68}$$

Com esta definição, as equações (1.61), (2.14) e (2.35) podem ser reescritas como

$$H(n) = H(n-1) + \varepsilon^*(n)\, \bar{G}(n) \tag{2.69}$$

$$A(n) = A(n-1) + \varepsilon_a^*(n)\, \bar{G}(n-1) \tag{2.70}$$

$$B(n) = B(n-1) + \varepsilon_b^*(n)\, \bar{G}(n) \tag{2.71}$$

Pretendemos usar $\bar{G}(n)$ ao invés de G(n) no tratamento a seguir. De (2.60) e (1.67) em (2.68), decorre uma definição equivalente para o vetor de ganho dual

$$\bar{G}(n) = \frac{1}{w} R^{-1}(n-1)X(n) \tag{2.72}$$

Assim como fizemos para G(n) na seção anterior, vamos definir o vetor de ganho estendido dual

$$\bar{G}_{N+1}(n) = \frac{1}{w} R_{N+1}^{-1}(n-1)X_{N+1}(n) \tag{2.73}$$

A matriz de autocorrelação estendida, no instante n-1, pode ser reescrita, de (2.44), como

$$R_{N+1}^{-1}(n-1) = \begin{bmatrix} 0 & 0_N^T \\ 0_N & R^{-1}(n-2) \end{bmatrix} + \frac{1}{E_a(n-1)} \begin{bmatrix} 1 & -A^H(n-1) \\ -A(n-1) & A(n-1)A^H(n-1) \end{bmatrix} \tag{2.74}$$

ou, de (2.45), como

$$R_{N+1}^{-1}(n-1) = \begin{bmatrix} R^{-1}(n-1) & 0_N \\ 0_N^T & 0 \end{bmatrix} + \frac{1}{E_b(n-1)} \begin{bmatrix} B(n-1)B^H(n) & -B(n-1) \\ -B^H(n-1) & 1 \end{bmatrix} \tag{2.75}$$

Usando (2.74), (2.41a) e a definição de erro de predição progressiva a priori em (2.73), temos

$$G_{N+1}(n) = \begin{bmatrix} 0 \\ G(n-1) \end{bmatrix} + \frac{e_a(n)}{w\, E_a(n-1)} \begin{bmatrix} 1 \\ -A(n-1) \end{bmatrix} \tag{2.76}$$

Se, ao contrário, usamos (2.75), (2.41b) e a definição do erro de predição regressiva a priori, temos

$$G_{N+1}(n) = \begin{bmatrix} G(n) \\ 0 \end{bmatrix} + \frac{e_b(n)}{w\, E_b(n-1)} \begin{bmatrix} -B(n-1) \\ 1 \end{bmatrix} \tag{2.77}$$

Definindo

$$G_{N+1}(n) = \begin{bmatrix} \bar{M}(n) \\ \tilde{m}(n) \end{bmatrix} \tag{2.78}$$

onde $\bar{M}(n)$ é um vetor de N elementos e $\tilde{m}(n)$ é um escalar, temos que

$$\tilde{m}(n) = \frac{e_b(n)}{w\, E_b(n-1)} \tag{2.79}$$

De (2.77), (2.78) e (2.79), podemos escrever

$$G(n) = \bar{M}(n) + \tilde{m}(n)\, B(n-1) \tag{2.80}$$

A equação (2.79) permite obter $e_b(n)$ evitando-se uma operação vetorial. Veremos no Capítulo 3 que este procedimento não é aconselhável por questões de instabilidade numérica devida a erros de arredondamentos nos sistemas de computação. Assumindo, no entanto, um processamento com precisão infinita, temos

$$e_b(n) = w\, \tilde{m}(n)\, E_b(n-1) \tag{2.81}$$

Resta agora encontrar uma forma de atualizar $\gamma(n)$. Para isso vamos definir a variável auxiliar

$$\gamma_1(n) = 1 - G_{N+1}^H(n)X_{N+1}(n) \tag{2.82}$$

Esta variável desempenha o mesmo papel que $\gamma(n)$, porém, no espaço vetorial estendido. Assim

$$\bar{G}_{N+1}(n) = \frac{1}{\gamma_1(n)} G_{N+1}(n) \tag{2.83}$$

Usando (2.47), (2.41a) e a definição do erro de predição progressiva a posteriori em (2.82), obtemos

$$\gamma_1(n) = \gamma(n-1) - \frac{|\varepsilon_a(n)|^2}{E_a(n)} \tag{2.84}$$

De (2.25) e (2.65), podemos escrever

$$|\varepsilon_a(n)|^2 = \gamma(n-1)\,[E_a(n) - w\,E_a(n-1)] \tag{2.85}$$

Substituindo este resultado em (2.84), temos

$$\gamma_1(n) = \gamma(n-1)\,\frac{w\,E_a(n-1)}{E_a(n)} \tag{2.86}$$

Similarmente à obtenção de (2.86), podemos usar (2.48) e (2.41b) em (2.82) resultando, após outras substituições

$$\gamma_1(n) = \gamma(n)\,\frac{w\,E_b(n-1)}{E_b(n)} \tag{2.87}$$

As equações (2.86) e (2.87) permitem atualizar $\gamma(n)$ através de $\gamma_1(n)$. Contudo, (2.87) pode sofrer ainda uma modificação. Usando (2.56), vem

$$\gamma_1(n) = \gamma(n)\,[1 - m(n)e_b^*(n)] \tag{2.88}$$

Lembrando que

$$\tilde{m}(n) = \frac{1}{\gamma_1(n)}\, m(n) \tag{2.89}$$

temos finalmente

$$\gamma(n) = \frac{\gamma_1(n)}{1 - \gamma_1(n)\tilde{m}(n)e_b^*(n)} \tag{2.90}$$

Tabela 2.2 - Algoritmo FTF

disponível, do instante n-1

$X(n-1), H(n-1), A(n-1), B(n-1), \bar{G}(n-1), \gamma(n-1), E_a(n-1), E_b(n-1)$

recebido no instante n

$x(n), d(n)$

atualização dos preditores

$$e_a(n) = x(n) - A^H(n-1)X(n-1)$$

$$\begin{bmatrix} \bar{M}(n) \\ \tilde{m}(n) \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{G}(n-1) \end{bmatrix} + \frac{e_a(n)}{w\, E_a(n-1)} \begin{bmatrix} 1 \\ -A(n-1) \end{bmatrix}$$

$$\varepsilon_a(n) = \gamma(n-1)\, e_a(n)$$

$$A(n) = A(n-1) + \varepsilon_a^*(n)\, \bar{G}(n-1)$$

$$\bar{G}(n) = \bar{M}(n) + \tilde{m}(n)\, B(n-1)$$

$$E_a(n) = w\, E_a(n-1) + e_a(n)\varepsilon_a^*(n)$$

$$e_b(n) = w\, E_b(n-1)\, \tilde{m}(n)$$

$$\gamma_1(n) = \gamma(n-1)\, \frac{w\, E_a(n-1)}{E_a(n)}$$

$$\gamma(n) = \frac{\gamma_1(n)}{1 - \gamma_1(n)\tilde{m}(n)e_b^*(n)}$$

$$\varepsilon_b(n) = \gamma(n)\, e_b(n)$$

$$B(n) = B(n-1) + \varepsilon_b^*(n)\, \bar{G}(n)$$

$$E_b(n) = w\, E_b(n-1) + e_b(n)\varepsilon_b^*(n)$$

atualização do filtro adaptativo

$$e(n) = d(n) - H^H(n-1)X(n)$$

$$H(n) = H(n-1) + \gamma(n)\, e^*(n)\, \bar{G}(n)$$

O Algoritmo FTF pode ser construído de diversas formas, a partir das equações derivadas acima. Essa multiplicidade de implementações deve-se exclusivamente ao número de arranjos permitidos no referido conjunto de equações. Na Tabela 2.2 apresentamos um destes arranjos. Achamos que essa forma é bastante conveniente, sobretudo com relação à praticidade no momento de converter o algoritmo para uma linguagem de programação.

A demanda computacional do Algoritmo FTF é de 7N + 3 somas, 7N + 11 multiplicações e 2 divisões por iteração. Números estes, que foram obtidos com a utilização da variável auxiliar $\gamma_1(n)\, E_a(n)$ ao invés de $\gamma_1(n)$. Nota-se que eles representam um ganho em relação ao Algoritmo FK.

2.5 – ALGORITMO FAEST

Historicamente, o Algoritmo FAEST (Fast a Posteriori Error Sequential Technique) precedeu o Algoritmo FTF por um curto espaço de tempo. Preferimos inverter essa ordem cronológica na apresentação por motivos meramente didáticos. De fato, estes dois algoritmos são muito semelhantes, diferindo apenas no uso da variável de verossimilhança. Ao invés de $\gamma(n)$, o FAEST emprega

$$\zeta(n) = \frac{1}{\gamma(n)} \tag{2.91}$$

Assim, de (2.64), (2.65) e (2.66), temos

$$\varepsilon(n) = \frac{e(n)}{\zeta(n)} \tag{2.92}$$

$$\varepsilon_a(n) = \frac{e_a(n)}{\zeta(n-1)} \tag{2.93}$$

e

$$\varepsilon_b(n) = \frac{e_b(n)}{\zeta(n)} \tag{2.94}$$

Também a variável auxiliar $\gamma_1(n)$ é substituída por

$$\zeta_1(n) = \frac{1}{\gamma_1(n)} \tag{2.95}$$

De (2.95), (2.86) e (2.91) temos

$$\zeta_1(n) = \zeta(n\text{-}1)\,\frac{E_a(n)}{w\,E_a(n-1)} \tag{2.96}$$

Usando (2.25) e (2.93) vem

$$\zeta_1(n) = \zeta(n\text{-}1) + \frac{|e_a(n)|^2}{w\,E_a(n-1)} \tag{2.97}$$

Por outro lado, de (2.90), (2.95) e (2.91), podemos escrever

$$\zeta(n) = \zeta_1(n) - \tilde{m}(n)e_b^*(n) \tag{2.98}$$

As equações (2.97) e (2.98) permitem atualizar $\zeta(n)$ no Algoritmo FAEST da mesma forma que é feito para $\gamma(n)$ no FTF. A Tabela 2.3 ilustra o Algoritmo FAEST, que pode ser implemen-

Tabela 2.3 - Algoritmo FAEST

disponível, do instante n - 1

$$X(n-1),\ H(n-1),\ A(n-1),\ B(n-1),\ \bar{G}(n-1),\ \zeta(n-1),\ E_a(n-1),\ E_b(n-1)$$

recebido no instante n

$$x(n),\ d(n)$$

atualização dos preditores

$$e_a(n) = x(n) - A^H(n-1)X(n-1)$$

$$\begin{bmatrix} \bar{M}(n) \\ \tilde{m}(n) \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{G}(n-1) \end{bmatrix} + \frac{e_a(n)}{w\, E_a(n-1)} \begin{bmatrix} 1 \\ -A(n-1) \end{bmatrix}$$

$$\varepsilon_a(n) = \frac{e_a(n)}{\zeta(n-1)}$$

$$A(n) = A(n-1) + \varepsilon_a^*(n)\, \bar{G}(n-1)$$

$$\bar{G}(n) = \bar{M}(n) + \tilde{m}(n)\, B(n-1)$$

$$E_a(n) = w\, E_a(n-1) + e_a(n)\varepsilon_a^*(n)$$

$$e_b(n) = w\, E_b(n-1)\, \tilde{m}(n)$$

$$\zeta_1(n) = \zeta(n-1) + \frac{|e_a(n)|^2}{w\, E_a(n-1)}$$

$$\zeta(n) = \zeta_1(n) - \tilde{m}(n) e_b^*(n)$$

$$\varepsilon_b(n) = \frac{e_b(n)}{\zeta(n)}$$

$$B(n) = B(n-1) + \varepsilon_b^*(n)\, \bar{G}(n)$$

$$E_b(n) = w\, E_b(n-1) + e_b(n)\varepsilon_b^*(n)$$

atualização do filtro adaptativo

$$e(n) = d(n) - H^H(n-1)X(n)$$

$$H(n) = H(n-1) + \frac{e_b^*(n)}{\zeta(n)}\, \bar{G}(n)$$

tado com 7N + 4 somas, 7N + 10 multiplicações e duas divisões por iteração. Uma complexidade equivalente à apresentada pelo Algoritmo FTF. Na verdade, estes dois algoritmos representam formas diferentes do que podemos chamar de "versão computacionalmente mais eficiente dos algoritmos rápidos". A diferença entre eles reside unicamente na forma de se calcular a variável de verossimilhança.

Os três algoritmos descritos neste capítulo representam os tipos básicos de algoritmos rápidos. Pequenas variações e combinações destes algoritmos também são encontradas na literatura especializada.

2.6 – CONDIÇÕES INICIAIS

Assim como o Algoritmo RLS, os algoritmos rápidos também exigem certos cuidados na inicialização. A estratégia na escolha dos valores iniciais reside em se preservar as condições de mínimos quadrados. Ou seja, o algoritmo deve partir de uma situação válida no âmbito do critério LS. O grande número de variáveis existentes permite uma ampla liberdade na arbitragem destes valores, uma vez que as condições de contorno são poucas. Entretanto, como estamos admitindo que as amostras da seqüência de entrada são todas nulas para $n \leq 0$, é razoável escolher $A(0) = 0_N$ e $B(0) = 0_N$ para os preditores. Também, dado a invariância destes nos instantes $n \leq 0$, podemos fazer $G(0) = 0_N$ para o vetor de ganho. Coerente com a escolha de G(0), a definição (2.58) estabelece $\gamma(0) = 1$ para o Algoritmo FTF. Assim, de (2.68), $\bar{G}(0) = 0_N$ e $\zeta(0) = 1$ para o Algoritmo FAEST. Resta somente arbitrar um valor inicial para as energias dos erros de predição.

É um fato inerente aos algoritmos rápidos que as variáveis $E_a(n)$ e $E_b(n)$ nunca devem assumir o valor zero, pois provocaria um denominador nulo. Dessa forma, uma constante positiva deve ser escolhida para $E_a(0)$ e outra para $E_b(0)$. Obviamente, estas duas constantes devem manter determinada relação. Das equações (2.86) e (2.87), podemos escrever

$$\gamma(n)\frac{E_a(n)}{E_b(n)} = \gamma(n-1)\frac{E_a(n-1)}{E_b(n-1)} \tag{2.99}$$

De onde concluímos que ambos os membros de (2.99) representam, na verdade, uma constante. O valor desta constante pode ser facilmente obtido particularizando-se a seqüência de entrada. Suponhamos para isso que $\{x(n)\}$ seja uma seqüência de amostras na qual $x(-N)$ é a única amostra não nula. Aplicando o Algoritmo FK nesta seqüência, obtemos

$$E_a(n) = w^{N+n}\,|x(-N)|^2 \qquad n > 0 \tag{2.100}$$

$$E_b(n) = w^{n}\,|x(-N)|^2 \tag{2.101}$$

$$\gamma(n) = 1 \tag{2.102}$$

Assim

$$\gamma(n)\frac{E_a(n)}{E_b(n)} = w^{N} \tag{2.103}$$

Esta relação é válida em geral, e não apenas para a seqüência acima. Uma prova analiticamente mais rigorosa da equação (2.103) é apresentada no Apêndice 2B. Lembrando que estamos assumindo $\gamma(0) = 1$, podemos fazer $E_a(0) = E_o$ e $E_b(0) = w^{-N} E_o$. A constante E_o deve ser escolhida com algum critério para não comprometer a estabilidade do algoritmo nos instantes iniciais. Abordaremos melhor esse aspecto no Capítulo 3. Nota-se que a influência de E_o decai exponencialmente segundo a equação (2.100), tornando-se desprezível após algumas constantes de tempo [vide equação (1.30)]. Na Tabela 2.4 reunimos os valores iniciais discutidos acima. Observe que fizemos $H(0) = H_o$ para o filtro adaptativo, pois é possível que uma estimativa a priori esteja disponível no instante de partida

Tabela 2.4 - Condições iniciais

$A(0) = 0_N$

$B(0) = 0_N$

$G(0) = \bar{G}(0) = 0_N$

$\gamma(0) = \zeta(0) = 1$

$E_a(0) = E_o$

$E_b(0) = w^{-N}E_o$

$H(0) = H_o$

2.7 – ESPAÇOS VETORIAIS PARA LS

É possível derivar os algoritmos rápidos utilizando conceitos geométricos. Ainda que traga uma certa redundância ao trabalho, achamos conveniente a inclusão deste tópico, devido ao poder de visualização que a argumentação geométrica confere ao problema. Por outro lado, a repetição do desenvolvimento sob um outro prisma ajuda a fixar o intrincado mecanismo de funcionamento dos algoritmos rápidos. A abordagem que iremos adotar é semelhante à encontrada em [5]. Um tratamento ligeiramente diferente e cuidadosamente elaborado pode também ser encontrado em [6].

Sem dúvida, a utilização de variáveis complexas não condiz com o nosso objetivo de adquirir uma visão espacial do problema. Por isso vamos particularizar o tratamento para o caso de variáveis reais. Faremos ainda uma outra simplificação: admitiremos $w = 1$. O espaço vetorial com o qual estaremos envolvidos tem dimensão M, onde M é grande o suficiente para abrigar um número de eixos coordenados maior que o número das amostras passadas. Em outras palavras, $M > n$. Definimos então os vetores Mx1

$$X_M^T(n) = [x(n)\ x(n-1)\ \ldots\ x(1)\ 0\ \ldots\ 0] \tag{2.104}$$

e

$$D_M^T(n) = [d(n)\ d(n-1)\ \ldots\ d(1)\ 0\ \ldots\ 0] \tag{2.105}$$

a matriz MxN

$$X_{MN}(n) = [X_M(n)\ X_M(n-1)\ \ldots\ X_M(n-N+1)] \tag{2.106}$$

e o vetor de erro

$$\varepsilon_M(n) = D_M(n) - Y_M(n) \tag{2.107}$$

onde

$$Y_M(n) = X_{MN}(n)H(n) \tag{2.108}$$

Assim, a função custo, expressa por (1.17)

$$E_N(n) = \sum_{i=1}^{n} [d(i) - X^T(i)H(n)]^2 \tag{2.109}$$

pode ser reescrita como

$$E_N(n) = \varepsilon_M^T(n)\varepsilon_M(n) \tag{2.110}$$

$$= \| \varepsilon_M(n) \|^2 \tag{2.111}$$

Vemos então que o parâmetro a ser otimizado pelo critério LS representa a norma Euclidiana, ou o comprimento ao quadrado, do vetor $\varepsilon_M(n)$.

Minimizar o comprimento ao quadrado de $\varepsilon_M(n)$ equivale a minimizar o comprimento de $\varepsilon_M(n)$, que é a distância entre $Y_M(n)$ e $D_M(n)$ no espaço de M dimensões. Uma vez que $Y_M(n)$ é um vetor formado pela combinação linear das colunas de $X_{MN}(n)$, está contido no subespaço vetorial gerado por elas. Como $D_M(n)$ em geral não pertence a este subespaço, o comprimento de $\varepsilon_M(n)$ é mínimo quando este vetor situa-se ortogonal ao subespaço gerado por $X_M(n-i)$, $i = 0, 1, \ldots, N-1$. Dessa forma, podemos escrever

$$X_M^T(n-i)\varepsilon_M(n) = 0 \qquad i = 0, 1, \ldots, N-1 \tag{2.112}$$

ou ainda

$$X_{MN}^T(n)\varepsilon_M(n) = 0_N \tag{2.113}$$

As equações (2.112) e (2.113) representam o "princípio da ortogonalidade", já mencionado na Seção 1.1. Este argumento permite obter o vetor de coeficientes de uma forma bastante direta, contrastando com a utilizada no Capítulo 1. Substituindo as equações (2.107) e (2.108) em (2.113), temos

$$H(n) = [X_{MN}^T(n)X_{MN}(n)]^{-1} X_{MN}^T(n)D_M(n) \tag{2.114}$$

Este resultado confere com o obtido na equação (1.22) usando derivação de vetores pois, para $w = 1$, decorre

$$X_{MN}^T(n)X_{MN}(n) = R(n) \tag{2.115}$$

e

$$X_{MN}^{T}(n)D_M(n) = r_{dx}(n) \tag{2.116}$$

Usando o vetor H(n) dado por (2.114) em (2.108), temos

$$Y_M(n) = X_{MN}(n)\,[X_{MN}^{T}(n)X_{MN}(n)]^{-1}\,X_{MN}^{T}(n)D_M(n) \tag{2.117}$$

e, de (2.107), vem

$$\varepsilon_M(n) = \{I_M - X_{MN}(n)\,[X_{MN}^{T}(n)X_{MN}(n)]^{-1}\,X_{MN}^{T}(n)\}\,D_M(n) \tag{2.118}$$

O vetor $Y_M(n)$ pode ser visto como sendo a projeção de $D_M(n)$ ao longo do subespaço vetorial gerado pelas colunas de $X_{MN}(n)$. Da mesma forma, $\varepsilon_M(n)$ representa a projeção de $D_M(n)$ ortogonal a este subespaço. Este fato sugere a definição dos operadores

$$P_X(n) = X_{MN}(n)\,[X_{MN}^{T}(n)X_{MN}(n)]^{-1}\,X_{MN}^{T}(n) \tag{2.119}$$

e

$$P_X^{o}(n) = I_M - P_X(n) \tag{2.120}$$

os quais projetam um vetor respectivamente "ao longo do" e "ortogonalmente" ao subespaço gerado por $X_M(n\text{-}i)$, $i = 0, 1, \ldots, N\text{-}1$. Usando estas definições, podemos escrever

$$Y_M(n) = P_X(n)D_M(n) \tag{2.121}$$

e

$$\varepsilon_M(n) = P_X^{o}(n)D_M(n) \tag{2.122}$$

Vamos estabelecer duas propriedades dos operadores de projeção, que decorrem diretamente das definições acima:

Propriedade 2.7.1

As matrizes que definem os operadores de projeção são simétricas

$$P_X^{T}(n) = P_X(n) \tag{2.123a}$$

$$P_X^{oT}(n) = P_X^{o}(n) \tag{2.123b}$$

Propriedade 2.7.2

As matrizes que definem os operadores de projeção são idempotentes

$$P_X(n)P_X(n) = P_X(n) \tag{2.124a}$$

$$P_X^o(n)P_X^o(n) = P_X^o(n) \tag{2.124b}$$

O mesmo tratamento que demos acima ao filtro transversal, H(n), pode ser dado aos preditores. No caso do preditor progressivo, definimos

$$\varepsilon_{aM}(n) = X_M(n) - X_{MN}(n-1)A(n) \tag{2.125}$$

Esta definição é análoga à do vetor de erro $\varepsilon_M(n)$. Usando novamente o princípio da ortogonalidade, temos que o comprimento mínimo de $\varepsilon_{aM}(n)$ ocorre quando

$$X_{MN}^T(n-1)\varepsilon_{aM}(n) = 0 \tag{2.126}$$

Substituindo (2.125) em (2.126), obtemos o vetor de coeficientes do preditor progressivo

$$A(n) = [X_{MN}^T(n-1)X_{MN}(n-1)]^{-1} X_{MN}^T(n-1)X_M(n) \tag{2.127}$$

Usando este resultado em (2.125), temos

$$\varepsilon_{aM}(n) = P_X^o(n-1)X_M(n) \tag{2.128}$$

Como era de se esperar, $\varepsilon_{aM}(n)$ representa a projeção ortogonal de $X_M(n)$ no subespaço gerado pelos vetores $X_M(n-i)$, i = 1, 2, . . , N. Esta base difere da utilizada anteriormente pela troca de um vetor: o vetor $X_M(n-N)$, que se encontra no lugar de $X_M(n)$. Para o caso do preditor regressivo, a solução obtida é

$$\varepsilon_{bM}(n) = P_X^o(n)X_M(n-N) \tag{2.129}$$

onde $\varepsilon_{bM}(n)$ é o vetor Mx1, cujo primeiro elemento representa o erro de predição regressiva a posteriori, $\varepsilon_b(n)$.

É possível estender a ordem do subespaço associado aos operadores de projeção, de modo que este comporte um número maior de dimensões. Seja então Z_{MK} uma matriz MxK e seja

$$W_{MK} = P_X^o Z_{MK} \tag{2.130}$$

uma matriz cujas colunas são vetores contidos no subespaço de Z_{MK}, os quais se encontram ortogonais ao subespaço de X_{MN} (A omissão do índice em parênteses objetiva unicamente simplificar as expressões). Temos então

$$P_{X,Z} = P_X + P_W \tag{2.131}$$

onde $P_{X,Z}$ é o operador que dá a projeção ao longo do subespaço gerado pelas colunas de X_{MN} e Z_{MK}. De (2.130) e (2.119), vem

$$P_{X,Z} = P_X + P_X^o Z [Z^T P_X^o Z]^{-1} Z^T P_X^o \tag{2.132}$$

onde usamos as propriedades 2.7.1 e 2.7.2, e omitimos também o sub-índice de Z_{MK}, que pode ser um vetor Mx1. Substituindo (2.120) em (2.132), temos finalmente

$$P_{X,Z}^o = P_X^o - P_X^o Z [Z^T P_X^o Z]^{-1} Z^T P_X^o \tag{2.133}$$

Queremos agora encontrar uma forma para a atualização temporal do operador $P_X^o(n)$. Antes porém, precisamos definir a matriz de deslocamento

$$S = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{2.134}$$

e o vetor Mx1

$$\sigma^T = [1 \ 0 \ 0 \dots 0] \tag{2.135}$$

A aplicação da matriz S em um vetor M-dimensional desloca seus elementos de uma unidade, provocando um "atraso" no índice temporal. Por exemplo:

$$SX_M(n) = X_M(n-1) \tag{2.136}$$

$$SX_{MN}(n) = X_{MN}(n-1) \tag{2.137}$$

o vetor σ, ao contrário, pode ser usado para obter a primeira linha de uma matriz, ou o elemento mais recente de um vetor. Por exemplo:

$$\sigma^T X_M(n) = x(n) \tag{2.138}$$

$$\sigma^T X_{MN}(n) = X^T(n) \tag{2.139}$$

Na discussão que faremos a seguir, usaremos também a relação

$$I_M = S^T S + \sigma\sigma^T \tag{2.140}$$

que decorre trivialmente das definições acima. Fazendo $Z=\sigma$ na equação (2.133), temos

$$P_{X,\sigma}^o = P_X^o - P_X^o \sigma [\sigma^T P_X^o \sigma]^{-1} \sigma^T P_X^o \tag{2.141}$$

Notando que na derivação de (2.133) poderíamos ter feito Z ser uma matriz MxN e X um vetor Mx1, concluímos que também é possível escrever

$$P^o_{\sigma,X} = P^o_\sigma - P^o_\sigma X\,[X^T P^o_\sigma X]^{-1} X^T P^o_\sigma \tag{2.142}$$

Da definição (2.120) e de (2.140), temos

$$P^o_\sigma = S^T S \tag{2.143}$$

Assim

$$P^o_{\sigma,X} = S^T\,[I_M - SX(X^T S^T SX)^{-1} X^T S^T]\,S \tag{2.144}$$

Reconhecendo a expressão em colchetes como P^o_{SX}, temos

$$P^o_{\sigma,X} = S^T P^o_{SX}\,S \tag{2.145}$$

Finalmente, igualando (2.141) com (2.145), vem

$$P^o_X = S^T P^o_{SX}\,S + P^o_X\,\sigma[\sigma^T P^o_X\,\sigma]^{-1}\sigma^T P^o_X \tag{2.146}$$

Com o uso do vetor σ, os erros de filtragem a posteriori são obtidos através das relações

$$\varepsilon(n) = \sigma^T \varepsilon_M(n) \tag{2.147}$$

$$= \sigma^T P^o_X(n) D_M(n) \tag{2.148}$$

$$\varepsilon_a(n) = \sigma^T \varepsilon_{aM}(n) \tag{2.149}$$

$$= \sigma^T P^o_X(n-1) X_M(n) \tag{2.150}$$

$$\varepsilon_b(n) = \sigma^T \varepsilon_{bM}(n) \tag{2.151}$$

$$= \sigma^T P^o_X(n) X_M(n-N) \tag{2.152}$$

Queremos agora encontrar um operador, $Q^o_X(n)$, que permita obter os erros de filtragem a priori de uma forma análoga à usada acima para os erros a posteriori. Ou seja

$$e(n) = \sigma^T e_M(n) \tag{2.153}$$

$$= \sigma^T Q^o_X(n) D_M(n) \tag{2.154}$$

$$e_a(n) = \sigma^T e_{aM}(n) \tag{2.155}$$

$$= \sigma^T Q_X^o(n-1) X_M(n) \tag{2.156}$$

$$e_b(n) = \sigma^T e_{bM}(n) \tag{2.157}$$

$$= \sigma^T Q_X^o(n) X_M(n-N) \tag{2.158}$$

onde $e_M(n)$, $e_{aM}(n)$ e $e_{bM}(n)$ são vetores Mx1 cujos primeiros elementos são os erros a priori correspondentes. Lembrando que estes erros são dados pelas combinações lineares obtidas no instante imediatamente anterior, temos, para o caso do filtro adaptativo

$$e_M(n) = D_M(n) - X_{MN}(n) H(n-1) \tag{2.159}$$

onde

$$H(n-1) = [X_{MN}^T(n) S^T S X_{MN}(n)]^{-1} X_{MN}^T(n) S^T S D_M(n) \tag{2.160}$$

Assim

$$e_M(n) = \{I_M - X_{MN}(n) [X_{MN}^T(n) S^T S X_{MN}(n)]^{-1} X_{MN}^T(n) S^T S\} D_M(n) \tag{2.161}$$

Dessa forma, devemos definir $Q_X^o(n)$ como

$$Q_X^o(n) = I_M - X_{MN}(n) [X_{MN}^T(n) S^T S X_{MN}(n)]^{-1} X_{MN}^T(n) S^T S \tag{2.162}$$

Vamos observar que $Q_X^o(n)$ é um operador que projeta um vetor Mx1 obliquamente ao subespaço gerado por $X_{MN}(n)$. Os operadores $P_X^o(n)$ e $Q_X^o(n)$ guardam entre sí a relação

$$\sigma^T Q_X^o(n) = [\sigma^T P_X^o(n) \sigma]^{-1} \sigma^T P_X^o(n) \tag{2.163}$$

cuja demonstração é feita no Apêndice 2C.

Observando as equações (2.114) e (2.127), notamos que é conveniente definirmos o operador $G_X(n)$, de dimensão NxM, por

$$G_X(n) = [X_{MN}^T(n) X_{NM}(n)]^{-1} X_{MN}^T(n) \tag{2.164}$$

o qual fornece a combinação linear relativa à projeção de um dado vetor ao longo do subespaço gerado por $X_{MN}(n)$. Dessa forma, temos

$$H(n) = G_X(n) D_M(n) \tag{2.165}$$

$$A(n) = G_X(n-1) X_M(n) \tag{2.166}$$

e

$$B(n) = G_X(n)X_M(n-N) \tag{2.167}$$

O vetor de ganho, tão empregado no tratamento matricial que fizemos anteriormente, corresponde à primeira coluna de $G_X(n)$, pois

$$G_X(n)\,\sigma = [X^T_{MN}(n)X_{MN}(n)]^{-1}\,X^T_{MN}(n)\,\sigma \tag{2.168}$$

$$= R^{-1}(n)X(n) \tag{2.169}$$

$$= G(n) \tag{2.170}$$

A esta altura, já é possível delinear a forma pela qual a solução do problema deve evoluir no espaço vetorial. Suponhamos que estamos no instante n. Desejamos então obter H(n), e temos como conhecidos todos os parâmetros referentes ao instante anterior, os quais foram obtidos a partir de projeções no subespaço vetorial de $X_{MN}(n-1)$. Este procedimento deve ser feito em duas etapas. Primeiro é acrescentado o vetor $X_M(n)$ ao espaço de $X_{MN}(n-1)$. O subespaço de N+1 dimensões resultante se relaciona tanto com a solução obtida no instante n-1, quanto com a que será obtida no instante n; após a retirada do vetor $X_M(n-N)$, que constitui a segunda etapa. Seguindo este raciocínio, desejamos encontrar uma forma para estender a ordem do operador $G_X(n)$ para um subespaço de N+1 dimensões. Para isso, notemos primeiramente que

$$P_X = X\,G_X \tag{2.171}$$

$$= [X\ Z]\begin{bmatrix} G_X \\ 0^T \end{bmatrix} \tag{2.172}$$

e

$$P_{X,Z} = [X\ Z]\,G_{X,Z} \tag{2.173}$$

onde Z é um vetor Mx1 e, novamente, estamos omitindo os índices das matrizes. Usando a equação (2.132), temos

$$P_{X,Z} = [X\ Z]\begin{bmatrix} G_X \\ 0^T \end{bmatrix} + P^o_X\,Z\,[Z^T P^o_X\,Z]^{-1}\,Z^T P^o_X \tag{2.174}$$

Notando que

$$P^o_X = I - X\,G_X \tag{2.175}$$

vem

$$P_{X,Z} = [X\ Z] \begin{bmatrix} G_X \\ 0^T \end{bmatrix} + [X\ Z] \begin{bmatrix} -G_X Z \\ 1 \end{bmatrix} [Z^T P_X^o Z]^{-1} Z^T P_X^o \tag{2.176}$$

De (2.173) e (2.176), temos

$$G_{X,Z} = \begin{bmatrix} G_X \\ 0^T \end{bmatrix} + \begin{bmatrix} -G_X Z \\ 1 \end{bmatrix} [Z^T P_X^o Z]^{-1} Z^T P_X^o \tag{2.177}$$

Permutando X com Z no desenvolvimento acima, e procedendo da mesma forma, obtemos

$$G_{Z,X} = \begin{bmatrix} 0^T \\ G_X \end{bmatrix} + \begin{bmatrix} 1 \\ -G_X Z \end{bmatrix} [Z^T P_X^o Z]^{-1} Z^T P_X^o \tag{2.178}$$

Resta agora encontrar uma expressão que relacione $G_X(n)$ com $G_X(n-1)$. Substituindo (2.175) e (2.163) em (2.146), podemos escrever

$$XG_X = I - S^T [I - SXG_{SX}] S - [I - XG_X] \sigma\sigma^T Q_X^o \tag{2.179}$$

De (2.140), temos

$$XG_X = \sigma\sigma^T [I - Q_X^o] + S^T SXG_{SX}S + XG_X\sigma\sigma^T Q_X^o \tag{2.180}$$

$$= \sigma\sigma^T XG_{SX}S + S^T SXG_{SX}S + XG_X\sigma\sigma^T Q_X^o \tag{2.181}$$

Usando novamente (2.140), e pré-multiplicando ambos os membros da expressão resultante por $[X^T X]^{-1}X^T$, obtemos

$$G_X = G_{SX}S + G_X\sigma\sigma^T Q_X^o \tag{2.182}$$

Os operadores definidos acima, assim como as expressões obtidas para atualizá-los em ordem e no tempo, formam uma base para a construção dos vários algoritmos rápidos. Iremos aqui derivar somente o Algoritmo FK. Os demais podem ser derivados de forma semelhante. Comecemos por definir o vetor de ganho estendido

$$G_{N+1}(n) = G_{Z,X}(n)\,\sigma \tag{2.183}$$

onde $Z = X_M(n)$ e $X = X_{MN}(n-1)$. Usando (2.178) em (2.183), temos

$$G_{N+1}(n) = \begin{bmatrix} 0 \\ G(n-1) \end{bmatrix} + \begin{bmatrix} 1 \\ -G_X(n-1)X_M(n) \end{bmatrix} \frac{X_M^T(n)P_X^o(n-1)\sigma}{X_M^T(n)P_X^o(n-1)X_M(n)} \tag{2.184}$$

Lembrando que

$$E_a(n) = \| \varepsilon_{aM} \|^2 \tag{2.185}$$

$$= \varepsilon_{aM}^T(n)\varepsilon_{aM}(n) \tag{2.186}$$

$$= X_M^T(n)P_X^o(n-1)X_M(n) \tag{2.187}$$

temos

$$G_{N+1}(n) = \begin{bmatrix} 0 \\ G(n-1) \end{bmatrix} + \frac{\varepsilon_a(n)}{E_a(n)} \begin{bmatrix} 1 \\ -A(n) \end{bmatrix} \tag{2.188}$$

onde usamos também as equações (2.150) e (2.166). Se, ao invés de (2.183), fizermos

$$G_{N+1}(n) = G_{X,Z}(n)\sigma \tag{2.189}$$

com $Z = X_M(n\text{-}N)$ e $X = X_{MN}(n)$, obtemos

$$G_{N+1}(n) = \begin{bmatrix} G(n) \\ 0 \end{bmatrix} + \frac{\varepsilon_b(n)}{E_b(n)} \begin{bmatrix} -B(n) \\ 1 \end{bmatrix} \tag{2.190}$$

A atualização do preditor progressivo é feita aplicando (2.182) ao vetor $X_M(n)$ e usando (2.166), (2.170) e (2.156)

$$G_X(n\text{-}1)X_M(n) = G_{SX}(n\text{-}1)SX_M(n) + G_X(n\text{-}1)\sigma\sigma^T Q_X^o(n\text{-}1)X_M(n) \tag{2.191}$$

de onde vem

$$A(n) = A(n\text{-}1) + e_a(n)\, G(n\text{-}1) \tag{2.192}$$

A expressão para $\varepsilon_a(n)$ é obtida pré-multiplicando (2.125) por σ^T.

$$\sigma^T\varepsilon_{aM}(n) = \sigma^T [X_M(n) - X_{MN}(n)A(n)] \tag{2.193}$$

de onde vem

$$\varepsilon_a(n) = x(n) - X^T(n)A(n) \tag{2.194}$$

O mesmo é feito para $e_a(n)$

$$\sigma^T e_{aM}(n) = \sigma^T [X_M(n) - X_{MN}(n)A(n\text{-}1)] \tag{2.195}$$

$$e_a(n) = x(n) - X^T(n\text{-}1)A(n\text{-}1) \tag{2.196}$$

Para a atualização de $E_a(n)$, pré-multiplicamos (2.146) por $X_M^T(n)$, pós-multiplicamos por $X_M(n)$ e substituímos (2.163). Daí resulta

$$X_M^T(n)P_X^o(n-1)X_M(n) = X_M^T(n)S^TP_{SX}^o(n-1)SX_M(n) + X_M^T(n)P_X^o(n-1)\sigma\sigma^TQ_X^o(n-1)X_M(n) \quad (2.197)$$

Usando (2.187), (2.150) e (2.156), temos

$$E_a(n) = E_a(n-1) + \varepsilon_a(n-1)e_a(n) \quad (2.198)$$

As expressões para atualizar B(n), $e_b(n)$, $\varepsilon_b(n)$ e $E_b(n)$ são obtidas de forma análoga às correspondentes para o preditor progressivo. O procedimento de construção do Algoritmo FK, é exatamente o mesmo descrito na Seção 2.3, e não convém ser repetido.

O parâmetro $\gamma(n)$, que desempenha um papel fundamental no contexto dos algoritmos FTF e FAEST, tem uma interpretação geométrica que vale a pena ser discutida. Lembrando que $\gamma(n)$ pode ser entendido como sendo o valor do erro quando o processo de otimização tem como seqüência de referência o vetor σ, podemos definir o vetor $\gamma_M(n)$, de dimensão Mx1, por

$$\gamma_M(n) = P_X^o(n)\sigma \quad (2.199)$$

que corresponde a $\varepsilon_M(n)$ para o caso em que $D_M(n) = \sigma$. Assim temos

$$\gamma(n) = \sigma^TP_X^o(n)\sigma \quad (2.200)$$

Este fator integra a equação (2.163) e constitui a base para a derivação dos algoritmos FTF e FAEST [7]. Notando que (2.200) representa o produto escalar dos vetores, σ e $\gamma_M(n)$, temos que

$$\gamma(n) = \|\sigma\| \, \|\gamma_M(n)\| \cos\theta \quad (2.201)$$

$$= \|\gamma_M(n)\| \cos\theta \quad (2.202)$$

onde θ é o ângulo formado pelo vetor σ e um vetor normal ao subespaço gerado por $X_{MN}(n)$. Como $\gamma_M(n)$ é a projeção de σ normal a este subespaço, temos

$$\|\gamma_M(n)\| = \|\sigma\| \cos\theta \quad (2.203)$$

$$= \cos^2\theta \quad (2.204)$$

e de (2.202) e (2.204), temos finalmente

$$\gamma(n) = \cos^2\theta \quad (2.205)$$

Por esse motivo, $\gamma(n)$ é também referida como sendo uma variável angular. A equação (2.205) permite concluir novamente que o valor de $\gamma(n)$ varia no intervalo de 0 a 1.

2.8 – COMENTÁRIO

Neste capítulo derivamos os três principais algoritmos rápidos utilizando duas diferentes abordagens: a algébrica e a vetorial. Estes algoritmos solucionam o problema LS com uma complexidade proporcional a N, ou, ao número de coeficientes do filtro transversal. Esta solução representa uma evolução considerável em termos de esforço computacional, principalmente quando comparada com o Algoritmo RLS, cuja complexidade é proporcional a N^2.

A proposta dos algoritmos rápidos permite melhorar sensivelmente os compromissos existentes nos processos de filtragem adaptativa entre o número de coeficientes, a capacidade de rastreio do sistema e o esforço computacional exigido. Nesse contexto, o processamento de sinais com amostras correlacionadas é o maior beneficiário.

A despeito do avanço representado pelos algoritmos rápidos, deve ser mencionado que a implementação física destes algoritmos, ao contrário do Algoritmo RLS, apresenta sérios inconvenientes de instabilidade numérica, devido ao limite de precisão imposto pelos sistemas reais. No desenvolvimento deste capítulo foram assumidos os valores teóricos das variáveis, o que corresponde a uma precisão infinita. Na prática, infelizmente, isto é impossível de ser feito.

APÊNDICE 2A

Vamos aqui demonstrar a equação (2.44). Como vimos na Seção 2.3, a matriz de autocorrelação estendida é dada por

$$R_{N+1}(n) = \begin{bmatrix} \rho(n) & r_a^H(n) \\ r_a(n) & R(n-1) \end{bmatrix} \tag{2.206}$$

Fazendo

$$R_{N+1}^{-1}(n) = \begin{bmatrix} u & V^H \\ V & Z_N \end{bmatrix} \tag{2.207}$$

e usando (2.19), podemos escrever

$$\begin{bmatrix} E_a(n) + r_a^H(n)A(n) & r_a^H(n) \\ r_a(n) & R(n-1) \end{bmatrix} \begin{bmatrix} u & V^H \\ V & Z_N \end{bmatrix} = \begin{bmatrix} 1 & 0_N^T \\ 0_N & I_N \end{bmatrix} \tag{2.208}$$

Esta expressão pode também ser representada pelo sistema de equações

$$u\,[E_a(n) + r_a{}^H(n)A(n)] + r_a{}^H(n)V = 1 \tag{2.209a}$$

$$u\,r_a(n) + R(n\text{-}1)V = 0_N \tag{2.209b}$$

$$r_a(n)V^H + R(n\text{-}1)Z_N = I_N \tag{2.209c}$$

De (2.209b), temos

$$V = -u\,R^{-1}(n\text{-}1)r_a(n) \tag{2.210}$$

$$= -u\,A(n) \tag{2.211}$$

Substituindo (2.111) em (2.209a), vem

$$u = \frac{1}{E_a(n)} \tag{2.212}$$

e, de (2.209b)

$$V = -\frac{1}{E_a(n)}\,A(n) \tag{2.213}$$

Usando estes resultados em (2.109c), temos

$$Z_N = R^{-1}(n\text{-}1) - R^{-1}(n\text{-}1)r_a(n)V^H \tag{2.214}$$

$$= R^{-1}(n-1) + \frac{1}{E_a(n)}\,A(n)A^H(n) \tag{2.215}$$

A equação (2.207) pode então ser reescrita como

$$R_{N+1}^{-1}(n) = \begin{bmatrix} 0 & 0_N^T \\ 0_N & R^{-1}(n-1) \end{bmatrix} + \frac{1}{E_a(n)} \begin{bmatrix} 1 & -A^H(n) \\ -A(n) & A(n)A^H(n) \end{bmatrix} \tag{2.216}$$

Ou seja, o que queríamos demonstrar. Usando um procedimento semelhante, podemos demonstrar também a equação (2.45).

APÊNDICE 2B

Neste apêndice iremos apresentar uma prova mais rigorosa da equação (2.103). Como resultado intermediário neste processo, obteremos também outras relações interessantes sobre as energias dos erros de predição dos algoritmos rápidos.

Tomemos inicialmente a identidade matricial.

$$\begin{bmatrix} \rho(n) & r_a^H(n) \\ r_a(n) & R(n-1) \end{bmatrix} \begin{bmatrix} 1 & 0_N^T(n) \\ -A(n) & R^{-1}(n-1) \end{bmatrix} = \begin{bmatrix} E_a(n) & A^H(n) \\ 0_N(n) & I_N \end{bmatrix} \tag{2.217}$$

que foi construída de forma a apresentar linhas e/ou colunas com um único elemento não nulo. Usando a equação (2.206) e operações elementares da álgebra de matrizes e determinantes, é possível escrever

$$\det R_{N+1}(n)\ \det R^{-1}(n-1) = E_a(n) \tag{2.218}$$

ou ainda

$$E_a(n) = \frac{\det R_{N+1}(n)}{\det R(n-1)} \tag{2.219}$$

De forma análoga, usando a identidade matricial

$$\begin{bmatrix} R(n) & r_b(n) \\ r_b^H(n) & \rho(n-N) \end{bmatrix} \begin{bmatrix} R^{-1}(n) & -B(n) \\ 0_N^T & 1 \end{bmatrix} = \begin{bmatrix} I_N & 0_N \\ B^H(n) & E_b(n) \end{bmatrix} \tag{2.220}$$

temos

$$\det R_{N+1}(n)\ \det R^{-1}(n) = E_b(n) \tag{2.221}$$

ou

$$E_b(n) = \frac{\det R_{N+1}(n)}{\det R(n)} \tag{2.222}$$

Tomando a equação de recursividade para a matriz de autocorrelação

$$R(n) = w\,R(n-1) + X(n)X^H(n) \tag{2.223}$$

podemos escrever

$$I_N - X(n)G^H(n) = w\,R(n-1)R^{-1}(n) \tag{2.224}$$

Aplicando novamente as propriedades do determinante de uma matriz, temos

$$\det [I_N - X(n)G^H(n)] = w^N \det R(n-1) \det R^{-1}(n) \tag{2.225}$$

O determinante no primeiro membro pode ser obtido com o uso da relação matricial

$$\det [I_N + YZ] = \det [I_M + ZY] \tag{2.226}$$

onde Y, Z, I_N e I_M são matrizes de dimensões compatíveis. Usando esta relação e a equação (2.58) em (2.225), obtemos

$$\gamma(n) = w^N \frac{\det R(n-1)}{\det R(n)} \tag{2.227}$$

A partir das equações (2.219), (2.222) e (2.227), podemos finalmente escrever

$$\gamma(n)\frac{E_a(n)}{E_b(n)} = w^N \tag{2.228}$$

que corresponde à equação (2.103).

APÊNDICE 2C

Neste apêndice iremos provar a equação (2.163). Comecemos para isso tomando a equação (2.140)

$$I = S^T S + \sigma\sigma^T \tag{2.229}$$

De onde podemos escrever

$$X^T S^T S X = X^T X - X^T \sigma\sigma^T X \tag{2.230}$$

Aplicando o Lema de Inversão de Matrizes, equação (1.64), com

$$A = X^T S^T S X \tag{2.231a}$$

$$B = X^T X \tag{2.231b}$$

$$C = X^T \sigma \tag{2.231c}$$

$$D = -1 \tag{2.231d}$$

temos

$$[X^T S^T S X]^{-1} = [X^T X]^{-1} - [X^T X]^{-1} X^T \sigma \, [\sigma^T X (X^T X)^{-1} X^T \sigma - 1]^{-1} \, \sigma^T X \, [X^T X]^{-1} \tag{2.232}$$

Usando $\sigma^T\sigma = 1$ e a definição (2.120), temos

$$[X^T S^T S X]^{-1} = [X^T X]^{-1} - [X^T X]^{-1} X^T \sigma \, [\sigma^T P_X^o \, \sigma]^{-1} \, \sigma^T X \, [X^T X]^{-1} \tag{2.233}$$

Tomando agora a definição do operador Q_X^o

$$Q_X^o = I - X \, [X^T S^T S X]^{-1} X^T S^T S \tag{2.234}$$

Substituindo (2.233) e (2.119), temos

$$Q_X^o = I - P_X S^T S - P_X \sigma \, [\sigma^T P_X^o \, \sigma]^{-1} \, \sigma^T P_X \, S^T S \tag{2.235}$$

Pré-multiplicando ambos os lados por σ^T e notando que

$$\sigma^T P_X S^T = -\sigma^T P_X^o S^T \tag{2.236}$$

temos, após outras simplificações

$$\sigma^T Q_X^o = \sigma^T - \sigma^T P_X S^T S + [\sigma^T P_X^o \sigma]^{-1} \sigma^T P_X^o S^T S - \sigma^T P_X^o S^T S \tag{2.237}$$

$$= \sigma^T - \sigma^T S^T S + [\sigma^T P_X^o \sigma]^{-1} \sigma^T P_X^o S^T S \tag{2.238}$$

Como $\sigma^T S^T = 0$, vem

$$\sigma^T Q_X^o = \sigma^T + [\sigma^T P_X^o \sigma]^{-1} \sigma^T P_X^o S^T S \tag{2.239}$$

Usando novamente (2.140), temos finalmente

$$\sigma^T Q_X^o = [\sigma^T P_X^o \sigma]^{-1} \sigma^T P_X^o \tag{2.240}$$

que corresponde à expressão (2.163), enunciada na Seção 2.7.

2.9 – REFERÊNCIAS

[1] Haykin, S., Adaptive Filter Theory, Prentice-Hall, 1986.

[2] Ljung, L., Morf, M. and Falconer, D., "Fast Calculation of Gain Matrices for Recursive Estimation Schemes", Int. J. Control, vol. 27, pp 1-19, 1978.

[3] Cioffi, J. M. and Kailath, T., "Fast, Recursive Least-Squares Transversal Filters for Adaptive Filtering", IEEE Trans. ASSP, vol. 32, pp. 304-337, April 1984.

[4] Carayannis, G., Manolakis, D. and Kalouptsidis, N., "A Fast Sequential Algorithm for Least-Squares Filtering and Prediction", IEEE Trans. ASSP, vol. 31, pp. 1394-1402, December 1983.

[5] Samson, C., "A Unified Treatment of Fast Algorithms for identification", Int. J. Control, vol. 35, pp. 909-934, 1982.

[6] Alexander, S.T., Adaptive Signal Processing: Theory and Aplications, Springer-Verlag, New York, 1986.

[7] Wang, J. D., "Unified Derivation and Initial Convergence of Three Prewindowed Fast Transversal RLS Algorithms", IEEE Trans. ASSP, vol. 36, pp. 1091-1096, July 1988.

CAPÍTULO 3

ANÁLISE DE ESTABILIDADE DOS ALGORITMOS LS

Não é possível conjecturar a aplicação dos algoritmos de mínimos quadrados na prática sem um cuidadoso entendimento de seus comportamentos com relação à estabilidade numérica. Para tanto, desenvolvemos neste e no próximo capítulo, um estudo que busca o entendimento do processo de instabilidade, assim como a proposição de soluções para este tipo de problema.

Neste capítulo analisaremos os mecanismos causadores da instabilidade, adiando para o Capítulo 4 a proposição de soluções. Nossa ênfase estará dirigida não apenas a entender e solucionar o problema da instabilidade, mas também a proporcionar uma visão geral do assunto, preenchendo assim uma lacuna existente na literatura.

Os fatores que provocam a instabilidade, ou a divergência, dos algoritmos de mínimos quadrados são sabidamente os erros de arredondamento (ou truncamento) conseqüentes da representação em precisão finita. Contudo, outros fatores contribuem severamente para a aceleração do processo de divergência. A escolha do fator de esquecimento e a característica de estacionaridade do sinal de entrada, são parâmetros às vezes tão influentes quanto o número de bits usado na representação das variáveis.

Para melhor entender os problemas de instabilidade foi desenvolvido um programa de computador que simula os algoritmos anteriormente descritos nas mais diversas condições. Dessa forma, as observações e os resultados aqui apresentados puderam ser confirmados experimentalmente.

O comportamento dos algoritmos de mínimos quadrados em uma implementação prática envolve tipicamente três fases: a primeira delas é a fase de convergência, onde as variáveis evoluem dos valores iniciais até os valores de regime. A duração desta fase é breve (algumas dezenas de iterações) e praticamente isenta de ruído de quantização, devido ao pouco tempo que

este teve para se acumular. A segunda fase, quando o algoritmo encontra-se em regime, é caracterizada por uma flutuação das variáveis em torno dos valores esperados (para o caso de sinais de entrada estacionários). Em condições hipotéticas de precisão infinita, tendo como entrada um sinal bem comportado, esta fase deveria durar indefinidamente. Entretanto, o ruído de quantização acumulado produz alterações nas variáveis. Estas alterações ao longo do tempo se tornam significativas até que, em dado instante, observa-se uma divergência abrupta nos valores das variáveis escalares e vetoriais. Esta divergência é uma particularidade dos algoritmos rápidos que pode se manifestar também no Algoritmo RLS se alguns cuidados especiais não forem tomados na sua implementação. Discutiremos este caso mais adiante. A terceira fase, que se segue após a divergência, é caracterizada por uma total instabilidade, com as variáveis atingindo valores aleatórios e extremos, sem qualquer tendência de retornar às condições anteriores, ou de regime. Esse processo de total descontrole pode também determinar uma ocorrência de overflow; o que representa uma situação inaceitável, sobretudo quando se processa sinais em tempo real.

A ocorrência de overflow, a instabilidade das variáveis, e outros inconvenientes, como divisão por zero, são eventos que devem ser evitados a qualquer custo, mesmo tendo que se prescindir da aplicação rigorosa do critério LS. Ou seja, pequenas alterações no algoritmo são toleráveis, desde que eliminem o problema da instabilidade. Para todos os efeitos práticos, é preferível um algoritmo cuja saída não esteja rigorosamente no contexto do critério LS, mas que seja numericamente estável, já que em um algoritmo não estável tão somente as primeiras saídas podem ser aproveitadas. Neste sentido, muitos artifícios foram propostos, visando principalmente a estabilização dos algoritmos rápidos. Alguns deles, que consideramos mais importantes, serão discutidos no início do próximo capítulo.

Em nossa experiência, identificamos três fatores que, conjunta ou separadamente, provocam a divergência dos algoritmos LS:

1 - A falta de excitação persistente

2 - O fator de esquecimento

3 - O ruído de quantização

Procuraremos estudar separadamente cada um destes fatores. O primeiro está ligado às características do sinal de entrada e é comum ao Algoritmo RLS e aos algoritmos rápidos. O fator de esquecimento tem uma influência mais marcante na estabilidade dos algoritmos rápidos. Já o ruído de quantização, ou ruído de arredondamento, assume características específicas em cada particular implementação, exigindo com isso um tratamento diferenciado para cada um dos algoritmos.

3.1 – A FALTA DE EXCITAÇÃO PERSISTENTE

O primeiro dos três fatores de instabilidade está ligado às características de estacionaridade do sinal de entrada. Para melhor entendê-lo, consideraremos inicialmente um processo de identificação de sistemas, onde um filtro transversal de comprimento N é excitado por um sinal

perfeitamente predizível, como por exemplo, uma somatória de K senóides complexas e ortogonais

$$x(n) = \sum_{k=1}^{K} A_k e^{j\omega_k n} \tag{3.1}$$

Este sinal determinístico tem sua representação espectral, $X(e^{j\omega})$, caracterizada pela existência de raias nas freqüências $e^{j\omega_k}$. O esforço do filtro transversal consiste em igualar sua saída, y(n), ao sinal d(n) proveniente do sistema que se deseja identificar. Assumindo a condição de regime, podemos dizer que a representação espectral do sinal na saída do filtro é dada por

$$Y(e^{j\omega}) = H(e^{j\omega})\, X(e^{j\omega}) \tag{3.2}$$

onde

$$H(e^{j\omega}) = \sum_{i=0}^{N-1} h_i e^{-j\omega i} \tag{3.3}$$

é a resposta em freqüência do filtro. Dessa forma, temos que a característica espectral de $Y(e^{j\omega})$ é também constituída unicamente por raias em $e^{j\omega_k}$, k = 1,2, . . , K. Nestas freqüências, os coeficientes do filtro transversal devem assumir valores tais que

$$A_k [h_0 + h_1 e^{-j\omega_k} + . . + h_{N-1} e^{-j(N-1)\omega_k}] = D(e^{j\omega_k}) \qquad k = 1, 2, . . , K \tag{3.4}$$

onde $D(e^{j\omega})$ é a resposta em freqüência do sinal presente na saída do sistema que se deseja identificar.

O conjunto de equações (3.4) é perfeitamente atendido quando K = N, ou seja, quando o número de senóides complexas do sinal se iguala à ordem do filtro. Neste caso, os coeficientes do filtro convergem para os valores determinados pela solução simultânea das equações e o erro de estimação tende a zero. Para o caso em que K > N, estas equações não podem ser atendidas simultaneamente. O erro de estimação não se anula, podendo ser minimizado pelo critério LS. O vetor de coeficientes de regime é obtido no domínio do tempo através da equação (1.22).

Para o caso em que K < N, o sistema de equações (3.4) é atendido com N – K graus de liberdade, o que significa uma subdeterminação nos valores dos coeficientes do filtro. As conseqüências dessa subdeterminação podem ser analisadas heuristicamente no domínio da freqüência. O filtro transversal, nas K posições correspondentes às raias do sinal de entrada, apresenta ganhos determinados pelas componentes do sinal de referência. Entretanto, nas regiões de freqüência nula do espectro de {x(n)}, o ganho pode variar extremamente, fazendo com que pequenos ruídos (de arredondamento por exemplo) sejam amplificados ao nível de produzirem a instabilidade.

O problema gerado por um sinal do tipo descrito pela equação (3.1) é denominado *falta de excitação persistente*. Ele também pode ser entendido no domínio do tempo notando-se que

a matriz de autocorrelação do processo x(n) é singular para o caso em que $K < N$ [1], o que inviabiliza a aplicação da equação (1.22).

Uma definição matematicamente precisa da condição de "excitação persistente" pode ser encontrada em [2] e [3]. Preferimos no entanto uma definição equivalente, assumindo que um sinal $\{x(n)\}$ apresenta excitação persistente quando sua matriz de autocorrelação determinística, R(n), for positiva definida.

Um meio simples de solucionar o problema da falta de excitação persistente consiste em adicionar um ruído branco de pouca amplitude ao sinal de entrada, eliminando-se assim as regiões nulas do espectro. Este procedimento equivale a acrescentar valores positivos aos elementos da diagonal principal de R(n), tornando a matriz melhor condicionada. Os sinais encontrados em aplicações práticas são, de forma geral, bem comportados e prescindem deste artifício.

A instabilidade ocasionada pela falta de excitação persistente não tem relação com o ruído de arredondamento, pois a matriz de autocorrelação do sinal de entrada tenderia à singularidade ainda que as variáveis tivessem precisão infinita. Uma medida do "bom comportamento" do sinal de entrada pode ser dada pelo fator de condicionamento da matriz de autocorrelação, introduzido na Seção 1.2. Este parâmetro determina o quanto a matriz está próxima ou distante da singularidade. Da Propriedade 1.2.4 temos que a matriz de autocorrelação torna-se singular na medida em que deixa de ser positiva definida. Neste caso, o fator de condicionamento tende ao infinito. No outro extremo, quando este fator é 1, as amostras do sinal de entrada são totalmente descorrelacionadas e o espectro de potência é plano (ruído branco).

Na procura de seqüências de entrada que favoreçam a instabilidade, concluímos que processos de entrada do tipo AR (autoregressivo), com pólos próximos ao círculo de raio unitário, são modelos representativos de sinais problemáticos para os algoritmos rápidos. Faremos a seguir uma breve revisão deste tipo de sinal.

O processo AR

Na Figura 3.1 ilustramos o esquema de geração de um sinal do tipo AR de ordem K. Este sinal obedece a relação

$$x(n) + \sum_{k=1}^{K} c_k\, x(n-k) = v(n) \tag{3.5}$$

onde os c_k são coeficientes do processo AR e v(n) é um processo estacionário com amostras descorrelacionadas e variância σ_v^2. A Transformada Z do gerador AR é dada por

$$\frac{X(z)}{V(z)} = \frac{1}{1 + \sum_{k=1}^{K} c_k\, z^{-k}} \tag{3.6}$$

ou ainda, ressaltando a existência dos pólos

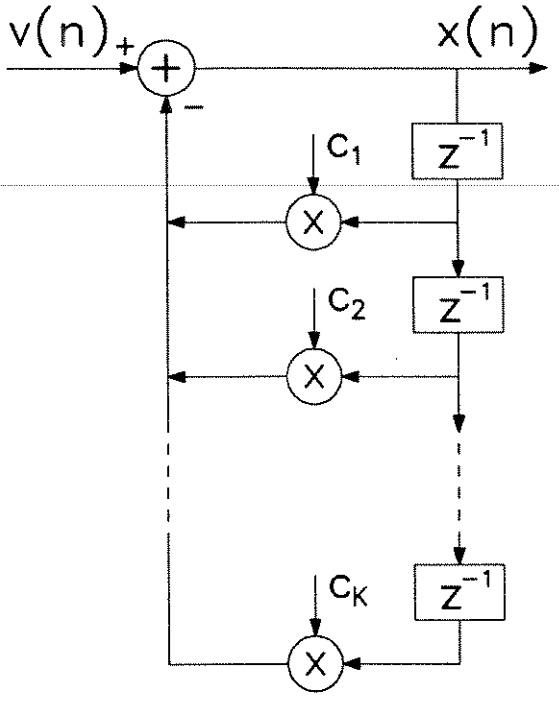


Figura 3.1 - Esquema de geração de um processo AR

$$\frac{X(z)}{V(z)} = \frac{1}{\prod_{k=1}^{K} [1 - p_k z^{-1}]} \tag{3.7}$$

onde os p_k representam os pólos no plano complexo. Como pode ser observado, o número de pólos é igual a ordem do processo AR. Estes pólos devem localizar-se no interior do círculo de raio unitário para garantir a estacionaridade (ou a estabilidade) do processo. De forma geral, a síntese de um sinal AR deve partir da alocação dos pólos no círculo de raio unitário. No exemplo abaixo ilustramos este procedimento para um processo AR de ordem 4.

Exemplo 3.1

Um sinal AR de ordem 4 deve apresentar 4 pólos. Impondo que estes pólos sejam conjugados dois a dois, obtemos um sinal real. Assim

$$\frac{X(z)}{V(z)} = \frac{1}{[1 - r_1 e^{j\theta_1}] [1 - r_1 e^{-j\theta_1}] [1 - r_2 e^{j\theta_2}] [1 - r_2 e^{-j\theta_2}]} \tag{3.8}$$

onde r_i e θ_i são respectivamente, o módulo e o argumento de p_i. A equação (3.8) pode ser reescrita como

$$\frac{V(z)}{X(z)} = 1 - 2\,[r_1 \cos\theta_1 + r_2 \cos\theta_2]\, z^{-1} + [4\, r_1 \cos\theta_1\, r_2 \cos\theta_2 + r_1^2 + r_2^2]\, z^{-2}$$
$$- 2\,[r_1^2\, r_2 \cos\theta_2 + r_2^2\, r_1 \cos\theta_1]\, z^{-3} + r_1^2\, r_2^2\, z^{-4} \qquad (3.9)$$

Comparando (3.9) com (3.6), fica claro que os coeficientes AR, neste caso particular, são dados por

$$c_1 = -2[r_1 \cos\theta_1 + r_2 \cos\theta_2] \qquad (3.10a)$$

$$c_2 = 4\, r_1 \cos\theta_1\, r_2 \cos\theta_2 + r_1^2 + r_2^2 \qquad (3.10b)$$

$$c_3 = -2\,[r_1^2\, r_2 \cos\theta_2 + r_2^2\, r_1 \cos\theta_1] \qquad (3.10c)$$

$$c_4 = r_1^2\, r_2^2 \qquad (3.10d)$$

Resta agora determinar a variância do ruído utilizado como entrada do gerador AR. Tomemos para isso os coeficientes de autocorrelação de $\{x(n)\}$, os quais são definidos por

$$r(i) = E[x(n)x^*(n-i)] \qquad (3.11)$$

Estes coeficientes podem ser obtidos, no processo AR, multiplicando-se ambos os membros de (3.5) por $x^*(n-i)$, $i = 0, \ldots, K$ e tomando-se a esperança matemática do produto. Efetuando estas operações, obtemos

$$r(0) + \sum_{k=1}^{K} c_k\, r(k) = \sigma_v^2 \qquad i = 0 \qquad (3.12a)$$

$$r(i) + \sum_{k=1}^{K} c_k\, r(i-k) = 0 \qquad i = 1, \ldots, K \qquad (3.12b)$$

Usando a propriedade $r(-i) = r^*(i)$, que decorre diretamente da definição (3.11), o conjunto de K+1 equações simultâneas obtido acima pode ser resolvido para as variáveis r(i) em função dos coeficientes c_k e de σ_v^2. Na prática, entretanto, é conveniente fixar r(0), que corresponde à variância de $\{x(n)\}$, e deixar que σ_v^2 seja obtido simultaneamente com os r(i), $i = 1, \ldots, K$.

A partir dos coeficientes r(i) é possível construir a matriz de autocorrelação estocástica do processo AR e obter seu fator de condicionamento. Nas simulações que fizemos dos vários algoritmos, utilizamos como sinais de entrada processos estacionários do tipo AR com fatores de condicionamento que variavam extremamente. Como norma de trabalho, fixamos em 1 a variância dos sinais empregados; ou seja, fizemos r(0) = 1.

Como vimos, os parâmetros do processo AR podem ser obtidos a partir da localização dos pólos no plano complexo. A medida em que os pólos caminham para a origem do plano, os coeficientes AR vão diminuindo em valor absoluto e o processo AR vai tendendo a um ruído

branco (o ruído que é utilizado na geração). Este fato pode ser constatado no caso particular do Exemplo 3.1, observando-se os resultados fornecidos pelas equações (3.10a-d). Por outro lado, quando os pólos se afastam da origem, o processo começa a assumir um caráter oscilatório, e a variância do ruído gerador diminui (para garantir a amplitude constante do sinal AR produzido). No limite, quando os pólos tendem ao círculo de raio unitário, o ruído tende a zero, e o processo AR tende a uma somatória de senóides complexas, do tipo representado pela equação (3.1).

Concluímos então que os sinais do tipo ruído branco e somatória de senóides podem ser considerados casos extremos do processos AR. De um lado temos um sinal sem nenhuma redundância e, do outro, um sinal predizível.

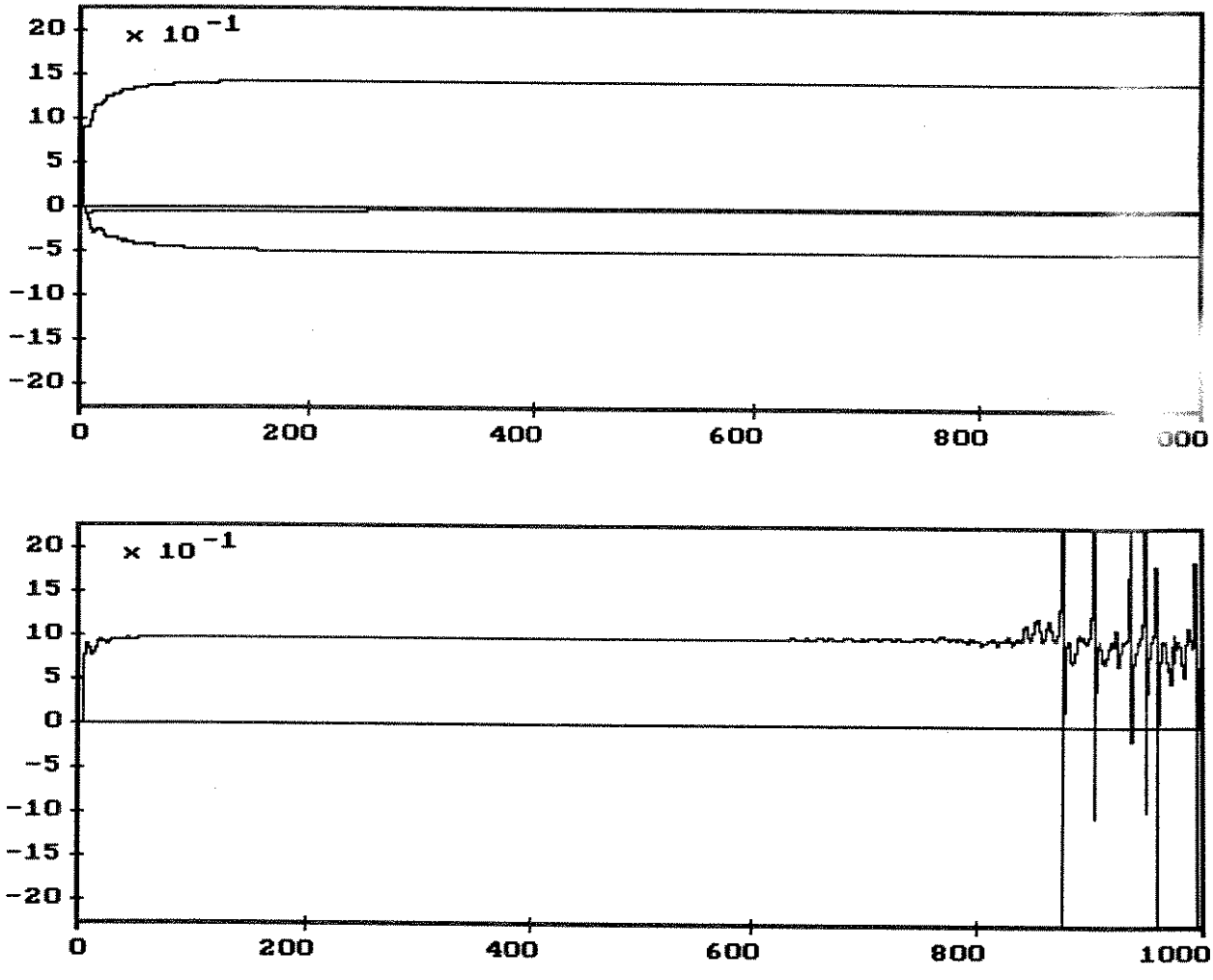


Figura 3.2 - Algoritmo RLS com N = 3 e w = 0,985, excitado por $x(n) = \sqrt{2}\cos(n\pi/12)$. Em **a** observa-se o comportamento dos coeficientes do preditor progressivo, A(n); e em **b**, o valor da variável $[w + X^H(n)P(n-1)X(n)]^{-1}$.

3.1.1 – Falta de excitação persistente no Algoritmo RLS

Diversas simulações foram feitas com o Algoritmo RLS. O algoritmo demonstrou total robustez numérica para uma vasta gama de sinais estacionários. No caso dos sinais compostos pela soma de senóides complexas, foi possível constatar que o algoritmo apresenta problemas em todos os casos em que $K < N$, ou seja, em que o número de exponenciais complexas presentes no sinal é menor que a ordem do filtro transversal. Ainda assim, os coeficientes do filtro convergem e apresentam saídas razoáveis durante todo o período em que o algoritmo se mantém em funcionamento (algumas centenas de iterações). O motivo da interrupção é sempre um overflow ou uma divisão por zero. Este problema, como já analisamos, é originado pela singularidade da matriz inversa.

A parte **a** da Figura 3.2 representa os elementos do preditor progressivo em uma execução do Algoritmo RLS quando excitado por uma senóide de potência unitária. O número de coeficientes usado foi $N = 3$, o que ocasionou overflow com aproximadamente 2000 iterações. A parte **b** desta mesma figura mostra a variável escalar $[w + X^H(n)P(n-1)X(n)]^{-1}$. Nota-se que apesar da aparente normalidade do preditor, esta variável denuncia irregularidades a partir da iteração 767, quando passou a assumir valores fora de sua faixa de validade. Tomando-se as equações (2.72), (2.68) e (2.58), pode-se mostrar que

$$\frac{1}{w + X^H(n)P(n-1)X(n)} = \frac{\gamma(n)}{w} \tag{3.13}$$

Usando (2.61), vemos que a faixa de validade desta variável situa-se entre 0 e 1/w. A ocorrência de um valor fora desta faixa indica que a matriz de autocorrelação perdeu a definição positiva, pois implica que $X^H(n)P(n-1)X(n) < 0$.

3.1.2 – Falta de excitação persistente nos algoritmos rápidos

Nos algoritmos rápidos, o problema da falta de excitação persistente parece menos influente que o gerado pelos erros de quantização. Isso porque estes algoritmos sempre divergem, quaisquer que sejam as características do sinal de entrada. Ainda assim, nos processos AR, o fenômeno da divergência é antecipado em cerca de três vezes na medida em que os pólos caminham do centro para a extremidade do círculo de raio unitário.

No caso dos algoritmos rápidos, as conseqüências de um sinal de entrada perfeitamente predizível ao nível de N coeficientes se manifestam de uma forma diferente do que no Algoritmo RLS. Quando o algoritmo é inicializado, os erros de predição progressiva a priori, e a posteriori, assumem valores altos, como é esperado, devido ao desarranjo inicial dos coeficientes do preditor progressivo. A medida em que estes coeficientes convergem, os erros de predição tendem a se anular, porque o preditor consegue estimar com exatidão as amostra do sinal de entrada. Se o fator de esquecimento for menor que 1, a relação (2.25) mostra que a energia do erro de predição progressiva, $E_a(n)$, também tende a zero. Ocorre que em todos os tipos de algoritmos rápidos existe um passo onde este parâmetro surge no denominador. É certo, portanto, que em uma execução prolongada destes algoritmos ocorrerá uma divisão por zero.

Em [4], Bellanger sugere uma "constante de estabilização", que a cada iteração é adicionada ao valor corrente de $E_a(n)$ para evitar a anulação desta variável. Por ter um valor muito pequeno, esta constante não deveria afetar significativamente o desempenho dos algoritmos rápidos. No caso de sinais bem comportados, entretanto, observamos que ela promove alterações que invariavelmente antecipam o processo de divergência. Com isso, é aconselhável que o emprego desta constante fique restrito ao uso de sinais de entrada específicos, que podem provocar o anulamento de $E_a(n)$.

As Figuras 3.3 e 3.4 ilustram respectivamente a evolução dos coeficientes de predição progressiva para o caso em que K < N nos algoritmos FK e FTF. Em ambas as figuras, o sinal x(n) utilizado é uma senóide real de potência unitária (K = 2) e o número de coeficientes é N = 3.

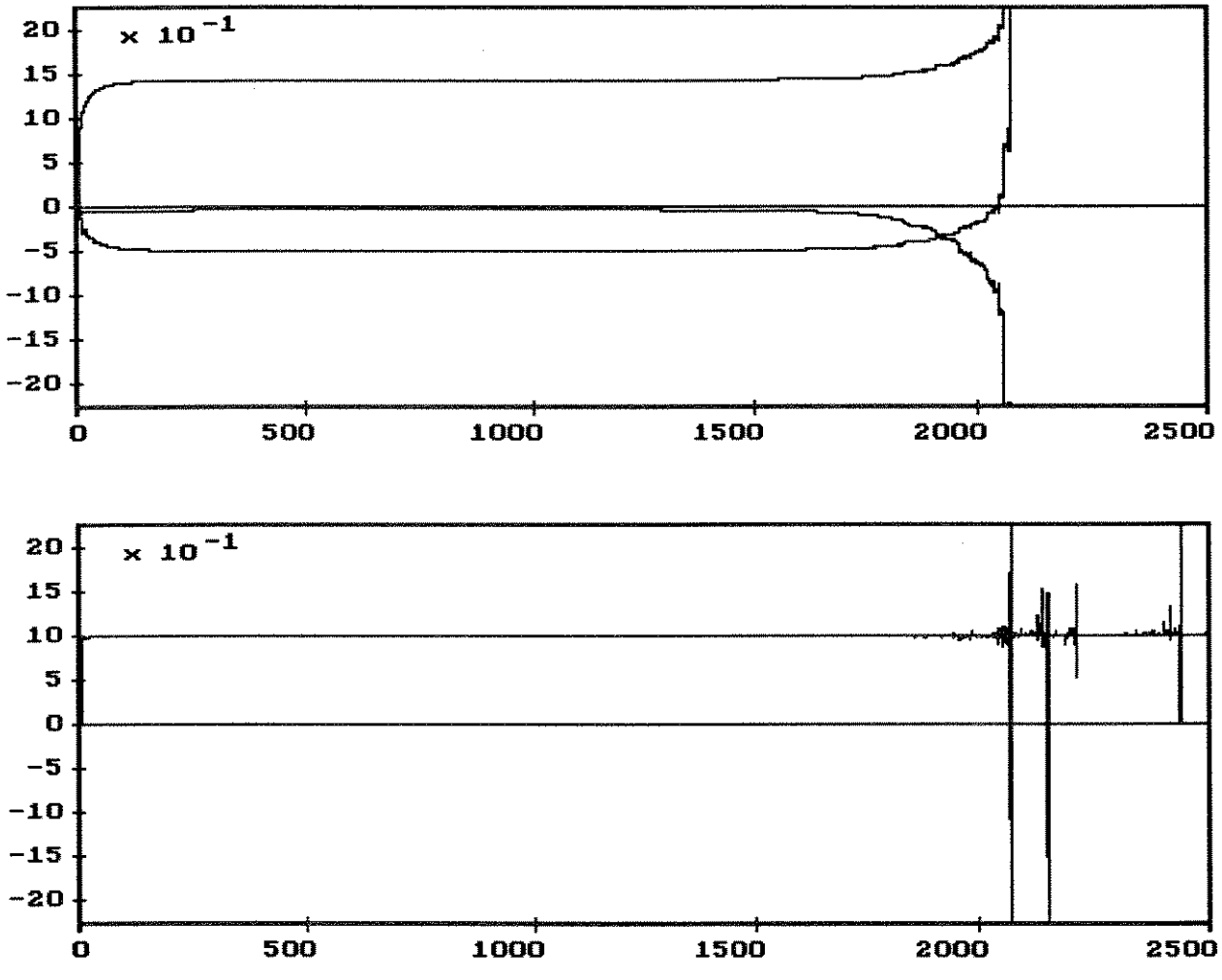


Figura 3.3 - Execução do Algoritmo FK com N = 3 e w = 0,985, excitado pelo sinal determinístico $x(n) = \sqrt{2}\cos(n\pi/12)$. Em **a**, observa-se o comportamento dos coeficientes do preditor progressivo, A(n); e em **b**, o valor da variável $1 - m(n)e_b(n)$.

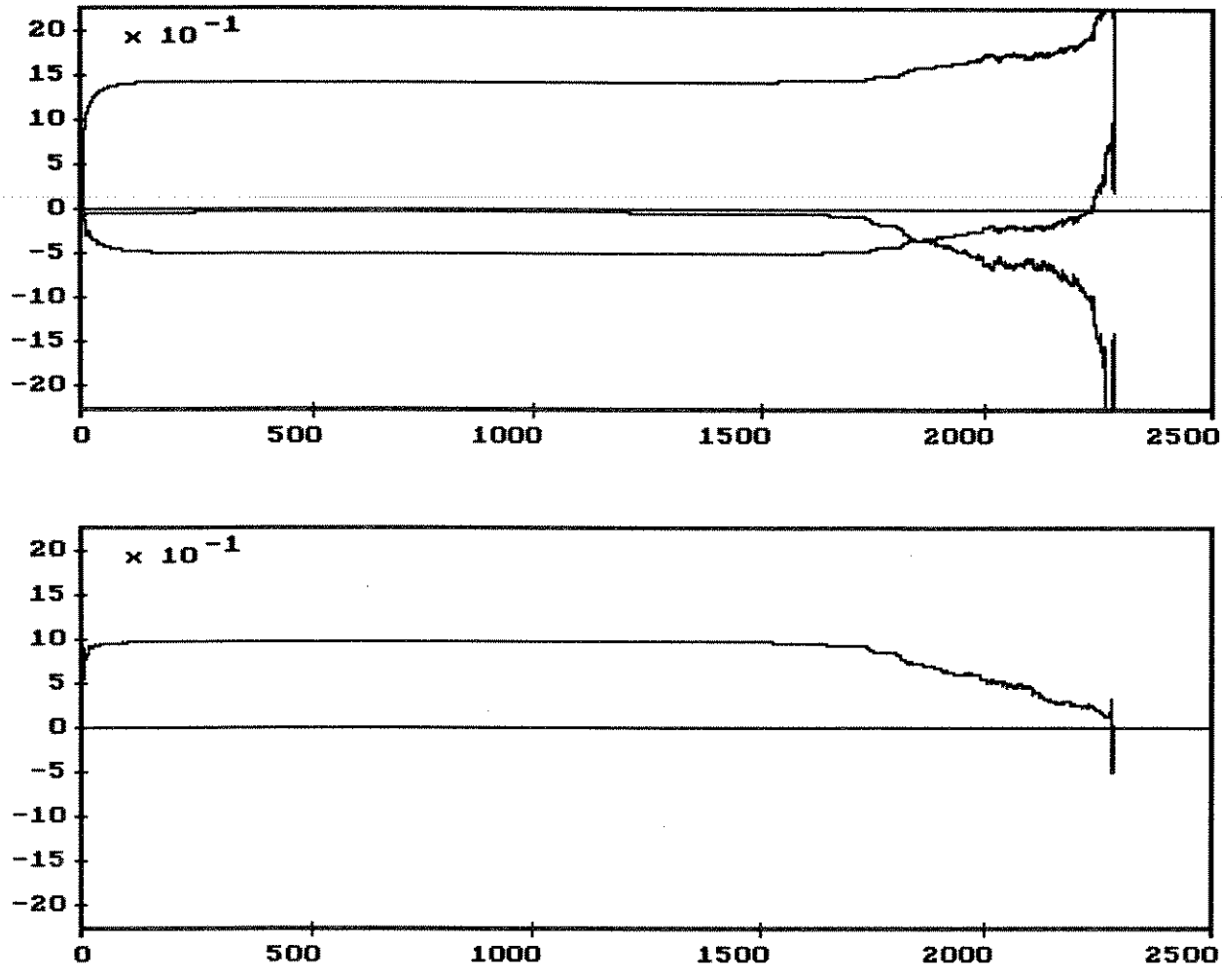


Figura 3.4 - Execução do Algoritmo FTF com N = 3 e w = 0,985, excitado pelo sinal determinístico $x(n) = \sqrt{2}\cos(n\pi/12)$. Em **a**, observa-se o comportamento dos coeficientes do preditor progressivo, A(n); e em **b**, o valor da variável $\gamma(n)$.

As variáveis escalares $1 - m(n)e_b(n)$ e $\gamma(n)$ também estão representadas nestas duas figuras. Estas variáveis são normalmente usadas para antecipar a ocorrência da divergência provocada pelos erros de quantização. Observa-se que elas não se prestam para prever a instabilidade gerada pela falta de excitação persistente, pois os elementos dos preditores divergem antes que elas apresentem um valor fora de sua faixa de validade.

É interessante notar que para o caso particular de sinais predizíveis (somatória de senóides) com K = N, o Algoritmo RLS exibe um funcionamento normal, enquanto que os algoritmos rápidos apresentam risco de overflow e divisão por zero. O entendimento deste processo reside na diferença estrutural existente entre os dois tipos de algoritmo. Enquanto no Algoritmo RLS, o vetor G(n) é obtido em cada iteração usando-se apenas a matriz P(n-1) e o vetor X(n), nos

algoritmos rápidos ele é obtido através de atualizações que envolvem os preditores e os erros de predição. A equação

$$A(n) = A(n-1) + e_a^*(n)\, G(n-1) \tag{3.14}$$

assim como a correspondente equação que atualiza o preditor regressivo mostra que, após haver convergido, os termos $e_a^*(n)G(n-1)$ e $e_b^*(n)G(n)$ devem assumir valores muito pequenos, ou mesmo nulos. Como os erros de predição também tendem a se anular, os elementos do vetor G(n) [ou $\bar{G}(n)$ nos algoritmos mais eficientes] podem assumir uma gama muito ampla de valores sem que ocorram alterações nos elementos dos preditores. Isto faz com que os elementos de G(n) comecem a crescer continuamente até que se dê a ocorrência de overflow.

Resumindo a questão da falta de excitação persistente, temos que ela se manifesta no Algoritmo RLS através da singularidade da matriz inversa, enquanto que nos algoritmos rápidos ela se deve à anulação dos erros de predição. Este fato fica bem patente quando analisamos o caso de sinais de entrada do tipo somatória de senóides com K = N, que funcionam perfeitamente no Algoritmo RLS, enquanto que ocasionam overflow e/ou divisão por zero nos algoritmos rápidos.

Vamos agora considerar o segundo fator de instabilidade nos algoritmos rápidos.

3.2 – O FATOR DE ESQUECIMENTO

Inicialmente, é necessário considerar a existência de dois tipos de divergência, ou instabilidade, nos algoritmos rápidos. O primeiro está associado aos casos descritos na seção anterior, e se caracteriza pela ocorrência de overflow (ou divisão por zero) ocasionada pelo desvanecimento no valor de certas variáveis. O segundo tipo se caracteriza por uma divergência abrupta dos preditores, sem que uma explicação tão óbvia esteja disponível. Apenas algumas variáveis [como $\gamma(n)$ no Algoritmo FTF] denunciam a iminência do fenômeno, quando seus valores extrapolam a faixa de validade, evidenciando uma anomalia. De forma geral, a falta de excitação persistente é responsável pela divergência do primeiro tipo, enquanto que os problemas causados pelo fator de esquecimento e pelos erros de quantização, são responsáveis pela divergência do segundo tipo.

As simulações mostram que a estabilidade dos algoritmos rápidos decai consideravelmente com a diminuição do fator de esquecimento, o que torna o uso desses algoritmos proibitivo para valores baixos de w. Tipicamente, um valor próximo de 0,97 pode causar a divergência de um algoritmo rápido em poucos milhares de iterações. O mesmo processo, executado com valores acima de 0,999, pode suportar milhões de iterações sem que o fenômeno da divergência se manifeste.

O Algoritmo RLS, ao contrário dos algoritmos rápidos, mostra-se insensível à escolha de w, chegando a manter-se estável mesmo para valores tão baixos quanto 0,5. A faixa de valores interessantes, para aplicações práticas, situa-se em geral acima de 0,99, quando a janela de tempo que influencia o processo de predição corresponde a pelo menos uma centena de amostras.

Assim como a falta de excitação persistente, a instabilidade gerada pelo fator de esquecimento também pode ser entendida através da matriz de autocorrelação. Os algoritmos rápidos, apesar de não utilizarem explicitamente esta matriz (ou sua inversa), mantém a cada instante uma certa correspondência com ela, pois foram derivados com base em sua estrutura. Dessa forma, é possível avaliar a influência do fator de esquecimento no processo de instabilidade dos algoritmos rápidos através do comportamento dos elementos de R(n).

A análise subseqüente admite que a seqüência de entrada {x(n)} seja um ruído branco de média nula, e que o cômputo das variáveis seja realizado com precisão infinita. Com estas restrições é possível isolar o aspecto que queremos observar (fator de esquecimento) das demais fontes de instabilidade (erros de quantização e falta de excitação persistente). Nesse sentido, os resultados dessa análise representam uma espécie de limitantes inferiores no processo de instabilidade, pois correspondem ao efeito produzido por somente uma das fontes.

Consideremos então a matriz de autocorrelação determinística de um ruído branco gaussiano, de média nula e variância σ^2. Da Propriedade 1.2.1 temos que, em condição de regime, esta matriz tende para

$$E[R(n)] = \frac{\sigma^2}{1-w} I_N \tag{3.15}$$

ou seja, os elementos r_{ij} da matriz assumem em média apenas dois valores: a constante $\sigma^2/(1-w)$ na diagonal principal e zero nas demais posições. Uma descrição mais completa destes elementos exige a análise de outros parâmetros estatísticos, que não apenas a média. No Apêndice 3A é mostrado que as distribuições de probabilidade destes elementos são também gaussianas e com variâncias

$$\sigma^2_{r_{ij}} = \frac{2}{1-w^2} \sigma^4 \qquad i = j \tag{3.16a}$$

$$\sigma^2_{r_{ij}} = \frac{1}{1-w^2} \sigma^4 \qquad i \neq j \tag{3.16b}$$

Usando a aproximação

$$1 - w^2 \approx 2\,[1-w] \tag{3.17}$$

que se aplica a valores de w próximos de 1, podemos reescrever os resultados em (3.16a) e (3.16b) como

$$\sigma_{r_{ij}} = \frac{1}{\sqrt{1-w}} \sigma^2 \qquad i = j \tag{3.18a}$$

$$\sigma_{r_{ij}} = \frac{1}{\sqrt{2(1-w)}} \sigma^2 \qquad i \neq j \tag{3.18b}$$

onde $\sigma_{r_{ii}}$ e $\sigma_{r_{ij}}$ são os desvios padrões dos elementos que estão respectivamente na diagonal principal de R(n) e fora dela.

Relembrando um conceito básico da distribuição gaussiana, temos que o desvio padrão é uma medida do espalhamento da variável em torno de sua média. Ou, mais precisamente: *a probabilidade de a variável aleatória assumir um valor no intervalo compreendido entre um desvio padrão abaixo e acima da média é aproximadamente 0,68*. Vamos utilizar esse conceito para avaliar o comportamento dos elementos da matriz de autocorrelação determinística. Considerando que a matriz dada pelos valores médios em (3.15) corresponde à matriz identidade multiplicada pela constante $\sigma^2/(1\text{-}w)$, teríamos um fator de condicionamento igual a 1. Entretanto, devido aos desvios padrões dos elementos, não se pode afirmar a priori que essa matriz seja bem condicionada. De fato, à medida em que aumenta o espalhamento causado pelo desvio padrão no valor dos elementos, diminui o condicionamento da matriz. É difícil quantificar exatamente essa degradação no condicionamento da matriz, mas é possível mensurar a amplitude do espalhamento em termos dos desvios padrões dos elementos. Para isso, vamos definir o parâmetro

$$\Sigma_{r_{ij}} = \frac{\sigma_{r_{ij}}}{E[r_{ii}]} \tag{3.19}$$

que representa a razão entre o desvio padrão dos elementos e a constante $\sigma^2/(1\text{-}w)$; uma espécie de desvio padrão normalizado. Este parâmetro pode assumir dois valores, conforme corresponda a elementos da diagonal principal ou não. De (3.18a), (3.18b) e (3.15), temos

$$\Sigma_{r_{ij}} = \begin{cases} \sqrt{1-w} & \text{se } i = j \\ \sqrt{\dfrac{1-w}{2}} & \text{se } i \neq j \end{cases} \tag{3.20}$$

Como pode ser observado, $\Sigma_{r_{ij}}$ depende fortemente do fator de esquecimento. A Figura 3.5 mostra um gráfico que descreve o comportamento deste parâmetro em função de w. Notemos que nessa figura ambos os eixos ordenados estão em escala logarítmica, o que ressalta a enorme influência do fator de esquecimento no condicionamento da matriz.

A influência do fator de esquecimento no condicionamento de R(n) também pode ser entendida notando-se que a janela de observação, ou número de amostras com ponderação significativa na função custo, diminui substancialmente com a redução de w, o que empobrece as médias fornecidas pelos elementos da matriz.

Do exposto acima, vemos que um filtro transversal, mesmo quando excitado por um sinal ideal (do ponto de vista da excitação persistente), deve apresentar problemas de instabilidade se o fator de esquecimento não for suficientemente próximo de 1. Isso porque a matriz de autocorrelação determinística torna-se mal condicionada. Assumindo precisão infinita, a matriz de autocorrelação mantém-se positiva definida, não obstante o mal condicionamento. Contudo, em realizações práticas, onde a precisão é limitada, o mal condicionamento redunda em

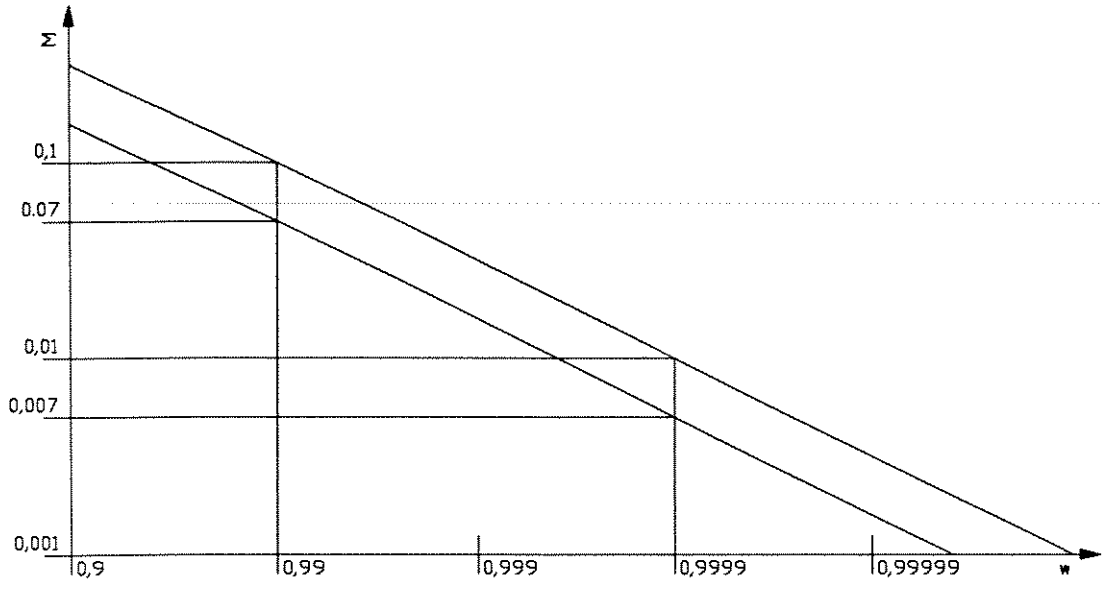


Figura 3.5 - Desvio padrão normalizado para os elementos da matriz de autocorrelação determinística, em função do fator de esquecimento.

dificuldades numéricas na atualização das variáveis. Os fatores que causam a instabilidade são, dessa forma, inter-relacionados. O fator de esquecimento provoca o mal condicionamento da matriz de autocorrelacão. Este por sua vez, determina a ocorrência de operações problemáticas, como por exemplo divisão por números pequenos. Este quadro é naturalmente agravado pelo limite de precisão da máquina: o terceiro fator de instabilidade.

A Figura 3.6 ilustra o comportamento dos elementos do preditor progressivo, para três valores de w, em diferentes simulações de um mesmo processo. Em todos os casos foi utilizado o Algoritmo FTF com N = 4, excitado pela mesma seqüência de entrada: um processo AR de quarta ordem. Sendo o número de elementos do preditor igual ao número de coeficientes AR, os valores para os quais os elementos do preditor devem convergir são trivialmente conhecidos. A variância do sinal de entrada, x(n), foi ajustada para 1, e a precisão na mantissa das variáveis em ponto flutuante foi de 23 bits. Pode-se observar a grande influência do fator de esquecimento na estabilidade do algoritmo através do número de iterações suportado em cada caso. Estes números são representativos de uma série de simulações realizadas, onde se variou também outros parâmetros menos influentes, como a ordem do filtro e o valor dos coeficientes AR. A Figura 3.7 ilustra a variável de verossimilhança, $\gamma(n)$, para as mesmas simulações da Figura 3.6.

Cabe uma última observação sobre o efeito do fator de esquecimento no Algoritmo RLS, uma vez que neste algoritmo a instabilidade não é observada. Como já dissemos, os fatores de instabilidade não são independentes. Veremos adiante que o Algoritmo RLS possui características numéricas muito robustas, o que garante sua estabilidade para um leque muito amplo de condições.

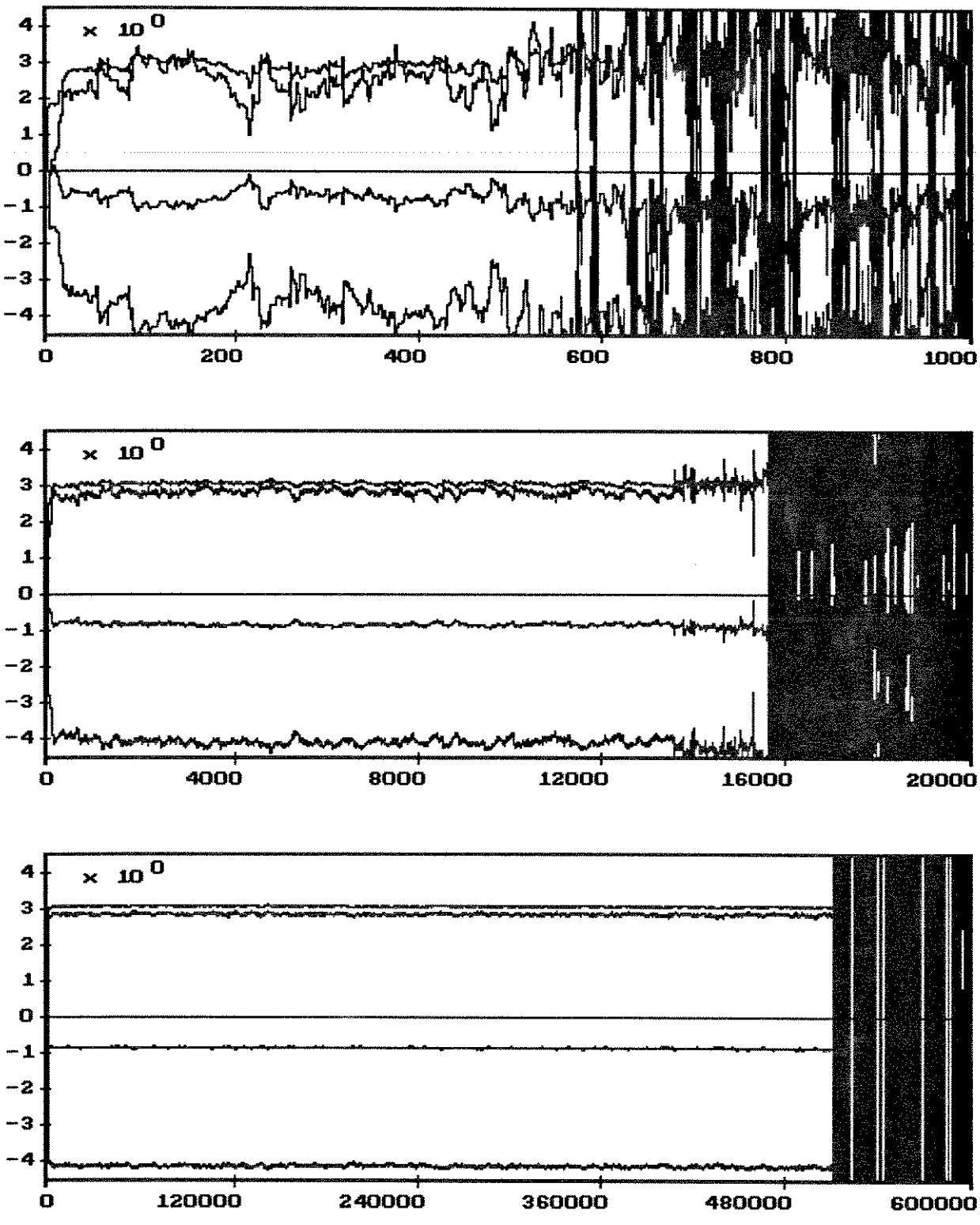


Figura 3.6 - Coeficientes de predição progressiva para 3 diferentes valores de w no Algoritmo FTF implementado com N = 4. Em **a**, w = 0,9; em **b**, w = 0,99; e em **c**, w = 0,999. Nos três casos foi utilizado como sinal de entrada um processo AR com variância 1 e coeficientes $c_1 = 3{,}08$, $c_2 = -4{,}13$, $c_3 = 2{,}86$ e $c_4 = -0{,}85$.

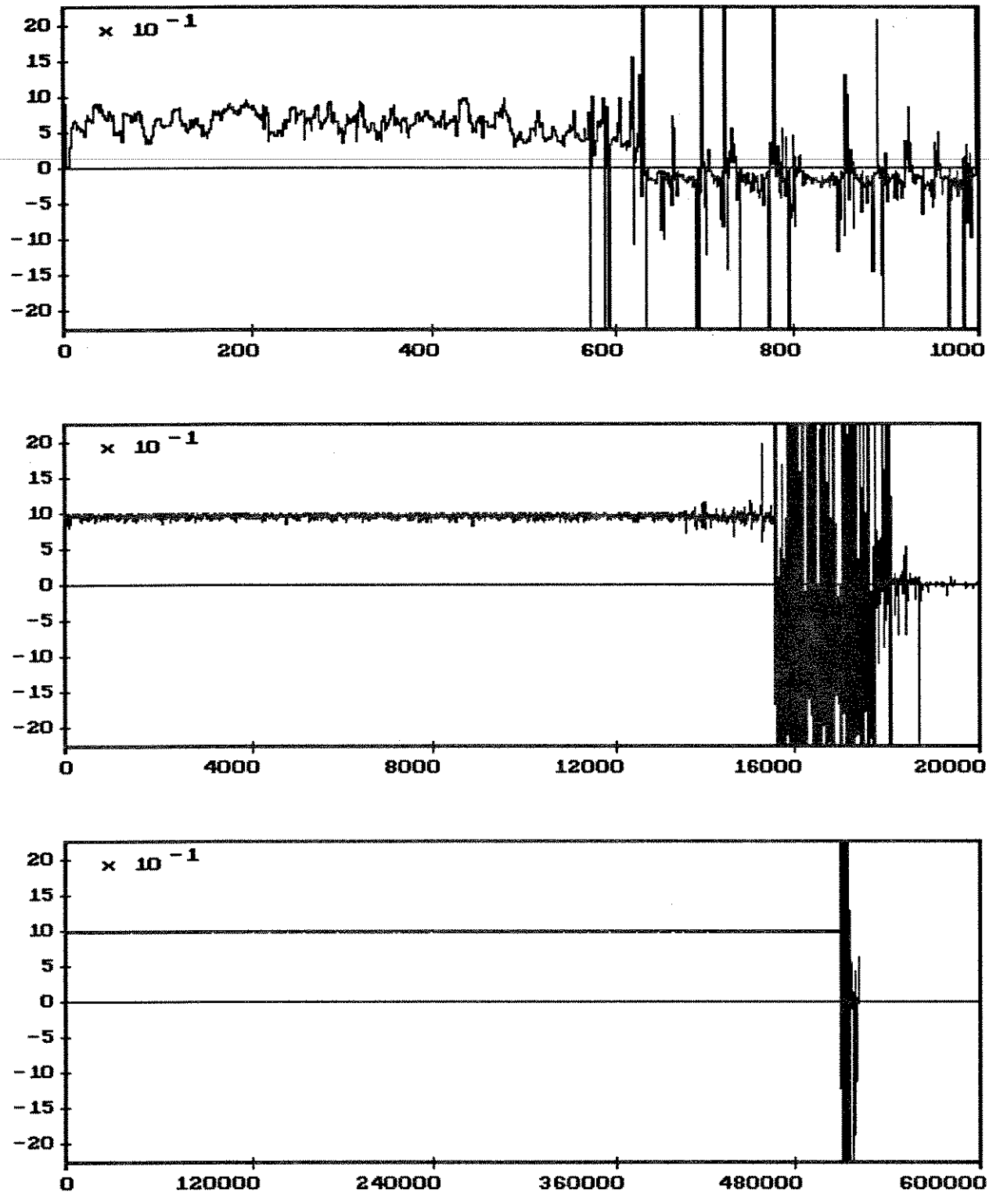


Figura 3.7 - Variável de verossimilhança, $\gamma(n)$, para as mesmas simulações da Figura 3.6. Esta variável antecipa a ocorrência da divergência. Em **a** este parâmetro extrapola a faixa de valores permitida com 572 iterações; em **b**, com 13.406; e em **c**, com 378.019 iterações.

3.3 – O RUÍDO DE QUANTIZAÇÃO

O limite de precisão da máquina, último dos três fatores de instabilidade, é com certeza o mais importante. De fato, sem ele a instabilidade devida ao fator de esquecimento não seria observada. Infelizmente, a influência do ruído de quantização nos algoritmos LS é difícil de ser analisada, sobretudo porque ela se manifesta de forma muito específica em cada particular implementação. O tratamento que faremos utiliza conceitos estatísticos, além de aproximações matemáticas bastante simples. Os resultados, no entanto, condizem com as simulações.

Antes porém, é oportuno fazer algumas considerações sobre a representação de números reais em sistemas computacionais.

Modelamento dos Erros de Quantização

Em uma implementação computacional, as variáveis podem ser representadas de duas formas: em ponto fixo e em ponto flutuante. Discutiremos brevemente ambas as formas, recomendando para uma discussão mais detalhada a referência [5].

No caso da representação em ponto fixo, as variáveis assumem valores discretos, cujo passo de quantização é dado pelo valor do bit menos significativo. Assim, para o caso de uma variável implementada com b bits depois da vírgula, temos um passo de quantização de 2^{-b}. O efeito de uma operação de multiplicação por exemplo, gera casas que se estendem além do bit menos significativo, implicando na necessidade de uma quantização. Essa quantização pode ser efetuada na forma de um truncamento, onde os bits excedentes são simplesmente ignorados, ou na forma de um arredondamento, onde o valor discreto mais próximo é adotado. Para esse último caso, o erro de quantização α gerado é tal que

$$-\frac{1}{2}2^{-b} < \alpha \leq \frac{1}{2}2^{-b} \tag{3.21}$$

Assim, a quantização por arredondamento de uma variável v em ponto fixo pode ser modelada como

$$\bar{v} = v + \alpha \tag{3.22}$$

onde $\bar{v}$ é o resultado da quantização e α é uma variável aleatória com distribuição uniforme no intervalo definido em (3.21). A média de α é nula e sua variância é dada por

$$\sigma_\alpha^2 = \frac{1}{12}2^{-2b} \tag{3.23}$$

Para o caso de uma implementação em ponto flutuante, o modelamento do processo de quantização é diferente. Uma variável em ponto flutuante pode ser escrita como

$$v = m\,2^c \tag{3.24}$$

onde m representa a mantissa, que é um número situado entre 0 e 1, e c representa a característica da variável em ponto flutuante. A quantização é realizada somente na mantissa, mas seu efeito global é o que realmente importa, de forma que é mais interessante encarar o erro em ponto flutuante como sendo um erro multiplicativo, do que como um erro meramente aditivo. Assim, para o processo de quantização de uma variável v em ponto flutuante, temos

$$\bar{v} = v(1 + \beta) \tag{3.25}$$

onde β é a variável aleatória associada ao processo ruidoso e $\bar{v}$ é a variável quantizada. Supondo que a quantização seja feita por arredondamento, o erro adicionado à mantissa se localiza no intervalo $[-2^{-b}/2, 2^{-b}/2]$, onde b é o número de bits utilizado na representação da mantissa. Assim

$$-2^c \frac{2^{-b}}{2} < \bar{v} - v \leq 2^c \frac{2^{-b}}{2} \tag{3.26}$$

Usando (3.25), temos

$$-2^c \frac{2^{-b}}{2} < v\beta \leq 2^c \frac{2^{-b}}{2} \tag{3.27}$$

e como

$$2^{c-1} \leq v < 2^c \tag{3.28}$$

podemos escrever

$$-2^{-b} < \beta \leq 2^{-b} \tag{3.29}$$

A densidade de probabilidade de β não é uniforme. Em [6] é mostrado que para b maior que algumas unidades, a variável β tem média nula e variância

$$\sigma_\beta^2 = 0{,}18\ 2^{-2b} \tag{3.30}$$

A equação (3.27) sugere que o erro de quantização na variável em ponto flutuante v seja definido como

$$\alpha = \beta v \tag{3.31}$$

Assumindo que v seja descorrelacionada de β, temos para a variância de α

$$\sigma_\alpha^2 = 0{,}18\ E[v^2]\ 2^{-2b} \tag{3.32}$$

Um ponto a ser ressaltado, é que o ruído provocado pela quantização nas variáveis representadas em ponto flutuante se apresenta tanto nos resultados das operações de multiplica-

ção como de adição. No caso da representação em ponto fixo, ao contrário, unicamente as multiplicações são afetadas.

As suposições feitas acima são bastante razoáveis e condizem com muitas situações práticas. No caso dos algoritmos rápidos, a aplicação deste modelo é extremamente dificultada pela grande quantidade de termos que se acumulam ao longo de uma iteração. Procuraremos, na medida do possível, usar uma abordagem mais genérica, que sirva tanto ao caso de ponto fixo quanto ao caso de ponto flutuante.

Vamos discutir separadamente o processo de acumulação de ruído nos vários algoritmos LS. Começaremos, no entanto, considerando uma questão que envolve tanto os algoritmos rápidos quanto o RLS. Esta questão se refere à atualização do filtro transversal adaptativo, que corresponde aos últimos passos encontrados nos vários algoritmos descritos.

3.3.1 – Análise de estabilidade do processo adjunto

Tomemos inicialmente o Algoritmo RLS, apresentado na Tabela 1.2. Observando rapidamente a referida tabela, vemos que este algoritmo pode ser entendido como sendo dois processos que ocorrem simultaneamente: o primeiro consiste da atualização da matriz inversa, P(n), e o segundo representa a adaptação do vetor de coeficientes, H(n), a qual é feita através do erro de estimação a priori, e(n), e do vetor de ganho G(n). Notadamente, o segundo processo é dependente do primeiro, pois utiliza o vetor G(n) que é obtido como um resultado intermediário do cômputo de P(n). O primeiro processo, ao contrário, é independente do segundo, uma vez que P(n) pode ser atualizada sem o uso de H(n) ou e(n). Com isso, queremos observar que o Algoritmo RLS pode, para efeitos de análise, prescindir do segundo processo, que a partir de agora passaremos a chamar de "processo adjunto". Mostraremos primeiramente que a malha de recursão formada pelo processo adjunto, igualmente encontrada nos algoritmos rápidos, é numericamente estável. Seguiremos para isso os passos traçados no trabalho pioneiro de Ljung [2].

A equação que efetua a atualização de H(n) é dada por

$$H(n) = H(n-1) + G(n)\, e^*(n) \tag{3.33}$$

$$= H(n-1) + G(n)\, [d^*(n) - X^H(n)H(n-1)] \tag{3.34}$$

$$= [I_N - G(n)X^H(n)]\, H(n-1) + G(n)\, d^*(n) \tag{3.35}$$

Usando (1.62) e (1.25), o termo incluso no colchetes pode ser reescrito como

$$[I_N - G(n)X^H(N)] = w\, P(n)R(n\text{-}1) \tag{3.36}$$

onde $P(n) = R^{-1}(n)$. Substituindo este resultado acima, vem

$$H(n) = w\, P(n)R(n\text{-}1)H(n\text{-}1) + G(n)\, d^*(n) \tag{3.37}$$

Dessa forma, se um vetor de erro $\delta H(n_o)$ é adicionado ao vetor H(n) no instante n_o, temos

$$\Delta H(n) = w^{n-n_o} P(n)R(n_o)\,\delta H(n_o) \qquad n > n_o \qquad (3.38)$$

onde $\Delta H(n)$ é o erro resultante no vetor H(n) em virtude da adição de $\delta H(n_o)$. Considerando $\{x(n)\}$ como sendo um processo estacionário e $w < 1$ temos, na condição de regime

$$E[P(n)R(n_o)] = I_N \qquad (3.39)$$

e, portanto, o erro adicionado em $H(n_o)$ tende a decair exponencialmente. Para o caso em que $w = 1$, o termo $E[P(n)R(n_o)]$ resulta em uma matriz cujos elementos decaem com 1/n. Dessa forma, podemos concluir que a malha formada pelo processo adjunto é estável.

A análise feita acima não permite afirmar que o Algoritmo RLS seja estável, como feito em [2], mas apenas que uma das malhas de recursão é estável. A outra malha, que atualiza a matriz P(n) no Algoritmo RLS, ou que atualiza o vetor G(n) nos algoritmos rápidos, é a que realmente importa na análise de estabilidade pois, como vimos, prescinde do processo adjunto. É nela que o ruído de quantização pode se acumular. Assim sendo, para os propósitos deste capítulo, o Algoritmo RLS fica reduzido simplesmente à expressão

$$P(n) = \frac{1}{w}\left[P(n-1) - \frac{1}{w + X^H(n)P(n-1)X(n)} P(n-1)X(n)X^H(n)P(n-1)\right] \qquad (3.40)$$

que representa a recursividade para a atualização da inversa da matriz de autocorrelação. Pela mesma razão, as análises de estabilidade que faremos dos algoritmos rápidos também dispensam os passos correspondentes ao processo adjunto.

Começaremos a análise de estabilidade pelo Algoritmo RLS. Como dissemos anteriormente, este algoritmo pode apresentar sérios problemas de instabilidade caso seu uso não leve em conta certos fatores de implementação. Vamos agora discutir com detalhes estes pormenores.

3.3.2 – Análise de estabilidade do Algoritmo RLS

O Algoritmo RLS é consagrado por sua robustez numérica. Contudo, sua implementação computacional exige cuidados na preservação da simetria da matriz inversa, P(n). Caso contrário, o algoritmo torna-se extremamente instável, apresentando um desempenho muito inferior aos algoritmos rápidos.

Mostraremos a seguir como a adição de uma matriz de erros, δP, com elementos de valores bem pequenos, pode afetar o comportamento do algoritmo nos instantes seguintes. Antes porém, é conveniente definir as matrizes

$$P_e = \frac{1}{2}\delta P + \frac{1}{2}\delta P^H \qquad (3.41a)$$

e

$$P_o = \frac{1}{2}\delta P - \frac{1}{2}\delta P^H \tag{3.41b}$$

que correspondem respectivamente às componentes Hermitiana e Anti-Hermitiana de δP. Estabelecemos estas definições por analogia a um tratamento encontrado na área de análise espectral de sinais, onde um dado sinal pode ser decomposto em suas partes pares e ímpares [7]. Com estas definições, a matriz de erro δP pode ser escrita como

$$\delta P = P_e + P_o \tag{3.42}$$

É importante notar que as matrizes P_e e P_o possuem as propriedades

$$P_e = P_e^H \tag{3.43a}$$

e

$$P_o = -P_o^H \tag{3.43b}$$

Como pode ser observado, qualquer matriz pode ser decomposta na forma da equação (3.43). As propriedades da componente Hermitiana foram bem estudadas no Capítulo 1. A componente Antiermitiana contudo, merece ainda alguma atenção. Para isso, vamos enunciar o seguinte lema

Lema 3.1

Se P_o *é uma matriz Anti-Hermitiana e* V *é um vetor não nulo qualquer, então o resultado do produto* $V^H P_o V$ *é um escalar imaginário puro*

Usando (3.41b) temos

$$V^H P_o V = V^H [\frac{1}{2}\delta P - \frac{1}{2}\delta P^H] V \tag{3.44}$$

$$= \frac{1}{2} V^H \delta P V - \frac{1}{2} V^H \delta P^H V \tag{3.45}$$

$$= \frac{1}{2} V^H \delta P V - \frac{1}{2} [V^H \delta P V]^* \tag{3.46}$$

$$= Im\,[V^H \delta P V] \tag{3.47}$$

De onde concluímos o enunciado do lema.

Retomando o raciocínio anterior, temos

$$\bar{P}(n-1) = P(n-1) + \delta P \tag{3.48}$$

Utilizando essa matriz assim alterada, ao invés de P(n-1), na recursão (3.40), obtemos

$$\bar{P}(n) = \frac{1}{w}\{P(n-1) + \delta P - \frac{1}{w + X^H(n)\,[P(n-1) + \delta P]\,X(n)} [P(n-1) + \delta P]\,X(n)X^H(n)\,[P(n-1) + \delta P^H]\} \tag{3.49}$$

Notando que a relação $X^H(n)\delta PX(n)/[w + X^H(n)P(n-1)X(n)]$ representa um escalar de valor bem pequeno, é valida a aproximação

$$\frac{1}{w + X^H(n)[P(n-1)+\delta P]X(n)} \approx \frac{1}{w + X^H(n)P(n-1)X(n)}\left[1 - \frac{X^H(n)\delta PX(n)}{w + X^H(n)P(n-1)X(n)}\right] \tag{3.50}$$

que foi obtida a partir da relação

$$\frac{1}{1+x} \approx 1 - x \qquad |x| << 1 \tag{3.51}$$

Usando (3.42) e considerando o resultado do Lema 3.1, é possível separar a parte real da parte imaginária no segundo membro de (3.50). Uma implementação acertada do Algoritmo RLS deve ignorar a parte imaginária, relativa ao produto $X^H(n)P_oX(n)$, pois ela se deve unicamente ao erro adicionado. Substituindo (3.50) em (3.49), com a parte imaginária devidamente ignorada, usando (3.40) e desprezando os erros de segunda ordem, temos

$$\bar{P}(n) = P(n) + \frac{1}{w}[\delta P - G(n)X^H(n)\delta P^H - \delta PX(n)G^H(n) + G(n)X^H(n)P_eX(n)G^H(n)] \tag{3.52}$$

onde usamos também a equação (1.67), que define o vetor ganho de adaptação

$$G(n) = \frac{1}{w + X^H(n)P(n-1)X(n)}\,P(n-1)X(n) \tag{3.53}$$

Substituindo (3.42) em (3.52), e usando as propriedades (3.43a) e (3.43b), temos

$$\bar{P}(n) = P(n) + \frac{1}{w}[I_N - G(n)X^H(n)]\,P_e\,[I_N - X(n)G^H(n)] + \frac{1}{w}P_o + \frac{1}{w}[G(n)X^H(n)P_o + P_o^HX(n)G^H(n)] \tag{3.54}$$

Usando a equação (3.36), obtemos finalmente

$$\bar{P}(n) = P(n) + w\,P(n)R(n-1)P_eR(n-1)P(n) + \frac{1}{w}P_o + \frac{1}{w}[G(n)X^H(n)P_o + P_o^HX(n)G^H(n)] \quad (3.55)$$

Esta equação permite uma boa compreensão do processo ruidoso, sobretudo com relação às componentes Hermitiana e Anti-Hermitiana de δP, pois seus efeitos comparecem separados. Pode-se notar que P_e produz uma perturbação que decai exponencialmente com o tempo, enquanto que o efeito produzido por P_o cresce sem limites. Supondo que a matriz de erro ΔP seja puramente Hermitiana ($P_o = 0_N$), e seja adicionada em $P(n_o)$, temos que a perturbação observada em $n > n_o$ é dada por

$$\Delta P(n) = w^{n-n_o}\,P(n)R(n_o)P_eR(n_o)P(n)] \quad (3.56)$$

Recorrendo novamente à equação (3.39), temos que, para $\{x(n)\}$ estacionário, a perturbação resultante de uma matriz de erro Hermitiana tende a desvanecer com o tempo. Além disso, enquanto seus elementos decaem, a matriz de perturbação se mantém Hermitiana, não perdendo suas características de simetria. Portanto, P(n) também se mantém rigorosamente Hermitiana.

Por outro lado, quando a matriz de erro δP possui uma parte Anti-Hermitiana ($P_o \neq 0_N$), a matriz P(n) é perturbada não apenas por uma componente Anti-Hermitiana, que cresce com 1/w, mas também por uma componente Hermitiana, correspondente ao último termo do segundo membro de (3.55). A primeira perturbação, como mostra o Lema 3.1, não altera a definição positiva da matriz P(n), mas a segunda, que evolui de uma forma acoplada com a primeira, é a grande responsável pela alteração da definição positiva da matriz P(n).

O artifício necessário para manter o Algoritmo RLS estável consiste em se garantir que a matriz P(n) permaneça simétrica ao longo do tempo. Na Tabela 3.1 apresentamos como exemplo duas implementações do Algoritmo RLS. A primeira delas, aparentemente igual a segunda, ocasiona uma assimetria em P(n). Para entender como isso ocorre, lembremos que o produto numérico de duas variáveis resulta em um valor cuja representação exige uma quantização. Denotando por Q[x] o valor quantizado do produto x, temos que, de forma geral, em uma implementação computacional

$$Q[k(n)U(n)]U^H(n) \neq k(n)Q[U(n)U^H(n)] \quad (3.57)$$

Dessa forma, a matriz P(n) resultante do primeiro algoritmo possui uma pequena assimetria, enquanto que a do segundo se mantém rigorosamente simétrica. O crescimento exponencial do ruído mostra que a pequena assimetria ocasionada na primeira implementação é suficiente para torná-la extremamente instável, o que não ocorre com a segunda. De fato, pudemos verificar que para o caso de w = 1, onde o crescimento do erro não é exponencial, até mesmo a primeira implementação da Tabela 3.1 apresenta um comportamento aceitável.

A segunda implementação foi testada usando representação em ponto flutuante com 23 bits na mantissa. Para uma grande variedade de sinais estacionários utilizados como entrada e

Tabela 3.1 - Duas possíveis implementações do Algoritmo RLS

a - Implementação computacionalmente incorreta

$$U(n) = P(n-1)X(n)$$

$$k(n) = \frac{1}{w + X^H(n)U(n)}$$

$$G(n) = k(n)U(n)$$

$$P(n) = \frac{1}{w}[P(n-1) - G(n)U^H(n)]$$

b - Implementação computacionalmente correta

$$U(n) = P(n-1)X(n)$$

$$k(n) = \frac{1}{w + X^H(n)U(n)}$$

$$G(n) = k(n)U(n)$$

$$P(n) = \frac{1}{w}[P(n-1) - k(n)U(n)U^H(n)]$$

uma ampla faixa de valores de w, esta implementação não demonstrou qualquer tendência de instabilidade. Procurando confirmar o problema gerado pela assimetria, realizamos o seguinte experimento: tomamos a segunda implementação (a computacionalmente correta) e adicionamos em determinado instante, e unicamente neste instante, pequenos incrementos a alguns elementos de P(n). Estes incrementos eram quase desprezíveis, cerca de 10^{-5} vezes os valores dos elementos de P(n), e foram adicionados de forma a gerar uma pequena assimetria. O resultado foi a divergência do algoritmo após aproximadamente uma centena de iterações; contrastando com as execuções normais desta implementação, cujo desempenho pôde ser observado por milhões de iterações, sem que este fenômeno se manifestasse.

A manutenção da simetria de P(n), comparando-se a primeira implementação com a segunda, tem um custo, que se reflete no aumento do número de operações. Contudo, existe ainda uma outra possibilidade de se garantir a simetria de P(n), que corresponde a utilizar somente os elementos situados na diagonal principal e acima dela. De fato, esta é visivelmente a implementação mais adequada, pois além da simetria intrínseca, exige um número menor de registradores e minimiza o número de operações aritméticas por iteração.

Uma análise complementar do Algoritmo RLS pode ser encontrada no trabalho de Verhaegen [3], onde é mostrado que sob certas condições, a preservação da simetria de P(n) garante que esta se mantém positiva definida, não obstante o acúmulo de erros. Verhaegen, contudo, não utiliza as definições das componentes Hermitiana e Anti-Hermitiana, que são próprias deste autor. Abaixo expomos um teorema que estabelece uma condição suficiente para a manutenção da definição positiva de P(n).

Teorema 3.1

Seja uma matriz de erro δP *tal que a soma de* $\bar{P}(n-1)$ *com* δP *resulte em uma matriz positiva definida. Então* $\bar{P}(n)$, *a matriz de autocorrelação inversa resultante, na iteração seguinte do Algoritmo RLS, é também positiva definida.*

Como $\bar{P}(n-1)$ é positiva definida, temos que

$$\bar{R}(n-1) = [\bar{P}(n-1)]^{-1} \tag{3.58}$$

é também positiva definida. Além disso

$$\bar{R}(n) = w\,\bar{R}(n-1) + X(n)X^H(n) \tag{3.59}$$

é positiva definida, uma vez que para qualquer vetor não nulo V, temos

$$w\,V^H\bar{R}(n-1)V + V^HX(n)X^H(n)V > 0 \tag{3.60}$$

Portanto $\bar{P}(n)$, a inversa de $\bar{R}(n)$, é também positiva definida.

Como conseqüência do Teorema 3.1, se a matriz de erro δP for incapaz de alterar a definição positiva de P(n) no instante em que ela é adicionada, isto também não poderá ocorrer em nenhum instante futuro como decorrência desta particular adição. Esta observação é significativa, pois demonstra como o Algoritmo RLS absorve os erros adicionados, não os amplificando. Por outro lado, como os erros de quantização são muito pequenos, é sempre improvável que ocorra, em uma única adição, a alteração da definição positiva de P(n), o que contribui para explicar a robustez numérica deste algoritmo (implementado na forma numericamente correta).

Na seção anterior, demonstramos que a matriz P(n) torna-se mal condicionada à medida em que w diminui. Considerando ainda as conclusões do parágrafo anterior, vemos que o Algoritmo RLS possui um interessante mecanismo de defesa para este tipo de problema pois, como mostra a equação (3.15), os elementos de P(n) crescem fortemente com a diminuição de w, de forma que os erros de quantização tornam-se menos capazes de afetar a definição positiva da matriz P(n).

Vamos agora analisar o comportamento dos algoritmos rápidos.

3.3.3 – Análise de estabilidade dos algoritmos 7N

Os algoritmos FTF e FAEST, como apresentados no Capítulo 2, são computacionalmente mais eficientes que o FK, pois possuem uma complexidade proporcional a 7 N, enquanto que este último apresenta uma complexidade computacional de 10 N. Contudo, este ganho computacional tem como preço o aumento da instabilidade numérica. Uma forma de se contornar este problema, consiste em se calcular vetorialmente o erro de predição regressiva a priori, e_b, como em (2.36), e não como o produto de escalares, dado por (2.81). Com isso, os algoritmos FTF e FAEST ficam com uma complexidade proporcional a 8N e a atualização relativa à variável $E_b(n)$ torna-se desnecessária. Os algoritmos que fazem uso deste recurso serão referidos no texto a

seguir como "algoritmos 8N"; em contraste com suas versões menos complexas, os algoritmos 7N.

Vamos explicar a geração da instabilidade observada nos algoritmos 7N tomando como base a malha de recursões formada pelas quatro equações abaixo, que foram extraídas do Algoritmo FAEST

$$e_b(n) = w\,\tilde{m}(n)\,E_b(n-1) \tag{3.61}$$

$$\zeta(n) = \zeta(n-1) + \frac{|e_a(n)|^2}{wE_a(n-1)} - \tilde{m}(n)e_b^*(n) \tag{3.62}$$

$$\varepsilon_b(n) = \frac{e_b(n)}{\zeta(n)} \tag{3.63}$$

$$E_b(n) = w\,E_b(n-1) + e_b^*(n)\varepsilon_b(n) \tag{3.64}$$

Tomemos inicialmente a variável $\zeta(n-1)$ e adicionemos um pequeno erro de quantização $\delta\zeta$.

$$\zeta(n-1) = \zeta(n-1) + \delta\zeta \tag{3.65}$$

a variável $\zeta(n-1)$, afetada pelo erro de quantização, é então usada nos demais passos da malha

$$\bar{\varepsilon}_b(n-1) = \frac{e_b(n-1)}{\zeta(n-1)} \tag{3.66}$$

$$= \frac{e_b(n-1)}{\zeta(n-1) + \delta\zeta} \tag{3.67}$$

Usando a aproximação (3.51) e (3.63), podemos reescrever (3.67) como

$$\bar{\varepsilon}_b(n-1) = \varepsilon_b(n-1)\,[1 - \frac{\delta\zeta}{\zeta(n-1)}] \tag{3.68}$$

Substituindo este resultado no cálculo da energia do erro de predição regressiva, temos

$$E_b(n-1) = w\,E_b(n-2) + e_b^*(n-1)\bar{\varepsilon}_b(n-1) \tag{3.69}$$

$$= w\,E_b(n-2) + e_b^*(n-1)\varepsilon_b(n-1)\,[1 - \frac{\delta\zeta}{\zeta(n-1)}] \tag{3.70}$$

e de (3.64) e (3.63), vem

$$E_b(n-1) = E_b(n-1) - \delta\zeta\,|\varepsilon_b(n-1)|^2 \tag{3.71}$$

O erro de predição regressiva no instante n, fica então

$$\bar{e}_b(n) = w\,\tilde{m}(n)\,\bar{E}_b(n-1) \tag{3.72}$$

$$= e_b(n) - \delta\zeta\, w\,\tilde{m}(n)\,|\varepsilon_b(n-1)|^2 \tag{3.73}$$

Onde usamos (3.61) e (3.71). A variável $\zeta(n)$, alterada pela adição do erro de quantização no instante anterior, fica então

$$\zeta(n) = \zeta(n-1) + \frac{|e_a(n)|^2}{wE_a(n-1)} - \tilde{m}(n)\bar{e}_b^*(n) \tag{3.74}$$

Substituindo (3.65), (3.73) e (3.62), resulta

$$\zeta(n) = \zeta(n) + \delta\zeta + \delta\zeta\, w\,|\tilde{m}(n)\varepsilon_b(n-1)|^2 \tag{3.75}$$

Ou ainda

$$\zeta(n) = \zeta(n) + k(n)\,\delta\zeta \tag{3.76}$$

onde

$$k(n) = 1 + w\,|\tilde{m}(n)\varepsilon_b(n-1)|^2 \tag{3.77}$$

Na equação (3.76) colocamos k(n) em evidência para mostrar que este fator é um escalar real maior ou igual a 1; o que mostra que o erro $\delta\zeta$ é amplificado na malha formada pelas equações (3.61) - (3.64), e portanto, esta malha é instável. Um resultado idêntico a este seria obtido se, ao invés da malha analisada acima, tivéssemos usado a malha correspondente do Algoritmo FTF.

Usando a operação vetorial para obter $e_b(n)$, a malha acima descrita é desfeita, e os algoritmos 8N resultantes apresentam uma característica de estabilidade numérica comparável ao Algoritmo FK. Em linhas gerais, podemos estimar que a vida dos algoritmos 7N (medida em número de iterações aproveitáveis) é prolongada em algumas dezenas de vezes quando estes são transformados em algoritmos 8N.

O problema de instabilidade apresentado pelos algoritmos 7N sugere que este tipo de implementação seja evitado, de forma que as referências feitas nas próximas seções aos algoritmos rápidos referem-se aos algoritmos 8N ou ao Algoritmo FK, que representam versões menos problemáticas, sob o ponto de vista da estabilidade numérica. Mesmo estes mostram, ainda assim, uma grande instabilidade numérica, devido principalmente à existência de outras malhas de realimentação de erros. Procuraremos analisar esta questão seguindo a mesma forma heurística utilizada acima.

3.3.4 – Análise de estabilidade dos algoritmos 8N

Na procura dos mecanismos que causam a instabilidade, é possível incorrer em erros caso alguns cuidados não sejam tomados, pois o tratamento matemático sempre inclui aproximações que podem mascarar os efeitos que se deseja observar. Dessa forma, é mais seguro investigar o comportamento dos algoritmos rápidos através de simulações e então formular um modelo matemático para descrever o processo.

O programa de computador que desenvolvemos tem a possibilidade de realizar simultaneamente duas simulações de um mesmo processo, ambas em ponto flutuante. Uma delas com precisão de 63 bits na mantissa, e a outra com precisão de 23, 39, ou 52 bits. Essa diferença de precisão pode representar um prolongamento de até 4 vezes na vida dos algoritmos rápidos, dependendo do processo que está sendo executado. A título de ilustração, apresentamos na Figura 3.8 o comportamento dos coeficientes de predição progressiva do Algoritmo FTF para três execuções de um mesmo processo de predição, onde a precisão das variáveis foi alterada de uma execução para outra. Nota-se que no primeiro caso, em que foi utilizada a precisão simples, o algoritmo divergiu com cerca de 2000 iterações, enquanto que no último, onde a maior precisão foi utilizada, ele suportou quase 8000 iterações.

A duplicidade de precisão oferecida pelo programa de simulação permite que as variáveis em precisão de 63 bits na mantissa sejam consideradas *livres de ruído de quantização* quando comparadas às variáveis representadas com menor precisão. Esta afirmativa, apesar de não valer sempre, é bastante verdadeira nas primeiras centenas de iterações - faixa interessante para a observação dos fenômenos.

Substituindo uma a uma, as variáveis do processo implementado com menor precisão pelas correspondentes variáveis livres de ruído, observa-se que, em quase todos os casos, a vida do algoritmo não é prolongada, chegando as vezes até a diminuir. Somente para o caso do preditor regressivo, B(n), é que uma melhoria apreciável é observada. Neste caso, a vida do algoritmo é prolongada até quase igualar a do processo implementado com maior precisão. Dessa observação, podemos presumir que os erros de quantização são realimentados nos algoritmos rápidos por alguma malha que inclui o vetor B(n). Na Figura 3.9 ilustramos um diagrama de blocos que representa as malhas formadas pelas variáveis vetoriais do Algoritmo FTF. Nesta figura, os blocos denotados por z^{-1} são atrasadores, enquanto que os blocos denotados por S representam uma operação de deslocamento nos elementos dos vetores. Como pode ser observado, existem duas malhas principais: uma envolvendo os vetores B(n) e G(n), e outra envolvendo os vetores A(n) e G(n). Mostraremos a seguir que os problemas de instabilidade devem ser creditados enfaticamente à primeira delas, ou seja, à malha que atualiza o preditor regressivo.

Consideremos inicialmente a adição de um vetor de erros δB, com elementos de pequenos valores, ao preditor regressivo no instante n-1

$$\bar{B}(n-1) = B(n-1) + \delta B \tag{3.78}$$

O erro de predição regressiva fica alterado para

$$\bar{e}_b(n) = e_b(n) - \delta B^H X(n) \tag{3.79}$$

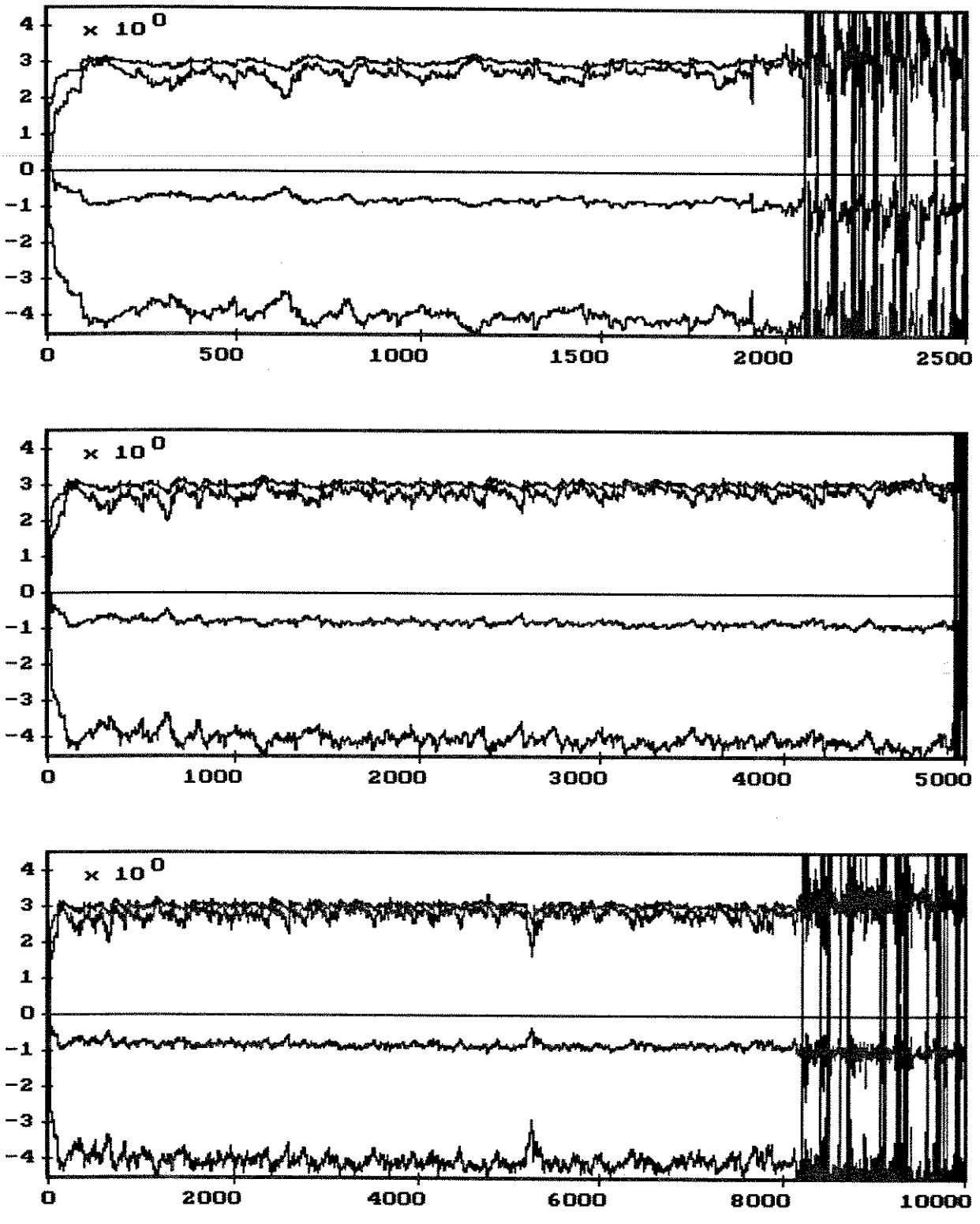


Figura 3.8 - Algoritmo FTF implementado com variáveis em ponto flutuante, N = 4, w = 0,97, excitado pelo mesmo sinal da Figura 3.6. Em **a** foi utilizado precisão de 23 bits na mantissa; em **b**, 39; e em **c**, 63 bits. A variável $\gamma(n)$ acusou anomalia para n = 1.980 em **a**; n = 4.402 em **b**; e n = 7.902 em **c**.

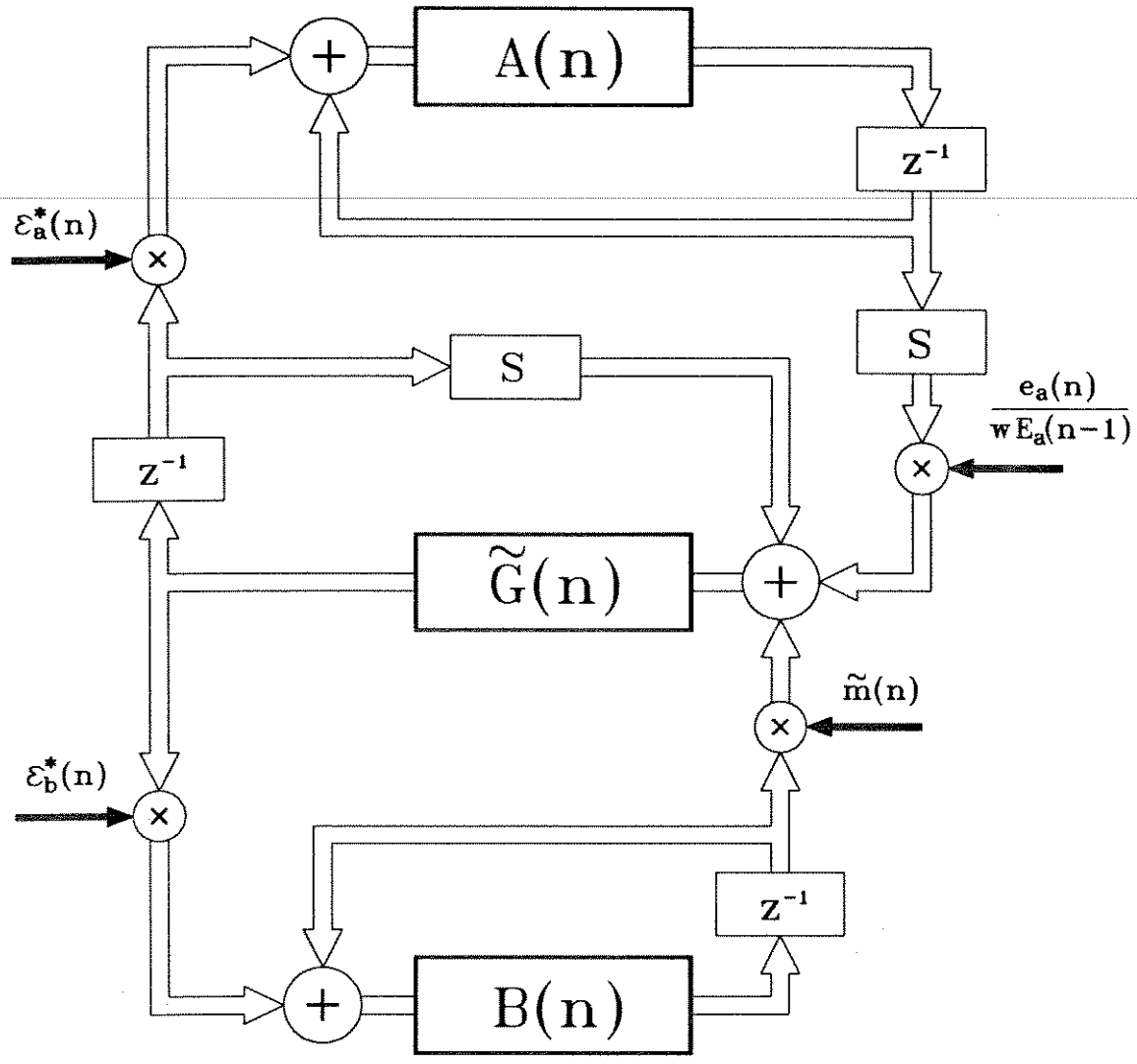


Figura 3.9 - Diagrama em blocos das variáveis vetoriais do Algoritmo FTF

e a variável de verossimilhança torna-se

$$\bar{\gamma}(n) = \frac{\gamma_1(n)}{1 - \gamma_1(n)\tilde{m}^*(n)\bar{e}_b(n)} \tag{3.80}$$

Substituindo (3.79) e usando novamente a aproximação (3.51), podemos escrever

$$\bar{\gamma}(n) = \frac{\gamma_1(n)}{1 - \gamma_1(n)\tilde{m}^*(n)e_b(n)} \left[1 - \frac{\gamma_1(n)\,\tilde{m}^*(n)\,\delta B^H X(n)}{1 - \gamma_1(n)\tilde{m}^*(n)e_b(n)}\right] \tag{3.81}$$

$$= \gamma(n)\,[1 - \gamma(n)\,\tilde{m}^*(n)\,\delta B^H X(n)] \tag{3.82}$$

O erro de predição regressiva a posteriori, fica então

$$\bar{\varepsilon}_b(n) = \bar{\gamma}(n)\,\bar{e}_b(n) \tag{3.83}$$

$$= \varepsilon_b(n) - \gamma(n)\,[1 + \varepsilon_b(n)\,\tilde{m}^*(n)]\,\delta B^H X(n) \tag{3.84}$$

onde, na última passagem, desprezamos os erros de segunda ordem. De (3.61) e (3.64), podemos escrever

$$1 + \varepsilon_b(n)\tilde{m}^*(n) = \frac{E_b(n)}{w\,E_b(n-1)} \tag{3.85}$$

Substituindo este resultado em (3.84), temos, para o erro de predição regressiva a posteriori

$$\bar{\varepsilon}_b(n) = \varepsilon_b(n) - \gamma(n)\,\frac{E_b(n)}{w\,E_b(n-1)}\,\delta B^H X(n) \tag{3.86}$$

O vetor de ganho dual é alterado para

$$\bar{G}(n) = \tilde{M}(n) + \tilde{m}(n)\,\bar{B}(n-1) \tag{3.87}$$

$$= G(n) + \tilde{m}(n)\,\delta B \tag{3.88}$$

e o preditor regressivo resultante no instante n é dado por

$$\bar{B}(n) = \bar{B}(n-1) + \bar{G}(n)\,\bar{\varepsilon}_b^*(n) \tag{3.89}$$

Substituindo (3.78), (3.87) e (3.86), e desprezando os erros de segunda ordem, temos

$$\bar{B}(n) = B(n-1) + \delta B + \bar{G}(n)\,\varepsilon_b^*(n) + \tilde{m}(n)\varepsilon_b^*(n)\,\delta B - \gamma(n)\,\frac{E_b(n)}{w\,E_b(n-1)}\,\bar{G}(n)X^H(n)\delta B \tag{3.90}$$

Usando (2.68) e (2.71), vem

$$\bar{B}(n) = B(n) + [1 + \tilde{m}(n)\varepsilon_b^*(n) - \frac{E_b(n)}{w\,E_b(n-1)}\,G(n)X^H(n)]\,\delta B \tag{3.91}$$

Substituindo novamente (3.85), resulta

$$\bar{B}(n) = B(n) + \frac{E_b(n)}{w\,E_b(n-1)}\,[I_N - G(n)X^H(n)]\,\delta B \tag{3.92}$$

e, de (3.36), temos finalmente

$$\overline{B}(n) = B(n) + \Gamma(n)\,\delta B \tag{3.93}$$

onde

$$\Gamma(n) = \frac{E_b(n)}{E_b(n-1)} P(n)R(n-1) \tag{3.94}$$

Vemos assim, que um erro adicionado nesta malha permanece indefinidamente no algoritmo ao longo do tempo sendo, em cada iteração, multiplicado pela matriz matriz $\Gamma(n)$. A relação (3.93) sugere um crescimento exponencial do ruído de quantização no Algoritmo FTF. A confirmação desta hipótese exige, contudo, outras investigações. Em primeiro lugar, o erro δB se propaga no algoritmo alterando todas as variáveis, e não apenas as componentes da malha acima analisada. Dessa forma, a menos que se prove que o efeito produzido pelas demais malhas possa ser desprezado, mesmo para o caso da adição de um erro isolado como estamos considerando, não se pode fazer afirmações sobre os instantes futuros. A relação (3.93) é garantidamente válida apenas para uma iteração. Em segundo lugar, é preciso estudar mais detalhadamente o efeito produzido pela matriz $\Gamma(n)$ como elemento de amplificação do ruído. Em primeira análise, podemos unicamente afirmar que essa matriz tende em média para I_N em um processo estacionário - o que não permite tirar grandes conclusões.

De qualquer forma, a adição de erros ao longo do tempo é, com certeza, o fator responsável pela divergência do algoritmo, e esta divergência decorre em função da realimentação realizada pela malha que contém o vetor B(n). A observação de inúmeras simulações mostra que existe uma certa constância no momento em que se dá a divergência dos algoritmos rápidos para uma mesma implementação. Ou seja, apesar da vida dos algoritmos variar extremamente em função de determinados fatores, em cada particular situação, o comportamento tende a ser o mesmo, podendo-se, dessa forma, prever o número aproximado de iterações que será suportado. Esta constatação estimula a procura de um modelo para descrever o processo de acumulação de erros nos algoritmos rápidos.

Antes de elaborar este modelo, ou mesmo, de atribuir toda a responsabilidade pela acumulação do ruído à malha de atualização do preditor regressivo, B(n), como estamos sugerindo, vamos analisar a malha do preditor progressivo, A(n), e constatar que ela não amplifica o ruído, como faz a primeira.

Adicionando um vetor de erro, δA, no preditor progressivo em n-1, temos

$$\overline{A}(n-1) = A(n-1) + \delta A \tag{3.95}$$

Os erros de predição progressiva ficam alterados para

$$\overline{e}_a(n) = e_a(n) - \delta A^H X(n-1) \tag{3.96}$$

e

$$\bar{\varepsilon}_a(n) = \bar{e}_a(n)\,\gamma(n-1) \tag{3.97}$$

$$= \varepsilon_a(n) - \gamma(n-1)\,\delta A^H X(n-1) \tag{3.98}$$

O preditor progressivo resultante no instante n, é então alterado para

$$\bar{A}(n) = \bar{A}(n-1) + \bar{G}(n-1)\,\bar{\varepsilon}_a^*(n) \tag{3.99}$$

Substituindo (3.95) e (3.98) em (3.99), usando (2.70) e (2.68), vem

$$\bar{A}(n) = A(n) + [I_N - G(n-1)X^H(n-1)]\,\delta A \tag{3.100}$$

e de (3.36), obtemos finalmente

$$\bar{A}(n) = A(n) + w\,P(n-1)R(n-2)\,\delta A \tag{3.101}$$

Notando novamente que o produto P(n-1)R(n-2) tem um valor esperado igual a matriz identidade, vemos que o erro δA tende a decrescer exponencialmente com o tempo, devido à presença de w.

A análise feita acima não é suficiente para garantir a estabilidade da malha formada por A(n) e $\bar{G}(n)$. Isto porque esta malha contém um atrasador. Este bloco impede que o ganho da malha seja avaliado em apenas uma iteração, como no caso anterior. A forma correta de se fazer a análise deveria levar em conta duas iterações seguidas, e não somente uma, como foi feito. O cômputo de duas iterações, além de muito trabalhoso, produz expressões complexas, que não permitem tirar quaisquer conclusões acerca da estabilidade. No entanto, existe uma forma mais fácil de se analisar a propagação do erro δA nesta malha. Observando o diagrama de blocos da Figura 3.9, vemos que esta malha inclui também um bloco deslocador S. Este bloco impede a realimentação direta dos vetores ao longo da malha. A cada iteração, os elementos dos vetores sofrem um deslocamento, de sorte que qualquer erro adicionado nesta malha não permanece em circulação por mais do que N iterações.

A malha formada por B(n) no diagrama da Figura 3.9 não possui nenhum bloco deslocador, de forma que os erros que afetam o vetor B(n) são mantidos no algoritmo em constante evolução.

Vamos agora demonstrar que o Algoritmo FK tem um comportamento similar ao FTF em relação à malha que atualiza o vetor B(n), e possui as mesmas características de estabilidade do Algoritmo FTF.

3.3.5 – Análise de estabilidade do Algoritmo FK

Vamos repetir o tratamento feito no item anterior, de se adicionar um erro isolado na malha de atualização do preditor regressivo, para o Algoritmo FK, e constatar que o mesmo resultado obtido para o Algoritmo FTF se repete. Com isso estaremos comprovando o fato, também

observado nas simulações, que o Algoritmo FK apresenta um desempenho comparável ao dos algoritmos 8N, no tocante à estabilidade numérica.

Adicionando δB em $B(n-1)$, temos novamente

$$\bar{B}(n-1) = B(n-1) + \delta B \tag{3.102}$$

e

$$\bar{e}_b(n) = e_b(n) - \delta B^H X(n) \tag{3.103}$$

O vetor de ganho, dado por (2.53), resulta então alterado para

$$\bar{G}(n) = \frac{1}{1 - m(n)e_b^*(n)} \left[1 - \frac{m(n)\, X^H(n)\delta B}{1 - m(n)e_b^*(n)}\right] [M(n) + m(n)\, B(n-1) + m(n)\, \delta B] \tag{3.104}$$

onde, graças à aproximação (3.51), fizemos

$$\frac{1}{1 - m(n)\bar{e}_b^*(n)} \approx \frac{1}{1 - m(n)e_b^*(n)} \left[1 - \frac{m(n)\, X^H(n)\delta B}{1 - m(n)e_b^*(n)}\right] \tag{3.105}$$

Substituindo (2.53), (2.50) e (2.56) em (3.104), e desprezando os erros de segunda ordem, temos

$$\bar{G}(n) = G(n) + \frac{\varepsilon_b(n)}{w\, E_b(n-1)} [I_N - G(n)X^H(n)]\, \delta B \tag{3.106}$$

O preditor regressivo no instante n, fica então alterado para

$$\bar{B}(n) = \bar{B}(n-1) + \bar{G}(n)\bar{e}_b^*(n) \tag{3.107}$$

Substituindo (3.103), (3.106) e (2.52), desprezando novamente os erros de segunda ordem, e rearranjando os termos restantes, temos

$$\bar{B}(n) = B(n) + \left[1 + \frac{\varepsilon_b(n)e_b^*(n)}{w\, E_b(n-1)}\right] [I_N - G(n)X^H(n)]\, \delta B \tag{3.108}$$

Substituindo (2.38), (3.36) e (3.94), temos finalmente

$$\bar{B}(n) = B(n) + \Gamma(n)\, \delta B \tag{3.109}$$

que corresponde exatamente à equação (3.93), obtida para o Algoritmo FTF no item anterior.

Podemos agora apresentar o modelo para a acumulação dos erros nos algoritmos rápidos, o qual assume que a malha de atualização do preditor regressivo é a única responsável pela geração e amplificação dos erros de arredondamento.

3.3.6 – Modelo para o crescimento do ruído nos algoritmos rápidos

A decisão de se atribuir toda a responsabilidade pela acumulação do ruído de quantização à malha do preditor regressivo parece uma hipótese excessivamente simplificadora. Contudo, além das análises feitas acima, pudemos constatar experimentalmente a grande influência dessa malha no processo de instabilidade dos algoritmos rápidos. Por outro lado, veremos que os resultados experimentais, apresentados nas simulações, confirmam os obtidos pelo modelo, atestando a validade desta hipótese.

Para viabilizar a análise, restringimos o estudo do crescimento do ruído ao caso em que o sinal de entrada é uma seqüência de amostras descorrelacionadas e com média nula. Este caso é especialmente interessante por apresentar uma grande constância na determinação do momento em que se dá a divergência.

Comecemos por definir o vetor de erro

$$F(n) = \overline{B}(n) - B(n) \tag{3.110}$$

onde $\overline{B}(n)$ representa o preditor regressivo alterado pelo ruído no instante n. Os elementos $f_i(n)$, $i = 1, 2, \ldots, N$, representam assim o erro acumulado no correspondente elemento $b_i(n)$. Assumindo que os $f_i(n)$ possuem média nula, temos que sua variância é dada por

$$\sigma_f^2(n) = E[f^2(n)] \tag{3.111}$$

Esta variância corresponde à potência do erro acumulado em cada elemento do vetor B(n). Seu crescimento ocorre devido a uma soma, em cada iteração, de uma nova quantidade de ruído, e da amplificação da potência do erro existente no instante anterior. Dessa forma, podemos usar o modelo:

$$\sigma_f^2(n) = a\,\sigma_f^2(n-1) + k \tag{3.112}$$

onde a é um escalar levemente maior que 1, e k é a variância do ruído de quantização adicionado em cada iteração. Esta equação de diferenças, com a condição de contorno $\sigma_f^2(0) = 0$, tem como solução

$$\sigma_f^2(n) = \frac{a^n - 1}{a - 1}\,k \tag{3.113}$$

O valor de k, para implementações em ponto flutuante, pode ser obtido usando-se o resultado da equação (3.32)

$$k = 0{,}18\ E[b^2(n)]\ 2^{-2b} \tag{3.114}$$

onde $E[b^2(n)]$ é o valor quadrado médio dos elementos do preditor regressivo. Na verdade, k deve ter um valor maior, pois o ruído adicionado em uma dada iteração é resultante de uma série de operações, e não de uma única quantização. Por sorte, o modelo é pouco sensível à variação

deste parâmetro, importando unicamente sua ordem de grandeza. Já o cálculo de a envolve uma análise do efeito produzido pela matriz $\Gamma(n)$, definida em (3.94). Usando (3.36) temos

$$\Gamma(n) = \frac{1}{w} \frac{E_b(n)}{E_b(n-1)} [I_N - P(n)X(n)X^H(n)] \tag{3.115}$$

Desprezando o fator $E_b(n)/E_b(n-1)$, notando que $P(n)X(n)X^H(n)$ apresenta valores pequenos quando comparados a I_N, e lembrando que o sinal de entrada é descorrelacionado, podemos usar a aproximação (3.15), resultando

$$\Gamma(n) = \frac{1}{w} [I_N - \frac{1-w}{\sigma^2} X(n)X^H(n)] \tag{3.116}$$

onde σ^2 é a variância do sinal de entrada x(n). Notando que a diagonal principal da matriz $X(n)X^H(n)$ apresenta elementos positivos e de média σ^2, podemos usar o seguinte modelo estatístico:

$$X(n)X^H(n) = \sigma^2 I_N - Q(n) \tag{3.117}$$

onde Q(n) é uma matriz cujos elementos são todos variáveis aleatórias com médias nulas e variâncias dadas por

$$\sigma_{q_{ij}} = \begin{cases} 2\sigma^4 & \text{se } i = j \\ \sigma^4 & \text{se } i \neq j \end{cases} \tag{3.118}$$

Substituindo (3.117) em (3.116), temos

$$\Gamma(n) = I_N + \frac{1-w}{w\sigma^2} Q(n) \tag{3.119}$$

Usando as propriedades da multiplicação e da adição de variáveis aleatórias, a relação (3.93) mostra que a amplificação exercida pela matriz $\Gamma(n)$ na variância dos elementos do vetor F(n) é dada por

$$a = 1 + \frac{[N+1][1-w]^2}{w^2} \tag{3.120}$$

Admitindo que a divergência do algoritmo ocorra quando $\sigma_f^2(n)$ atinja determinado valor, temos, de (3.113) e (3.114), que este valor é dado por

$$\sigma_D^2 = \frac{a^{n_D}}{a-1} 0{,}18\, E[b^2(n)]\, 2^{-2b} \tag{3.121}$$

onde n_D é o instante de tempo em que se dá a divergência. O número 1 do nominador foi omitido, em virtude de seu pequeno valor frente à a^{n_D}.

Foi notado experimentalmente que a divergência do algoritmo ocorre invariavelmente quando o valor de $\sigma_f^2(n)$ atinge aproximadamente um décimo do valor de $E[b^2(n)]$. Também neste caso o modelo é pouco sensível à variação no valor desta relação e pode ser arbitrado com alguma liberdade. Tomando, por conveniência

$$\sigma_D^2 = 0{,}18\, E[b^2(n)] \tag{3.122}$$

temos

$$n_D = \frac{1}{\ln(a)} [\ln(a-1) + 2b \ln(2)] \tag{3.123}$$

Dado que a tem um valor próximo de 1, vale a aproximação $\ln(a) \approx [a-1]$. Assim

$$n_D = \frac{w^2 [\ln(N+1) + 2\ln(1-w) - 2\ln(w) + 1{,}4\, b]}{[N+1][1-w]^2} \tag{3.124}$$

Uma vez que não podemos esperar um resultado senão aproximado do instante de divergência, outras simplificações podem ser feitas. Usando N no denominador, ao invés de N+1, e desprezando alguns termos em função de seus pequenos valores em relação ao termo 1,4 b, temos

$$n_D = \frac{1{,}4\, b}{[1-w]^2 N} \tag{3.125}$$

A equação (3.125) descreve bem o comportamento dos algoritmos rápidos implementados com variáveis em ponto flutuante. Em particular, ela estabelece que a vida do algoritmo deve ser proporcional ao número de bits utilizado na representação das variáveis. Este é, de fato, um aspecto árduo dos algoritmos rápidos, que já foi comentado anteriormente e também ilustrado na Figura 3.8. É desestimulante constatar que o esforço de se aumentar o limite de precisão das variáveis não prolonga a vida do algoritmo senão pelo mesmo fator usado no acréscimo do número de bits.

O comportamento da potência do erro em relação ao limite de precisão das variáveis pode ser observado nas simulações da Figura 3.10. Nesta figura está ilustrado o crescimento do módulo ao quadrado do vetor de erros, F(n), para duas simulações de um mesmo processo, uma com precisão de 23 bits e outra com precisão de 52 bits. Pode-se observar claramente, em ambas as implementações, que o crescimento do erro ocorre de forma exponencial. Além disso, como previsto pelo modelo da equação (3.113), pode ser notado um fato bastante curioso: *que uma maior precisão na representação das variáveis não diminui a taxa de crescimento do erro, fazendo apenas com que este comece a ser desenvolvido em um ponto inferior.* Notemos que, na parte **b** da Figura 3.10, o parâmetro de erro começa com uma amplitude de aproximadamente 10^{-24}. A partir do momento em que ele atinge o valor 10^{-10}, seu crescimento se dá de forma semelhante ao da implementação de menor precisão, ilustrado na parte **a** desta figura.

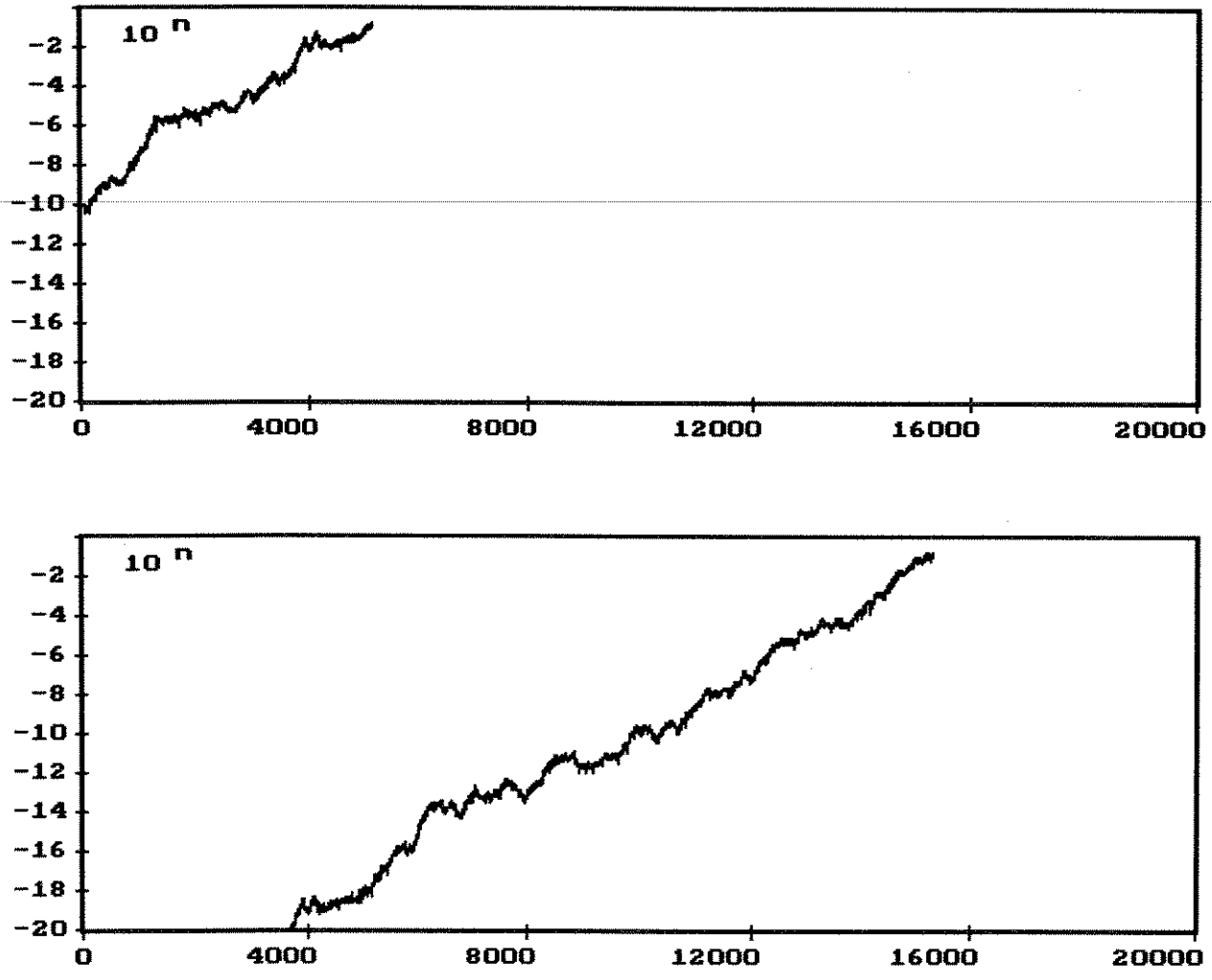


Figura 3.10 - Parâmetro $|\hat{B}(n) - B(n)|^2$ obtido em um processo de identificação de sistemas usando o Algoritmo FTF com N = 4, w = 0,97, excitado por um ruído branco, gaussiano, de média nula e variância 1. Em **a** foi usado ponto flutuante com precisão de 23 bits na mantissa. Em **b** foi usado ponto flutuante com precisão de 52 bits.

Em inúmeras simulações realizadas, pudemos constatar que o modelo representado pela equação (3.124) descreve bem o processo de acumulação de erros nos algoritmos rápidos para a faixa de valores interessantes do fator de esquecimento, da ordem do filtro e do número de bits. Seu uso permite que se preveja, com razoável exatidão, o instante que determina a divergência do algoritmo.

Na Tabela 3.2 apresentamos alguns resultados que permitem constatar a validade da equação (3.124). Em **a** estão relacionados os números obtidos usando-se a fórmula (3.124) para o caso de N = 4, e para uma série de valores de b e de w. Em **b** está ilustrado, para efeito de comparação, os resultados obtidos experimentalmente, com o uso do Algoritmo FTF, para as mesmas condições relacionadas em **a**. Na Tabela 3.3 apresentamos uma comparação semelhante,

Tabela 3.2 - Comprovação da validade da equação (3.124) para N = 4

a - valores de n_D obtidos com a fórmula (3.124)

w \ b	b = 23	b = 39	b = 52	b = 63
w = 0,9	485	856	1.157	1.412
w = 0,99	47.652	91.142	126.478	156.377
w = 0,999	3.928.281	8.355.566	11.952.735	14.996.494

b - valores de n_D obtidos experimentalmente

w \ b	b = 23	b = 39	b = 52	b = 63
w = 0,9	775	1.560	2.140	2.630
w = 0,99	53.400	122.000	163.000	196.000
w = 0,999	3.870.000	9.600.000	13.000.000	19.000.000

Tabela 3.3 - Comprovação da validade da equação (3.124) para w = 0,99

a - valores de n_D obtidos com a fórmula (3.124)

N \ b	b = 23	b = 39	b = 52	b = 63
N = 4	47.652	91.142	126.478	156.377
N = 40	6.326	11.639	15.956	16.609
N = 400	715	1.268	1.718	2.098

b - valores de n_D obtidos experimentalmente

N \ b	b = 23	b = 39	b = 52	b = 63
N = 4	53.400	122.000	163.000	196.000
N = 40	3.970	11.500	15.700	20.200
N = 400	2.060	4.870	7.330	9.280

desta vez mantendo-se fixo o fator de esquecimento em w = 0,99 e variando-se N e b. Pode-se notar a grande proximidade entre os números obtidos com a fórmula e os resultados experimentais, o que confirma a validade do modelo.

Talvez uma ressalva deva ser feita quanto à ordem N do filtro. A partir de N = 200 nota-se uma constância no valor de n_D, deixando de valer o efeito "inversamente linear" previsto pela equação (3.125). Entretanto, para filtros de ordens inferiores a 200, o modelo se aplica satisfatoriamente.

O sinal de entrada, x(n), utilizado em **b**, conforme estabelecido no desenvolvimento do modelo, foi uma seqüência de amostras gaussianas, descorrelacionadas e com média nula. Também fizemos a variância de x(n) ser unitária e assumimos o instante de divergência, n_D, como sendo o momento em que a variável de verossimilhança, $\gamma(n)$, apresenta um valor fora de sua faixa de validade.

De forma geral, é necessário guardar uma boa margem de segurança, de pelo menos uma ordem de grandeza, quanto ao instante n_D. Com isso, a precisão fornecida pela equação (3.124) é satisfatória para quaisquer efeitos práticos. Contudo, esta equação foi desenvolvida com o único propósito de descrever o processo de acumulação de erros. Para fins de aplicações práticas, recomendamos as técnicas de estabilização discutidas no Capítulo 4.

Ao longo deste capítulo pudemos analisar os vários fatores que influem na estabilidade dos algoritmos rápidos. Vamos, a seguir, considerar outros parâmetros, que na realidade constituem condições de contorno, mas que também interferem no processo de estabilidade dos algoritmos rápidos.

3.4 – INFLUÊNCIA DE OUTROS FATORES

As condições iniciais, assim como a potência do sinal de entrada, também podem influenciar a estabilidade dos algoritmos rápidos. Estes pontos não foram incluídos em nossa consideração inicial dos fatores de instabilidade. O primeiro, devido ao seu caráter transitório, e o segundo, devido à sua forma indireta de ação no algoritmo.

3.4.1 – Influência da energia inicial de predição, E_o

Os processos simulados em computador demonstram que a ocorrência da divergência pode ser antecipada em até algumas centenas de iterações quando se usa valores muito pequenos para E_o. Antes de atacarmos este problema, é cabível uma ressalva quanto a uma questão de consistência com o Algoritmo RLS.

Para efeito de comparação de resultados, é desejável que tanto as simulações do Algoritmo RLS quanto a dos algoritmos rápidos apresentem saídas iguais para uma mesma seqüência de entrada. A forma usual de se obter essa situação consiste em se manter as condições iniciais,

descritas na Seção 2.6, para os algoritmos rápidos, e introduzir uma pequena modificação na matriz diagonal, P(0), sugerida no Capítulo 1 para a inicialização do Algoritmo RLS:

$$P(0) = \frac{1}{E_o} W \tag{3.126}$$

onde

$$W = \text{diag}(1, w, w^2, \ldots, w^{N-1}) \tag{3.127}$$

Na equação (3.126), E_o é o valor inicial da energia do erro de predição progressiva empregado nos algoritmos rápidos. A obtenção desta equação é feita, considerando-se a seqüência de entrada descrita na Seção 2.6, cujas amostras são todas nulas para $n \leq 0$, exceto a amostra x(-N), que é dada por

$$|x(-N)|^2 = E_o\, w^{-N} \tag{3.128}$$

Deve ser notado que o parâmetro E_o desempenha o papel da constante δ mencionada na Seção 1.4. A escolha de E_o admite uma ampla faixa de valores, podendo em teoria assumir qualquer valor positivo. Em uma primeira aproximação, deve ser considerado como válida a gama de valores que a variável $E_a(n)$ pode apresentar. Dessa forma, um limitante superior natural para E_o é dado por

$$E_o \leq \frac{\sigma_x^2}{1 - w} \tag{3.129}$$

que é o valor esperado para $E_a(n)$ quando o algoritmo é excitado por um ruído branco de média nula e variância σ_x^2. Para sinais com amostras correlacionadas este valor diminui, tendendo a zero na medida em que {x(n)} torna-se um sinal predizível.

A Figura 3.11 ilustra o comportamento dos coeficientes de predição progressiva no Algoritmo FTF para três diferentes valores de E_o. Notadamente, quanto menor o valor de E_o, menor é o tempo necessário para os coeficientes convergirem, pois menor será a interferência secundária imposta por este parâmetro.

Por outro lado, o uso de valores excessivamente pequenos para E_o deve ser evitado, devido a ocorrência de um sobressalto apresentado pelas variáveis nos instantes de tempo iniciais. Este sobressalto pode ser observado nos coeficientes de predição progressiva da Figura 3.11 para o caso em que $E_o = 0{,}001$. Ele é resultado do grande valor inicial assumido pelos elementos do vetor $\bar{G}(n)$ em conseqüência de um denominador pequeno. Na medida em que E_o diminui, o sobressalto tende a aumentar, atingindo níveis muito altos. Nesse caso, observa-se também um outro efeito indesejável, que é a divergência das variáveis logo nos primeiros instantes de tempo. Para o caso de implementações em ponto flutuante, com 23 bits na mantissa, esta situação ocorre quando $E_o < 10^{-6}$.

Na falta de um critério mais decisivo para se arbitrar o valor de E_o, julgamos que um bom método consiste em se escolher um valor bem pequeno para garantir uma rápida convergência,

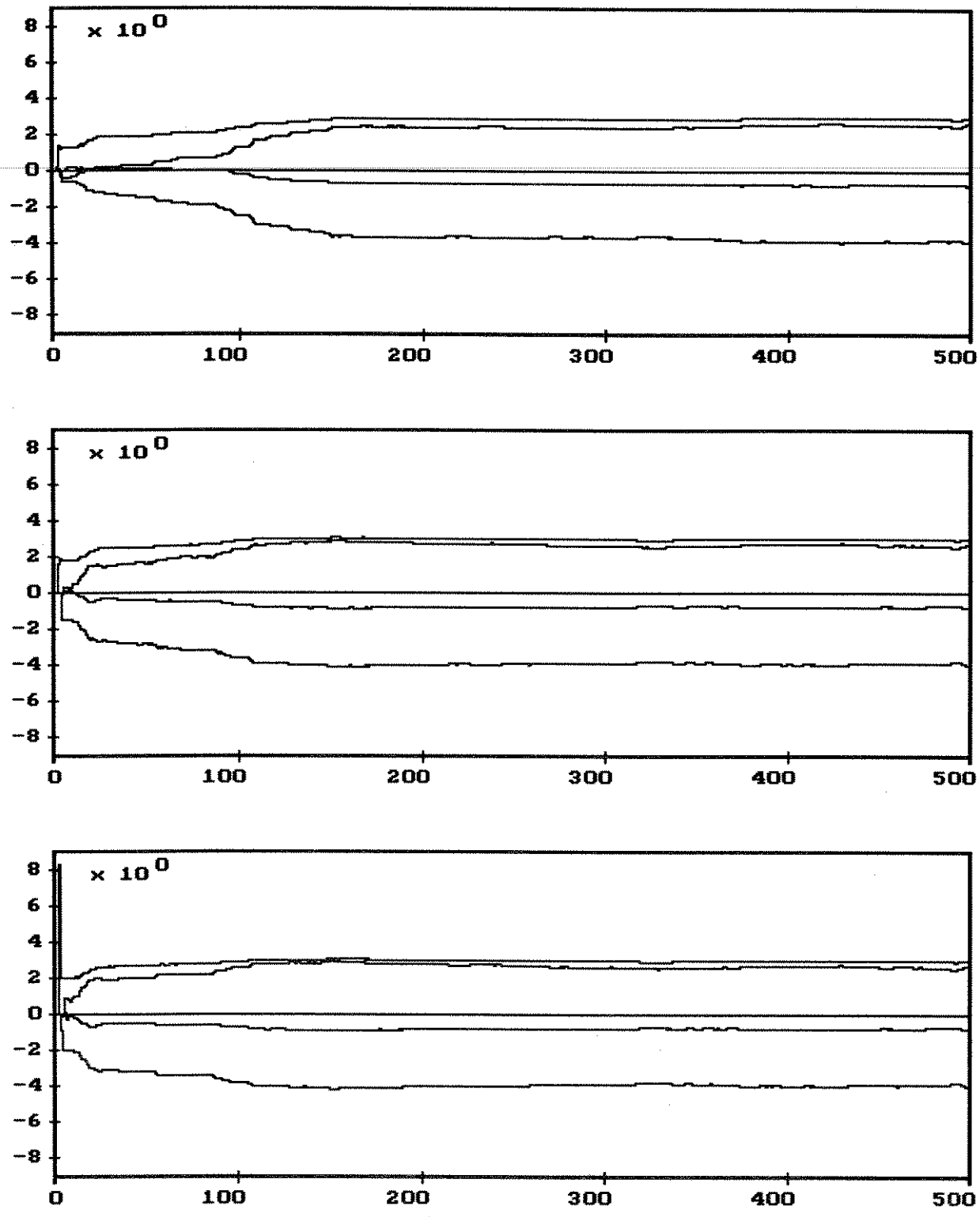


Figura 3.11 - Comportamento dos coeficientes de predição progressiva para três diferentes valores de E_o no Algoritmo FTF, com N = 4 e w = 0,99. Nas três execuções foi utilizada a mesma seqüência de entrada - um processo AR com χ = 3.000, média nula e variância 1. Em **a**, E_o = 0,1, em **b**, E_o = 0,01 e em **c**, E_o = 0,001.

porém, não tão pequeno que provoque o sobressalto das variáveis. Para os diversos casos que pudemos simular, notamos que

$$E_o = 0{,}01\ \sigma_x^2 \tag{3.130}$$

representa uma boa escolha, pois atende os requisitos. Variações de uma ordem de grandeza em torno deste valor são também plenamente aceitáveis.

Nos algoritmos rápidos, a escolha de valores demasiadamente grandes para E_o traz, como conseqüência, um atraso no surgimento da instabilidade. Contudo, observamos que este atraso corresponde justamente ao tempo gasto para o decaimento do alto valor de E_o por ação do fator de esquecimento; o que não representa uma melhoria no desempenho, já que o prolongamento da vida do algoritmo equivale ao tempo gasto para os coeficientes convergirem.

3.4.2 – Influência da potência do sinal de entrada

A potência do sinal de entrada é outro fator que influencia a estabilidade dos algoritmos rápidos. Em qualquer experimento onde se aumenta este parâmetro em proporções elevadas (algumas ordens de grandeza), observa-se uma diminuição na vida do algoritmo.

Este efeito pode ser explicado da seguinte forma: O aumento da amplitude do sinal de entrada implica em um aumento no valor absoluto de praticamente todas as demais variáveis do algoritmo. Os erros de quantização, para variáveis em ponto flutuante, dependem diretamente de sua amplitude, conforme descrito pela equação (3.32). Com isso, os erros adicionados em cada iteração assumem uma magnitude proporcionalmente maior, acarretando um maior acúmulo de ruído.

3.5 – COMENTÁRIO

Neste capítulo procuramos esclarecer o problema da instabilidade numérica dos algoritmos rápidos. Vimos que os fatores que mais influem neste processo são: a característica do sinal de entrada, o fator de esquecimento, e os erros de arredondamento.

Procuramos também contrastar o desempenho dos algoritmos rápidos com o do Algoritmo RLS, mostrando primeiramente os fatores que determinam a grande estabilidade deste último. Achamos que o entendimento deste processo seria de fundamental importância, pois ele reside na diferença estrutural existente entre os dois tipos de algoritmo. Enquanto no Algoritmo RLS toda uma matriz de elementos é repassada de um iteração à outra, nos algoritmos rápidos este grande número de elementos não existe, graças à presença dos preditores. No Algoritmo RLS, o vetor G(n) é obtido em cada instante considerando-se unicamente a matriz P(n) e o respectivo vetor de entrada. Com isso, a única entidade realmente atualizada é a matriz P(n). Nos algoritmos rápidos, ao contrário, os preditores são envolvidos na atualização de G(n), que por sua vez, é utilizado na atualização dos preditores, formando assim malhas que podem acumular e amplificar erros de arredondamento.

Mostramos que o acúmulo do ruído de quantização é um fenômeno que está basicamente relacionado com a malha de atualização do preditor regressivo e propusemos um modelo para descrever este processo para o caso de algoritmos rápidos implementados com variáveis em ponto flutuante. Vimos que os resultados obtidos por este modelo representam com fidelidade o processo de acumulação de erro no preditor regressivo, sendo inclusive possível estimar, com razoável precisão, o instante em que ocorre a divergência.

Como resultado mais significativo do modelo, vimos que a taxa de crescimento da potência do erro independe da precisão utilizada na representação das variáveis, o que não é um fato intuitivo. O aumento do número de bits meramente faz com que o erro comece a ser desenvolvido em um nível inferior. Conclui-se então, que é totalmente improdutivo tentar estabilizar o algoritmo através do aumento do número de bits.

No próximo capítulo iremos discutir e propor métodos para contornar o problema da instabilidade numérica dos algoritmos rápidos.

APÊNDICE 3A

Neste apêndice vamos demonstrar as equações (3.16a) e (3.16b), que estabelecem, respectivamente, as variâncias dos elementos da diagonais principal e das diagonais secundárias da matriz de autocorrelação determinística de um processo estacionário cujas amostras são gaussianas, independentes, de média nula e variância σ^2. Consideremos para isso uma transformação do tipo

$$y = x^2 \tag{3.131}$$

onde x é uma variável aleatória gaussiana de média nula e variância σ^2. Através dessa transformação, encontramos a distribuição de probabilidade

$$P(y) = \frac{1}{\sqrt{2\pi y\sigma}} e^{-\frac{y}{2\sigma^2}} \tag{3.132}$$

A variância de y pode ser obtida através da relação

$$\sigma_y^2 = E[y^2] - E^2[y] \tag{3.133}$$

Usando (3.131), temos

$$\sigma_y^2 = E[x^4] - E^2[x^2] \tag{3.134}$$

ou ainda, dado que x tem média nula

$$\sigma_y^2 = E[x^4] - \sigma^4 \tag{3.135}$$

A esperança de x^4 pode ser obtida usando-se a relação

$$E[x^4] = (-j)^4 \frac{d^4}{d\omega^4} C(\omega) \Big|_{\omega=0} \tag{3.136}$$

onde

$$C(\omega) = e^{\frac{\omega^2 \sigma^2}{2}} \tag{3.137}$$

é a função geradora associada à variável gaussiana. De (3.136) e (3.137) temos

$$E[x^4] = 3\,\sigma^4 \tag{3.138}$$

que substituído em (3.135) resulta

$$\sigma_y^2 = 2\,\sigma^4 \tag{3.139}$$

Podemos agora obter a variância dos elementos da diagonal principal da matriz de autocorrelação determinística. Lembrando que estes elementos são dados por uma soma ponderada onde todos os termos tem densidade de probabilidade igual a P(y), ou seja

$$r_{ii}(n) = \sum_{k=1}^{n} w^{n-k} x^2(k+1-i) \tag{3.140}$$

temos que a variância de r_{ii} para $n \to \infty$ é dada por

$$\sigma_{r_{ii}}^2 = \frac{2}{1-w^2}\,\sigma^4 \tag{3.141}$$

A distribuição de probabilidade de r_{ii}, em virtude do Teorema do Limite Central, é aproximadamente gaussiana com média $\sigma^2/(1-w)$ e variância $2\sigma^4/(1-w^2)$

Para os elementos situados fora da diagonal principal, consideremos a transformação

$$z = x_1\, x_2 \tag{3.142}$$

onde x_1 e x_2 são variáveis gaussianas independentes, com médias nulas e variâncias σ^2. Procedendo como anteriormente, temos que a média de z é nula e a variância é dada por

$$\sigma_z^2 = \sigma^4 \tag{3.143}$$

Os elementos das diagonais secundárias da matriz de autocorrelação determinística são do tipo

$$r_{ij}(n) = \sum_{k=1}^{n} w^{n-k}\, x(k+1-i)\, x(k+1-j) \qquad i \neq j \qquad (3.144)$$

e portanto, a variância de r_{ij}, $i \neq j$, para $n \to \infty$, é dada por

$$\sigma^2_{r_{ij}} = \frac{1}{1-w^2}\,\sigma^4 \qquad i \neq j \qquad (3.145)$$

Invocando novamente o Teorema do Limite Central, podemos presumir que a distribuição de probabilidade destes elementos é também aproximadamente gaussiana e tem média nula.

APÊNDICE 3B

Neste apêndice vamos mostrar como é possível obter uma variável aleatória com distribuição gaussiana a partir de variáveis com distribuição uniforme. Este procedimento é especialmente interessante para gerar seqüências pseudoaleatórias gaussianas em computadores. Isto porque os compiladores, em geral, somente permitem a geração de números randômicos com distribuição uniforme.

Seja y, uma variável aleatória com função de densidade de probabilidade uniforme no intervalo [0,1]. Queremos u, gaussiana, com média nula e variância 1. Para isso seria suficiente relacionar y e u através de

$$u = F^{-1}(y) \qquad (3.146)$$

onde

$$F(u) = \int_{-\infty}^{u} \frac{1}{\sqrt{2\pi}}\, e^{-\frac{\alpha^2}{2}}\, d\alpha \qquad (3.147)$$

Contudo, na prática, isto não é possível, pois não é conhecida uma forma fechada para a integral acima. Dessa forma, a solução é definir uma variável auxiliar r, com distribuição de Rayleigh

$$r = G^{-1}(y) \qquad (3.148)$$

onde

$$G(r) = \int_{0}^{r} \alpha\, e^{-\frac{\alpha^2}{2}}\, d\alpha \qquad (3.149)$$

De (3.148) e (3.149), temos

$$r = \sqrt{-2\ln(1-y)} \tag{3.150}$$

Notando que (1-y) tem a mesma distribuição de y, podemos simplificar (3.150) para

$$r = \sqrt{-2\ln(y)} \tag{3.151a}$$

Definindo também

$$\theta = 2\pi z \tag{3.151b}$$

onde z tem a mesma distribuição de probabilidade de y, temos

$$p(\theta) = \frac{1}{2\pi} \qquad 0 \le \theta \le 2\pi \tag{3.152a}$$

e

$$p(r) = r\, e^{-\frac{r^2}{2}} \qquad 0 \le r \tag{3.152b}$$

Considerando agora as transformações

$$u = r\cos\theta \tag{3.153a}$$

e

$$v = r\,\mathrm{sen}\,\theta \tag{3.153b}$$

podemos utilizar o método do Jacobiano, e obter p(u,v) a partir de

$$p(u,v) = \frac{p(r,\theta)}{|J|} \tag{3.154}$$

onde J é o Jacobiano da transformação

$$J = \begin{vmatrix} \frac{\partial u}{\partial r} & \frac{\partial u}{\partial \theta} \\ \frac{\partial v}{\partial r} & \frac{\partial v}{\partial \theta} \end{vmatrix} \tag{3.155}$$

e

$$p(r,\theta) = p(r)\,p(\theta) \tag{3.156}$$

em virtude da independência entre as variáveis r e θ. Substituindo (3.156), (3.155) e (3.153) em (3.154), temos

$$p(u,v) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} \frac{1}{\sqrt{2\pi}} e^{-\frac{v^2}{2}} \tag{3.157}$$

Vemos então que p(u) e p(v) são variáveis gaussianas, independentes, com média nula e variância unitária. Dessa forma, a geração de amostras gaussianas pode ser obtidas com o uso da fórmula

$$u = \sqrt{-2\ln(y)}\cos(2\pi z) \tag{3.158a}$$

ou

$$v = \sqrt{-2\ln(y)}\,\mathrm{sen}(2\pi z) \tag{3.158b}$$

onde y e z são variáveis independentes e com distribuição uniforme no intervalo [0,1], do tipo disponível nas várias linguagens de computador.

3.6 – REFERÊNCIAS

[1] Haykin, S., Adaptive Filter Theory, Prentice-Hall, 1986.

[2] Ljung, S., Ljung, L., "Error propagation properties of recursive least-squares adaptation algorithms", Automatica, vol 21, pp 157-167, 1985.

[3] Verhaegen, M. W., "Round-off error propagation in four generally-applicable, recursive, least-squares estimation schemes", Automatica, vol 25, pp 437-444, 1989.

[4] Bellanger, M. G., Adaptive Filters and Signal Analysis, Marcel Dekker Inc., New York, 1987.

[5] Oppenheim, A. V., Schafer, R. W., Digital Signal Processing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[6] Sripad, A. B., Snyder, D. L., "Quantization errors in floating-point arithmetic", IEEE Trans. Acoustics, Speech, Signal Processing, vol ASSP-26, pp. 456-463, Oct. 1978.

[7] Proakis, J. G., Digital Communications, Mc Graw-Hill, 1983.

[8] Cioffi, J. M., "Limited-Precision Effects in Adaptive Filtering", IEEE trans. on Circuits and Systems, vol 34, pp 821-833, July 1987.

[9] Ardalan, S. H., "Floating-point error analysis of recursive least-squares and least-mean-squares adaptive filters", IEEE Trans. Circuits and Systems, vol CAS-33, pp. 1192-1208, Dec. 1986.

CAPÍTULO 4

ALGORITMOS RÁPIDOS NUMERICAMENTE ESTÁVEIS

O controle da instabilidade numérica nos algoritmos rápidos é um problema de difícil solução. Como dissemos anteriormente, não existe uma ferramenta matemática adequada ao seu tratamento. O tedioso método da tentativa e erro representa, sem dúvida, um valioso recurso na busca de artifícios para manter o algoritmo livre da acumulação crescente de ruído. A aplicação deste método, no entanto, quase sempre falha, ou apresenta resultados insatisfatórios. Qualquer modificação efetuada na estrutura do algoritmo, sem um criterioso estudo, produz um verdadeiro desastre já nas primeiras iterações. A simples soma de um incremento, ainda que de valor muito pequeno, em qualquer das variáveis, na tentativa de produzir um vazamento, por exemplo, resulta na antecipação acentuada da divergência. As variáveis passam a assumir um comportamento caótico, com valores tendendo a extremos e fora da faixa de validade, demonstrando a enorme sensibilidade desses algoritmos.

Ainda assim, existem propostas para a solução do problema de instabilidade numérica. As técnicas mais interessantes, serão discutidas neste capítulo. Também será apresentado um novo método para a estabilização do Algoritmo FTF.

A nosso ver, as técnicas de estabilização se enquadram em três diferentes categorias: as que não alteram a estrutura do algoritmo, permitindo seu uso apesar da instabilidade; as que alteram a estrutura do algoritmo, mutilando a solução matematicamente precisa do problema LS; e por último, as que modificam o algoritmo sem contudo alterar a solução LS.

Logicamente, as sugestões que se enquadram nesta última categoria são as mais atrativas, e por isso daremos a elas uma maior atenção. Iremos estudar pela ordem já mencionada os três tipos de técnicas, começando por uma das primeiras propostas para a utilização dos algoritmos rápidos, conhecida como método da reinicialização.

4.1 – MÉTODOS QUE NÃO ALTERAM O ALGORITMO

O método da reinicialização foi implicitamente mencionado em alguns pontos dos capítulos anteriores, sobretudo quando discutimos as condições iniciais para os algoritmos rápidos. Iremos primeiro discutir mais detalhadamente alguns aspectos deste tipo de implementação e a seguir apresentar a sugestão feita por Lin, que também faz uso da reinicialização, e constitui uma sofisticação do problema LS no âmbito dos algoritmos rápidos.

4.1.1 – O método da reinicialização

Neste método, que pode ser aplicado a qualquer um dos algoritmos rápidos, todas as atualizações são preservadas integralmente, permitindo inclusive o acúmulo do ruído de quantização. A idéia básica é aproveitar a série inicial de iterações confiáveis, reinicializando periodicamente o algoritmo para evitar a divergência.

A reinicialização pode ser feita a intervalos pré-definidos, com uma periodicidade que garanta a operação segura do algoritmo. Para isso, o modelo elaborado no Capítulo 3 fornece um limitante superior ao número de iterações, devendo-se apenas considerar uma boa margem de segurança. Por outro lado, também é possível reinicializar as variáveis somente quando for detectada a iminência do fenômeno da divergência. Isso é conseguido através da observação das chamadas "variáveis de alerta" [1].

As variáveis de alerta, em geral, possuem valores teoricamente restritos a uma determinada faixa de validade. O acúmulo de ruído no algoritmo faz com que estas variáveis passem a apresentar valores fora dessa faixa, indicando uma situação anômala, alheia ao critério LS. Dessa forma, a manifestação da divergência pode ser prevista com uma antecedência de, tipicamente, dezenas ou centenas de iterações através da ocorrência de um valor indevido.

No Algoritmo FK, as variáveis que aparecem nos denominadores são potenciais candidatas a servir de alerta contra a instabilidade. Tanto $E_a(n)$ quanto $1 - e_b^*(n)m(n)$ são teoricamente positivas, mas podem se tornar negativas nos instantes que antecedem a divergência em uma implementação com precisão finita. As simulações mostram que as relações

$$1 - e_b^*(n)m(n) > 0 \tag{4.1}$$

e

$$1 - e_b^*(n)m(n) \leq 1 \tag{4.2}$$

representam as condições mais vulneráveis, pois suas violações precedem qualquer outra. Apesar de não haver uma comprovação matemática de que a violação dessas relações ocorra antes da divergência do algoritmo, a prática mostra que estes eventos sempre se sucedem nesta ordem. Em [2] a relação (4.1) é apontada como sendo o melhor indicador de instabilidade para o Algoritmo FK. Em nossa experiência, entretanto, a relação (4.2) é a que se mostrou mais susceptível aos problemas da precisão finita. Para os casos que analisamos, de sinais estacionários, com excitação persistente e implementações com variáveis em ponto flutuante, a violação

de (4.2) ocorre invariavelmente antes que a de (4.1). Outras relações existem para se conferir o estado de correção do Algoritmo FK, como por exemplo

$$0 < \frac{\varepsilon_a(n)}{e_a(n)} \leq 1 \tag{4.3}$$

mas sua violação, assim como a do sinal de $E_a(n)$, não se relaciona tanto com o processo de instabilidade quanto a violação das relações (4.1) e (4.2).

A variável de alerta usada no Algoritmo FK pode também ser expressa em termos de variáveis do Algoritmo FTF. Usando (2.56) e (2.87), temos

$$1 - e_b^*(n)m(n) = \frac{\gamma_1(n)}{\gamma(n)} \tag{4.4}$$

Contudo, a experiência mostra que para detectar a instabilidade no Algoritmo FTF é suficiente a checagem da relação

$$\gamma(n) \leq 1 \tag{4.5}$$

Uma outra abordagem ao problema da avaliação do grau de instabilidade dos algoritmos rápidos, ou da definição de uma variável de alerta, consiste em se usar um parâmetro cujo valor é nulo para o caso de precisão infinita, mas que apresenta valores diferentes de zero quando os efeitos numéricos estão presentes. Assim, por exemplo, de (2.50) e (2.89), temos

$$\frac{\varepsilon_b(n)}{E_b(n)} = \gamma_1(n)\,\tilde{m}(n) \tag{4.6}$$

De forma que a variável

$$\eta(n) = \left| \frac{\varepsilon_b(n)}{E_b(n)} - \gamma_1(n)\,\tilde{m}(n) \right| \tag{4.7}$$

deveria ser constantemente nula. Na prática, contudo, isto não ocorre. Inicialmente ela apresenta valores muito pequenos que, com o tempo, passam a assumir maiores proporções, em virtude do acúmulo de ruído. Dessa forma, é possível estabelecer um limiar para decidir o instante que o algoritmo deve ser reinicializado.

Assim como $\eta(n)$, outras variáveis podem ser definidas para indicar o nível de anormalidade dos algoritmos rápidos. Algumas delas serão apresentadas na Seção 4.3 em um outro contexto, que não o da reinicialização. De forma geral, elas são obtidas pela subtração de duas variáveis escalares cujos valores, teoricamente iguais, podem ser calculados de maneiras diferentes.

A Figura 4.1 ilustra a evolução das variáveis $\gamma(n)$ e $\eta(n)$ em uma simulação do Algoritmo FTF. Pode-se notar que tanto uma como a outra denunciam o surgimento da divergência com razoável antecedência.

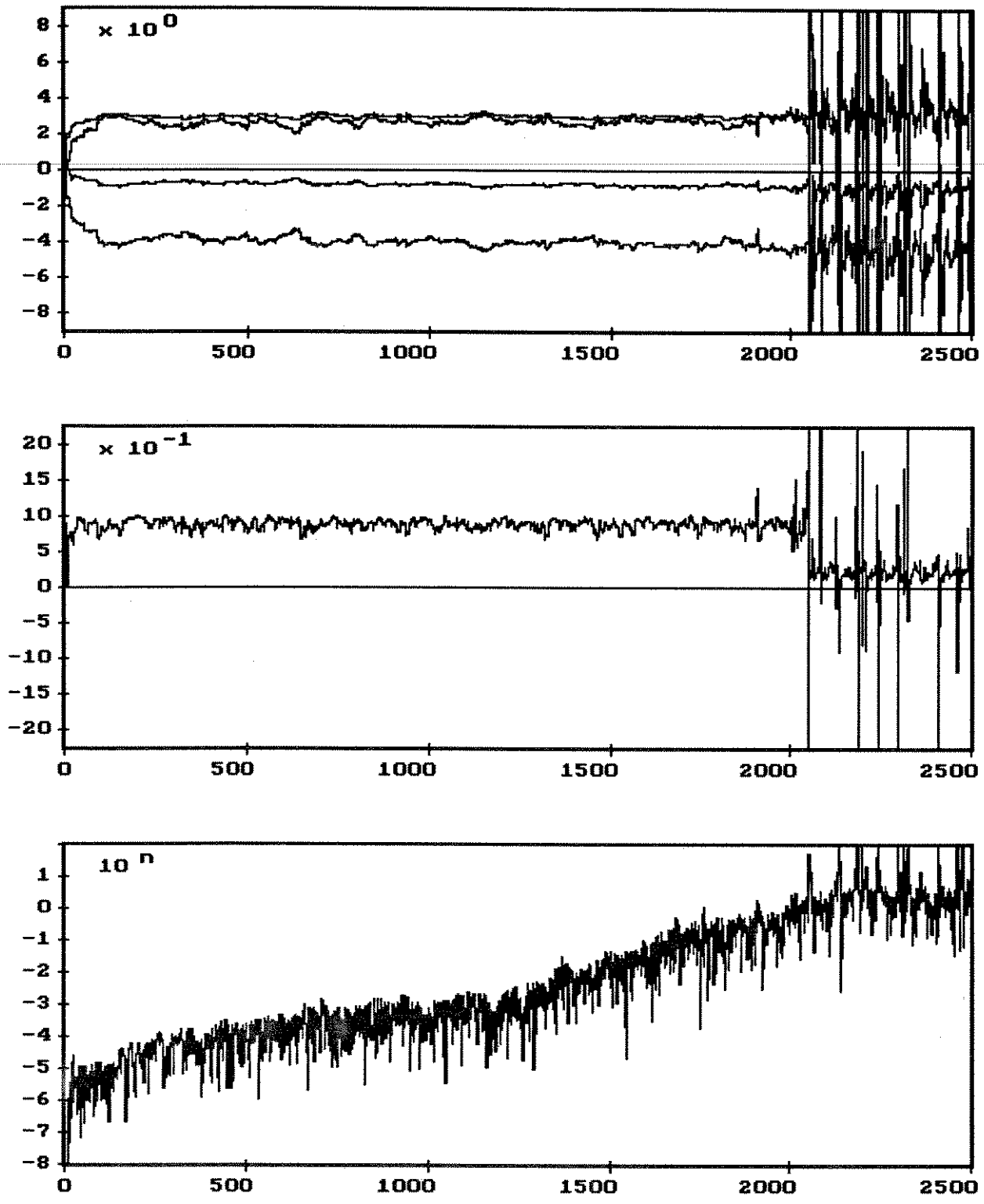


Figura 4.1 - Algoritmo FTF, com N = 4 e w = 0,97, excitado pelo mesmo processo AR descrito na Figura 3.6. Em **a** observa-se o comportamento dos coeficientes de predição progressiva; em **b**, a variável de verossimilhança, $\gamma(n)$, que extrapola sua faixa de validade com 1.715 iterações; e em **c**, o crescimento exponencial da variável $\eta(n)$.

O método da reinicialização encontra especial interesse nos processos de identificação de sistemas, onde o objetivo principal é estimar o vetor de coeficientes H(n). Como este vetor não participa do esquema de predição, seus elementos podem ser preservados no processo de reinicialização, tornando-o menos traumático.

Devido à súbita transição a que são submetidas as variáveis do algoritmo no momento da reinicialização, é necessário também prover um meio de evitar que os coeficientes de H(n) sejam afetados. Isto pode ser conseguido diminuindo-se a capacidade de adaptação do algoritmo no período de tempo que sucede à transição. Para tanto é suficiente uma escolha apropriada do parâmetro E_o. Esta questão foi discutida na Seção 3.4 para o caso dos preditores, em que os valores iniciais são nulos. No caso do filtro adaptativo, a influência de E_o na inicialização corresponde a modificar a função custo, dada em (1.17), por

$$E_N(n) = \sum_{i=1}^{n} w^{n-i} |d(i) - H^H(n)X(i)|^2 + E_o\, w^n\, [H(n) - H(0)]^H\, W\, [H(n) - H(0)] \tag{4.8}$$

onde W é a matriz definida em (3.127). O último termo de (4.8), que depende de E_o, é o responsável pela inércia introduzida na adaptação do filtro. Este termo desvanece com o tempo, permitindo que o algoritmo recupere gradualmente a capacidade de adaptação.

4.1.2 – O método de Lin

O método proposto por Lin [2], também chamado de Algoritmo FK de Covariância, apresenta uma abordagem diferente ao problema da seqüência inicial de entrada, dispensando a restrição $x(n) = 0$ para $n \leq 0$. Com isso, é possível manter as amostras presentes em X(n) no momento da reinicialização. A derivação feita em [2], do Algoritmo FK de Covariância, utiliza um desenvolvimento similar ao apresentado na Seção 2.3 para o Algoritmo FK. Iremos, alternativamente, derivá-lo considerando apenas as modificações em relação ao algoritmo original.

Os valores das variáveis deste algoritmo não correspondem exatamente aos fornecidos pelo Algoritmo FK, devido a diferença do vetor de entrada inicial X(0). Entretanto, por uma questão de simplicidade, desejamos manter os nomes e a notação utilizada anteriormente, mesmo para o caso do vetor de ganho, G(n), que agora deve atender a relação

$$\hat{R}(n)G(n) = X(n) \tag{4.9}$$

onde

$$\hat{R}(n) = R(n) + w^n\, X(0)X^H(0) \tag{4.10}$$

é a matriz de autocorrelação determinística alterada em virtude de X(0). Por outro lado, temos

$$\hat{P}(n)\, X(n) = G(n) \tag{4.11}$$

onde

$$\hat{P}(n) = [\hat{R}(n)]^{-1} \tag{4.12}$$

O algoritmo original não permite atualizar o vetor de ganho, pois ele foi derivado sob a hipótese de que $X(0) = 0_N$. Por outro lado, a execução de uma iteração completa do Algoritmo FK permite obter, no lugar de G(n), o vetor F(n), definido pela relação

$$[\hat{R}(n) - w^n X(0)X^H(0)] F(n) = X(n) \tag{4.13}$$

Igualando (4.9) e (4.13), obtemos

$$G(n) = [I_N - D(n)X^H(0)] F(n) \tag{4.14}$$

onde definimos

$$D(n) = w^n \hat{P}(n)X(0) \tag{4.15}$$

A equação (4.14) permite obter G(n) a partir de F(n) com uma complexidade numérica proporcional a 2N. O esforço computacional dependente de N^2, implícito na definição de D(n), pode ser evitado através de um cálculo recursivo. De (4.15) e (4.12) podemos escrever

$$\hat{R}(n)D(n) = w\,\hat{R}(n-1)D(n-1) \tag{4.16}$$

$$= [\hat{R}(n) - X(n)X^H(n)] D(n-1) \tag{4.17}$$

Pré-multiplicando ambos os termos de (4.17) por $\hat{P}(n)$, usando (4.11) e (4.14) e fazendo algumas manipulações, vem

$$D(n) = \frac{1}{1 - X^H(0)F(n)X^H(n)D(n-1)} [I_N - F(n)X^H(n)] D(n-1) \tag{4.18}$$

As equações (4.14) e (4.18) representam a complexidade adicional introduzida pelo método de Lin no Algoritmo FK. Elas demandam um esforço numérico proporcional a 4N, que somado ao esforço original resulta em uma complexidade proporcional a 14N. O algoritmo sugerido por Lin, assim como as disposições requeridas para as reinicializações, estão apresentados na Tabela 4.1.

O vetor D(n), em virtude de sua definição, tende a se anular com o crescimento de n, fazendo com que o Algoritmo da Covariância se reduza gradativamente ao Algoritmo FK.

Novamente, o parâmetro E_o representa um papel de fundamental importância, devendo seu valor ser arbitrado em função da inércia que se deseja imprimir ao processo de adaptação nos instantes que se sucedem às reinicializações.

As decontinuidades impostas pelas reinicializações, com reflexos na capacidade de rastreio, tornam o método pouco interessante, motivando a procura por soluções mais eficazes.

Tabela 4.1 - Algoritmo de Lin

disponível, do instante n-1

$$X(n-1),\ H(n-1),\ A(n-1),\ B(n-1),\ \hat{G}(n-1),\ D(n-1),\ E_a(n-1)$$

recebido no instante n

$$x(n),\ d(n)$$

atualização dos preditores

$$e_a(n) = x(n) - A^H(n-1)X(n-1)$$

$$A(n) = A(n-1) + e_a^*(n)\, G(n-1)$$

$$\varepsilon_a(n) = x(n) - A^H(n)X(n-1)$$

$$E_a(n) = w\, E_a(n-1) + e_a^*(n)\varepsilon_a(n)$$

$$\begin{bmatrix} M(n) \\ m(n) \end{bmatrix} = \begin{bmatrix} 0 \\ G(n-1) \end{bmatrix} + \frac{\varepsilon_a(n)}{E_a(n)} \begin{bmatrix} 1 \\ -A(n) \end{bmatrix}$$

$$e_b(n) = x(n-N) - B^H(n-1)X(n)$$

$$B(n) = \frac{1}{1 - m(n)e_b^*(n)} [B(n-1) + e_b^*(n)\, M(n)]$$

$$F(n) = M(n) + m(n)B(n)$$

$$D(n) = \frac{1}{1 - X^H(0)F(n)X^H(n)D(n-1)} [I_N - F(n)X^H(n)]\, D(n-1)$$

$$G(n) = [I_N - D(n)X^H(0)]\, F(n)$$

atualização do filtro adaptativo

$$e(n) = d(n) - H^H(n-1)X(n)$$

$$H(n) = H(n-1) + e^*(n)G(n)$$

reinicializar caso haja violação de

$$0 < 1 - e_b^*(n)m(n) \leq 1$$

condições iniciais

$$Ea(0) = E_o$$

$$A(0) = 0_N$$

$$B(0) = 0_N$$

$$D(0) = \frac{1}{E_o - X^H(0)W^{-1}(0)X^H(0)} W^{-1}(0)X(0)$$

$$G(0) = \frac{1}{E_o - X^H(0)W^{-1}(0)X^H(0)} W^{-1}(0)X(0)$$

4.2 – MÉTODOS QUE ALTERAM A SOLUÇÃO LS

Dentre os métodos de estabilização que alteram a solução LS, podemos considerar primeiramente o método da adição de ruído branco que, na verdade, não altera a estrutura do algoritmo, mas apenas os sinais envolvidos.

4.2.1 – O método da adição de ruído branco

A proposta de se adicionar um ruído branco ao sinal de entrada, x(n), se aplica tão somente ao caso de sinais de entrada mal comportados. Ela representa uma tentativa de melhorar o condicionamento da matriz de autocorrelação. Seria inútil, além de desnecessário, aplicar este método a um sinal cujas amostras já fossem descorrelacionadas.

Assinalamos na Seção 3.1 que a vida de um algoritmo rápido varia em cerca de três vezes em função das características do sinal de entrada. Esta afirmativa, obviamente, é válida para o caso de se utilizar um valor razoável para o fator de esquecimento. Como observado em [1], o método da adição de ruído branco somente surte o desejado efeito de estabilização quando w é muito próximo de 1. Neste caso, o crescimento do ruído de quantização devido ao fator de esquecimento fica reduzido, ao ponto do comportamento do sinal de entrada se tornar o fator preponderante na estabilidade numérica do algoritmo.

A adição de ruído branco não altera muito a solução, principalmente se a potência do ruído for devidamente pequena. O efeito de polarização produzido nos preditores e no filtro, por exemplo, equivale aproximadamente a substituir a matriz de autocorrelação determinística pela matriz

$$\hat{R}(n) = R(n) + \frac{\sigma_\delta^2}{1-w} I_N \tag{4.19}$$

onde σ_δ^2 é a variância do ruído adicionado.

O método da adição de ruído branco não constitui uma solução efetiva para o problema da instabilidade, pois atua unicamente na característica do sinal de entrada. Além disso, de forma geral, os casos que requerem um fator de esquecimento muito próximo de 1 prescindem da utilização do critério LS. Já vimos que a argumentação para o emprego deste critério reside exatamente na sua capacidade de adaptação.

4.2.2 – O método da constante de estabilização

Outro método, já mencionado anteriormente, e que também representa um paliativo para evitar a divergência dos algoritmos rápidos, é o da chamada constante de estabilização. Este método consiste em se trocar a forma de atualização da energia do preditor progressivo, $E_a(n)$, pela expressão

$$E_a(n) = E_a(n-1) + \varepsilon_a^*(n)e_a(n) + \Delta \tag{4.20}$$

onde Δ é um escalar positivo de valor bem pequeno. Este artifício evita o anulamento da variável $E_a(n)$, impedindo a ocorrência de uma divisão por zero, ou por valores muito pequenos.

Em [3] é analisada a influência da constante de estabilização no processo de adaptação e mostrado que ela tem o efeito de aumentar a constante de tempo, como se o fator de esquecimento fosse de fato um pouco maior.

Em nossa experiência com variáveis em ponto flutuante, entretanto, notamos que a utilização desta constante não melhora a característica de estabilidade numérica do algoritmo senão nos casos de sinais perfeitamente predizíveis, onde existe uma tendência ao anulamento da variável $E_a(n)$. Mesmo nestes casos, os incrementos adicionados necessitam ter valores extremamente pequenos, sob pena de acentuar a instabilidade.

Sob uma ótica comparativa, este método padece dos mesmos problemas que o da adição de ruído branco. Primeiro, por que ele somente atua no sentido de corrigir os efeitos do mau comportamento do sinal de entrada. Segundo, por que ele se mostra ineficiente para estabilizar o algoritmo quando são utilizados valores plausíveis para o fator de esquecimento.

Talvez o método que melhor represente a tentativa de estabilizar os algoritmos rápidos, relegando a exatidão da solução LS, seja o sugerido por Fabre e Guegen.

4.2.3 – O método de Fabre e Guegen

Este método, apresentado em [4], tem uma motivação interessante. Segundo seus autores, os zeros dos preditores nos algoritmos rápidos, quando excitados por sinais estacionários, sofrem um deslocamento radial, caminhando em direção contrária à origem do plano Z. Ou seja, sofrem um aumento em módulo, mantendo seus argumentos aproximadamente constantes. Esse efeito, atribuído à acumulação do ruído de quantização nas variáveis do algoritmo, seria o responsável pela instabilidade. A proposta de Fabre e Guegen consiste em aplicar periodicamente uma transformação aos preditores, de modo a forçar os zeros a se deslocarem em direção à origem.

Como pode ser comprovado, as transformações

$$\hat{A} = \Psi A \tag{4.21a}$$

$$\hat{B} = \Psi B \tag{4.21b}$$

onde

$$\Psi = \mathrm{diag}(\rho, \rho^2, \rho^3, \ldots, \rho^N) \tag{4.22}$$

provocam um deslocamento radial dos zeros da função de transferência dos preditores. O método descrito em [4] determina que se faça periodicamente as transformações (4.21), com um valor de ρ menor e próximo de 1, para compensar o mencionado afastamento radial dos zeros. A complexidade adicionada por este método é proporcional a 2N e somente afeta as iterações nas quais se dão as correções dos preditores. Ainda segundo [4], estas correções devem ser feitas com uma periodicidade de aproximadamente 20 iterações.

Com o intuito de comprovar a eficiência do método, fizemos diversas simulações, aplicando-o ao Algoritmo FTF. Utilizamos variáveis em ponto flutuante, e testamos uma ampla faixa de valores para o parâmetro ρ, assim como para o período de correção dos preditores. Observamos que em todos os casos o método se mostrou falho, não trazendo qualquer benefício em termos de estabilidade numérica. De forma geral, a vida do algoritmo resultou diminuída com a aplicação do método. Somente nos casos em que foi utilizado ρ muito próximo de 1 e períodos grandes para a correção dos preditores é que observamos um leve prolongamento na vida do algoritmo. Isto, no entanto, é pouco para que o método seja considerado um processo estabilizante.

A comprovação da ineficiência do método apenas ratifica o que foi dito no início deste capítulo sobre a sensibilidade dos algoritmos rápidos. *Qualquer modificação feita na estrutura do algoritmo, que não leve em conta a exatidão da solução LS, contribui para torná-lo numericamente mais instável.*

Acreditamos que se as transformações (4.21) forem aplicadas aos preditores acompanhadas de correções nas demais variáveis, para manter o algoritmo dentro do contexto LS, o método deve funcionar. Entretanto, a complexidade matemática necessária para determinar a realização de tais correções seria muito grande, inviabilizando a implementação.

Ao que parece, o problema da estabilização dos algoritmos rápidos passa pela manutenção das variáveis dentro do critério LS. Os métodos da constante de estabilização e de Fabre e Guegen, são exemplos clássicos de tentativas que comprometem esta regra e apresentam resultados insatisfatórios.

Passemos então à discussão dos métodos que não alteram a solução LS.

4.3 – MÉTODOS QUE PRESERVAM A SOLUÇÃO LS

Nesta seção vamos descrever várias técnicas para a estabilização numérica dos algoritmos rápidos. Estas técnicas, que apresentam a desejável característica de manter a estrutura e a solução original, não constituem a solução definitiva para o problema, pois não existe uma forma de se provar a estabilidade proclamada por elas. Além disso, é notório que elas falham para valores pequenos de w, pondo em dúvida a segurança de desempenho para o caso de valores razoáveis. De qualquer forma, estas técnicas representam um grande avanço na viabilização de sistemas práticos com algoritmos rápidos.

4.3.1 – O método de Botto e Moustakides

O método proposto por Botto e Moustakides constitui uma solução atraente para o problema da instabilidade numérica dos algoritmos rápidos. Este método se aplica melhor aos algoritmos FAEST e FTF, que são computacionalmente mais eficientes. Seu princípio de funcionamento utiliza alguns conceitos da teoria de controle, em que um sinal de erro é utilizado para corrigir continuamente as variáveis de estado do sistema. Iremos apresentar uma versão

simplificada desse método, onde somente as variáveis integrantes da malha do preditor regressivo são corrigidas. A versão original, apresentada em [5], efetua correções também no preditor progressivo. Como vimos na Seção 3.3, a malha formada pelo preditor progressivo não é afetada pelo ruído de quantização, de forma que sua correção traz uma complexidade desnecessária ao algoritmo. Este fato foi também observado experimentalmente por Cioffi em [6], onde é apresentado um desenvolvimento similar ao que faremos.

O sinal de erro acima mencionado, que é utilizado para corrigir as variáveis de estado do algoritmo, é dado pela diferença entre as duas diferentes formas de se calcular o erro de predição regressiva a priori

$$e_b^f(n) = x(n-N) - B^H(n-1)X(n) \tag{4.23}$$

e

$$e_b^s(n) = w\, E_b(n-1)\, \tilde{m}(n) \tag{4.24}$$

onde os superíndices ^{f} e ^{s} indicam se o cálculo do erro resultou de uma operação de filtragem ou de operações com variáveis escalares. Como estas expressões produzem o erro de predição de forma independente, temos que o valor absoluto médio do parâmetro

$$\xi(n) = x(n-N) - B^H(n-1)X(n) - w\, E_b(n-1)\, \tilde{m}(n) \tag{4.25}$$

representa uma medida do quanto o algoritmo encontra-se fora do critério LS, ou da quantidade de ruído acumulado. Idealmente, para o caso de precisão infinita, $\xi(n)$ deveria ser nulo. A ocorrência de valores diferentes de zero deve-se unicamente aos efeitos numéricos. A parte **a** da Figura 4.2 ilustra o crescimento do valor absoluto desta variável para o Algoritmo FTF. Na parte **b** está ilustrada esta mesma variável após a estabilização do algoritmo pelo método de Botto e Moustakides.

No método de Botto e Moustakides é desejável não apenas efetuar a minimização referente ao critério LS, mas também minimizar, em cada iteração, a função custo

$$\Omega(n) = w\, [\overline{B}(n-1) - B(n-1)]^H\, R(n-1)\, [\overline{B}(n-1) - B(n-1)] + \rho\, |\overline{\xi}(n)|^2 \tag{4.26}$$

onde $\overline{B}(n)$ é o valor corrigido do preditor regressivo, $\overline{\xi}(n)$ é o valor corrigido de $\xi(n)$, e ρ é uma constante cujo valor deve ser arbitrado de forma a proporcionar o melhor funcionamento do método. Valores em torno de 1 atendem bem este requisito, sendo mesmo justificável a escolha do número 1 como forma de diminuir o esforço numérico. O significado do primeiro termo de $\Omega(n)$ na minimização pode ser melhor entendido notando-se que

$$[\overline{B}(n-1) - B(n-1)]^H\, R(n-1)\, [\overline{B}(n-1) - B(n-1)] = \|\varepsilon_{bM}(n-1) - \overline{\varepsilon}_{bM}(n-1)\|^2 \tag{4.27}$$

onde $\varepsilon_{bM}(n)$ foi definido na Seção 2.7 como

$$\varepsilon_{bM}(n) = X_M(n) - X_{MN}(n)B(n) \tag{4.28}$$

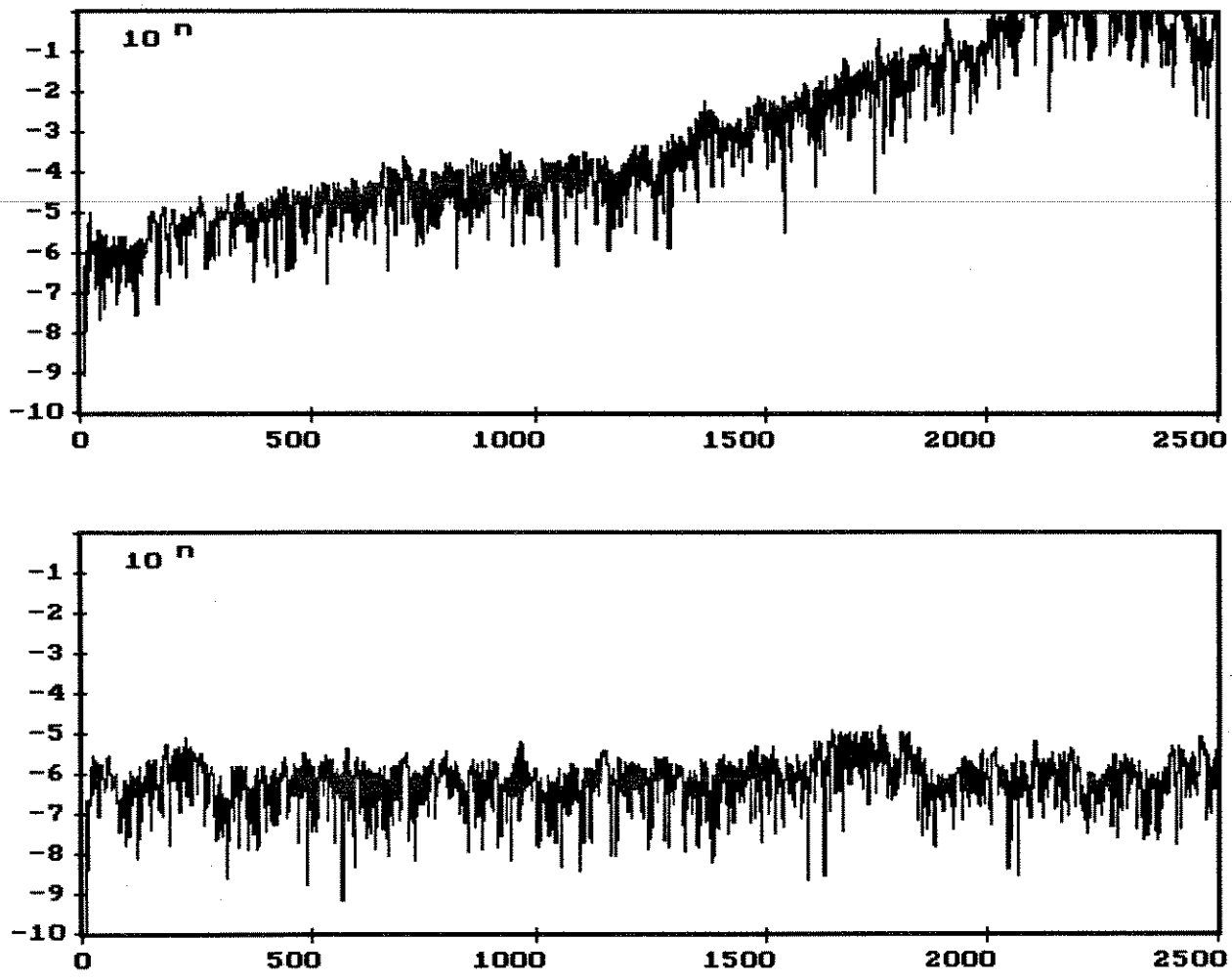


Figura 4.2 - Valor absoluto da variável $\xi(n)$ no Algoritmo FTF com N = 4 e w = 0,97. Em **a** está representado o comportamento desta variável para o processo da Figura 4.1. Em **b**, é apresentada a mesma simulação para o algoritmo numericamente estabilizado de Botto e Moustakides com $\rho = 1$.

e $\bar{\varepsilon}_{bM}(n)$ é dado por

$$\bar{\varepsilon}_{bM}(n) = X_M(n) - X_{MN}(n)\bar{B}(n) \tag{4.29}$$

Derivando $\Omega(n)$ em relação a $\bar{B}(n-1)$, igualando o vetor resultante a 0_N e usando (2.72), obtemos

$$\bar{B}(n-1) = B(n-1) + \rho\, \bar{\xi}^*(n)\, G(n) \tag{4.30}$$

Substituindo este preditor corrigido na definição de $e_b(n)$, vem

$$\bar{e}_b(n) = e_b(n) - \rho\, [\frac{1}{\gamma(n)} - 1]\, \xi(n) \tag{4.31}$$

Tabela 4.2 - Algoritmo de Botto e Moustakides Simplificado

disponível, do instante n-1

$$X(n-1),\ H(n-1),\ A(n-1),\ B(n-1),\ \bar{G}(n-1),\ \gamma(n-1),\ E_a(n-1),\ E_b(n-1)$$

recebido no instante n

$$x(n),\ d(n)$$

atualização dos preditores

$$e_a(n) = x(n) - A^H(n-1)X(n-1)$$

$$\begin{bmatrix} \bar{M}(n) \\ \tilde{m}(n) \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{G}(n-1) \end{bmatrix} + \frac{e_a(n)}{w\, E_a(n-1)} \begin{bmatrix} 1 \\ -A(n-1) \end{bmatrix}$$

$$\varepsilon_a(n) = \gamma(n-1)\, e_a(n)$$

$$A(n) = A(n-1) + \varepsilon_a^*(n)\, \bar{G}(n-1)$$

$$E_a(n) = w\, E_a(n-1) + \varepsilon_a^*(n) e_a(n)$$

$$e_b^f(n) = x(n-N) - B^H(n-1)X(n)$$

$$e_b^s(n) = w\, E_b(n-1)\, \tilde{m}(n)$$

$$\xi(n) = e_b^f(n) - e_b^s(n)$$

$$\gamma_1(n) = \frac{w\, E_a(n-1)}{E_a(n)}\, \gamma(n-1)$$

$$\gamma(n) = \frac{\gamma_1(n)}{1 - \gamma_1(n)\tilde{m}^*(n) e_b^f(n)}$$

$$\bar{\xi}(n) = \frac{1}{1 + \rho\,[\frac{1}{\gamma(n)} - 1]}\, \xi(n)$$

$$\bar{e}_b(n) = e_b^f(n) - \rho\,[\frac{1}{\gamma(n)} - 1]\, \bar{\xi}(n)$$

$$\bar{B}(n-1) = \frac{1}{1 - \rho\, \tilde{m}(n)\bar{\xi}^*(n)}\, [B(n-1) + \rho\, \bar{\xi}^*(n)\, \bar{M}(n)]$$

$$\bar{G}(n) = \bar{M}(n) + \tilde{m}(n)\, \bar{B}(n-1)$$

$$\varepsilon_b(n) = \gamma(n)\bar{e}_b(n)$$

$$B(n) = \bar{B}(n-1) + \varepsilon_b^*(n)\bar{G}(n)$$

$$E_b(n) = w\, E_b(n-1) + \varepsilon_b^*(n) e_b(n)$$

atualização do filtro adaptativo

$$e(n) = d(n) - H^H(n-1)X(n)$$

$$\varepsilon(n) = \gamma(n) e(n)$$

$$H(n) = H(n-1) + \varepsilon^*(n)\bar{G}(n)$$

onde usamos a expressão

$$\tilde{G}^H(n)X(n) = [\frac{1}{\gamma(n)} - 1] \tag{4.32}$$

que é derivada diretamente das equações (2.58) e (2.68). De forma semelhante, substituindo $\bar{B}(n-1)$ em (4.25), obtemos

$$\bar{\xi}(n) = \frac{1}{1 + \rho\,[\frac{1}{\gamma(n)} - 1]}\,\xi(n) \tag{4.33}$$

A equação (4.30) corrige B(n-1) através do vetor de ganho dual, $\bar{G}(n)$, o qual também depende de B(n-1). Dessa forma, é conveniente fazer

$$\bar{G}(n) = \bar{M}(n) + \tilde{m}(n)\,\bar{B}(n-1) \tag{4.34}$$

Substituindo esta relação em (4.30) obtemos, para a correção do preditor regressivo, a expressão

$$\bar{B}(n-1) = \frac{1}{1 - \rho\,\tilde{m}(n)\bar{\xi}^*(n)}\,[B(n-1) + \rho\,\bar{\xi}^*(n)\,\bar{M}(n)] \tag{4.35}$$

Na Tabela 4.2 apresentamos o Algoritmo de Botto e Moustakides simplificado, que tem uma complexidade de 10N + 20 multiplicações, 9N + 8 somas e 5 divisões, constituindo uma solução interessante para o problema da instabilidade. No caso de precisão infinita, quando os erros de truncamento não estão presentes, temos que $\xi(n)$ é sempre nulo, e o algoritmo fica reduzido ao Algoritmo FTF.

Com o intuito de comprovar a eficiência do método, fizemos diversas simulações com o algoritmo da Tabela 4.2. Para valores razoáveis de w ele se mostrou bastante estável, chegando a suportar milhões de iterações sem que qualquer tendência de instabilidade fosse detectada. Para valores abaixo de 0,95, no entanto, o algoritmo diverge.

Os valores de w que se situam abaixo de 0,95 representam para a estabilidade dos Algoritmos Rápidos um desafio insuperável. Apesar de não evitar a divergência das variáveis quando operando nesta região, o método de Botto e Moustakides, assim como outros métodos de estabilização que veremos a seguir, conseguem prolongar substancialmente a vida dos processos.

No intuito de aproveitar o algoritmo também nas regiões de desempenho duvidoso, ou mesmo como medida de precaução, é proposto que ele seja usado da mesma forma que o descrito na Seção 4.1, fazendo-se reinicializações sempre que for apontado um determinado acúmulo de ruído. A variável de alerta, sugerida em [5], para este propósito é o módulo ao quadrado do sinal de erro. Ou seja, o algoritmo deve ser reinicializado sempre que

$$|\xi(n)|^2 > t \tag{4.36}$$

onde t é um limiar de decisão previamente estabelecido.

4.3.2 – O método de Slock e Kailath

O método de estabilização que descreveremos a seguir foi apresentado em [7] por Slock e Kailath e se aplica ao Algoritmo FAEST. Assim como no método de Botto e Moustakides, o princípio de funcionamento deste método também tira proveito das diferentes formas de se calcular uma mesma variável. Aqui, não apenas a variável $e_b(n)$, mas também as variáveis escalares $\tilde{m}(n)$ e $\zeta(n)$ são calculadas de formas alternativas. Para o caso de $e_b(n)$ temos novamente os resultados escalar e convolucional do item anterior: $e_b^s(n)$ e $e_b^f(n)$. Para o caso de $\tilde{m}(n)$, temos

$$\tilde{m}^s(n) = \tilde{g}_N(n-1) - \frac{e_a(n)}{w\, E_a(n-1)}\, a_N(n-1) \tag{4.37}$$

que é resultado da partição de $\tilde{G}_{N+1}(n)$, e também

$$\tilde{m}^f(n) = \frac{e_b^f(n)}{w\, E_b(n-1)} \tag{4.38}$$

que resulta diretamente de (2.79). Para o caso de $\zeta(n)$, ou $\gamma^{-1}(n)$, temos as três seguintes possibilidades

$$\zeta^s(n) = \zeta(n-1) + \frac{|e_a(n)|^2}{w\, E_a(n-1)} - \tilde{m}^*(n)\, e_b^{(5)}(n) \tag{4.39}$$

$$\zeta^f(n) = 1 + \tilde{G}(n)X(n) \tag{4.40}$$

e

$$\zeta^a(n) = w^{-N}\, \frac{E_a(n)}{E_b(n)} \tag{4.41}$$

O valor corrigido para as variáveis é calculado por meio de uma média ponderada entre os valores alternativos. Assim

$$e_b^{(i)}(n) = k_i\, e_b^f(n) + [1 - k_i]\, e_b^s(n) \qquad i = 1, 2, 5 \tag{4.42}$$

corresponde a três diferentes valores corrigidos para o erro de predição regressiva a priori, cada qual originado em função de um diferente valor da constante k_i, i = 1, 2, 5. De forma análoga, temos as correções

$$\tilde{m}(n) = k_4\, \tilde{m}^f(n) + [1 - k_4]\, \tilde{m}^s(n) \tag{4.43}$$

e

$$\zeta(n) = k_3\, \zeta^f(n) + k_6\, \zeta^s(n) + [1 - k_3 - k_6]\, \zeta^a(n) \tag{4.44}$$

Tabela 4.3 - Algoritmo de Slock e Kailath

disponível, do instante n-1

$$X(n-1),\ H(n-1),\ A(n-1),\ B(n-1),\ \bar{G}(n-1),\ \zeta(n-1),\ E_a(n-1),\ E_b(n-1)$$

recebido no instante n

$$x(n),\ d(n)$$

atualização dos preditores

$$e_a(n) = x(n) - A^H(n-1)X(n-1)$$

$$\begin{bmatrix} \bar{M}(n) \\ \tilde{m}^s(n) \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{G}(n-1) \end{bmatrix} + w^{-1}\, e_a(n)\, E_a^{-1}(n-1) \begin{bmatrix} 1 \\ -A(n-1) \end{bmatrix}$$

$$\varepsilon_a(n) = \frac{e_a(n)}{\zeta(n-1)}$$

$$A(n) = A(n-1) + \varepsilon_a^*(n)\, \bar{G}(n-1)$$

$$\zeta_1(n) = \zeta(n-1) + e_a^*(n)\, \bar{M}_0(n)$$

$$E_a^{-1}(n) = w^{-1}\, E_a^{-1}(n-1) - \frac{|\bar{M}_0(n)|^2}{\zeta_1(n)}$$

$$e_b^f(n) = x(n-N) - B^H(n-1)X(n)$$

$$e_b^s(n) = w\, E_b(n-1)\, \tilde{m}^s(n)$$

$$e_b^{(i)}(n) = k_i\, e_b^f(n) + [1 - k_i]\, e_b^s(n) \qquad i = 1, 2, 5$$

$$\tilde{m}^f(n) = \frac{e_b^f(n)}{w\, E_b(n-1)}$$

$$\tilde{m}(n) = k_4\, \tilde{m}^f(n) + [1-k_4]\, \tilde{m}^s(n)$$

$$\bar{G}(n) = \bar{M}(n) + \tilde{m}(n)B(n-1)$$

$$\zeta^s(n) = \zeta_1(n) - \tilde{m}^*(n) e_b^{(5)}(n)$$

$$\zeta^f(n) = 1 + \bar{G}(n)X(n)$$

$$B(n) = B(n-1) + \frac{e_b^{(1)*}(n)}{\zeta(n)}\, \bar{G}(n)$$

$$E_b(n) = w\, E_b(n-1) + \frac{|e_b^{(2)}(n)|^2}{\zeta(n)}$$

$$\zeta^a(n) = w^{-N}\, \frac{E_a(n)}{E_b(n)}$$

$$\zeta(n) = k_3\, \zeta^f(n) + k_6\, \zeta^s(n) + [1 - k_3 - k_6]\, \zeta^a(n)$$

atualização do filtro adaptativo

$$e(n) = d(n) - H^H(n-1)X(n)$$

$$H(n) = H(n-1) + \frac{e^*(n)}{\zeta(n)}\, \bar{G}(n)$$

A multiplicidade de valores definidos para a correção de $e_b(n)$ confere ao método uma maior flexibilidade. Como esta variável é utilizada em vários pontos do algoritmo, cada ponto possui um grau de correção específico.

É possível notar uma certa similaridade no funcionamento deste método com o de Botto e Moustakides. Para isso, notemos que, usando a equação (4.25), é possível reescrever a relação (4.42) como

$$e_b^{(i)}(n) = e_b^s(n) + k_i\,\xi(n) \qquad i = 1, 2, 5 \qquad (4.45)$$

Visto dessa forma, o mecanismo de controle da instabilidade envolve também a utilização de sinais de erro. A diferença é que, neste caso, os sinais de erro são aplicados em outros pontos do algoritmo, que não aqueles do Algoritmo de Botto e Moustakides. Mais precisamente, neste método, a adição dos sinais de erro é feita nas próprias variáveis com as quais eles estão relacionados.

É oportuno ressaltar que dentre os vários sinais utilizados para avaliar a condição de erro, o que melhor responde pelo estado de anormalidade do algoritmo é o sinal $\xi(n)$. Isto acontece porque este é o sinal mais diretamente relacionado com a malha de atualização do preditor regressivo.

A Tabela 4.3 descreve o Algoritmo de Slock e Kailath, cujo esforço computacional é de 9N + 12 somas, 9N + 26 multiplicações e 3 divisões. A despeito desta menor complexidade em relação ao Algoritmo de Botto e Moustakides, atentamos para a presença das constantes de ponderação, que representam, a nosso ver, uma solução deselegante. Em [9] é feito um estudo para a definição destas constantes em função de parâmetros conhecidos. Pode-se notar também que para o caso de $k_i = 0$, $i = 1, 2, \ldots, 6$, o algoritmo fica reduzido ao Algoritmo FAEST 7N.

4.3.3 – O método de Benallal e Gilloire

O método de Benallal e Gilloire utiliza uma abordagem muito parecida com a de Slock e Kailath. A diferença mais significativa é que este método atua com base unicamente na variável de controle de erro $\xi(n)$, desprezando as demais variáveis. Além disso, ele se aplica ao Algoritmo FTF, e não ao FAEST.

Aplicando a equação (4.45) com três diferentes valores para a constante k_i é possível obter três diferentes valores corrigidos para o erro de predição a priori; que são específicos para cada uma das variáveis que o utilizam: $B(n)$, $E_b(n)$ e $\gamma(n)$.

O algoritmo resultante está apresentado na Tabela 4.4. Este método de estabilização possui a virtude de não acrescentar muita complexidade numérica ao processo, uma vez que mantém a relação proporcional a 8N correspondente ao algoritmo original. Como ponto negativo, nota-se novamente a presença das constantes de ponderação, que precisam ser arbitradas convenientemente para o bom funcionamento do método. Dos estudos apresentados em [10], parece que uma escolha recomendável para estas constantes é $k_i = 1$, $i = 1, 2, 3$. Vale notar que fazendo $k_i = 0$, $i = 1, 2, 3$, temos que o algoritmo reduz-se ao FTF-8N, ao passo que fazendo $k_i = -1$, $i = 1, 2, 3$, temos o Algoritmo FTF-7N.

Tabela 4.4 - Algoritmo de Benallal e Gilloire

disponível, do instante n-1

$$X(n-1),\ H(n-1),\ A(n-1),\ B(n-1),\ \bar{G}(n-1),\ \gamma(n-1),\ E_a(n-1),\ E_b(n-1)$$

recebido no instante n

$$x(n),\ d(n)$$

atualização dos preditores

$$e_a(n) = x(n) - A^H(n-1)X(n-1)$$

$$\begin{bmatrix} \bar{M}(n) \\ \tilde{m}(n) \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{G}(n-1) \end{bmatrix} + \frac{e_a(n)}{w\, E_a(n-1)} \begin{bmatrix} 1 \\ -A(n-1) \end{bmatrix}$$

$$\varepsilon_a(n) = \gamma(n-1)\, e_a(n)$$

$$A(n) = A(n-1) + \varepsilon_a^*(n)\, \bar{G}(n-1)$$

$$E_a(n) = w\, E_a(n-1) + \varepsilon_a^*(n) e_a(n)$$

$$e_b^f(n) = x(n-N) - B^H(n-1)X(n)$$

$$e_b^s(n) = w\, E_b(n-1)\, \tilde{m}(n)$$

$$\xi(n) = e_b^f(n) - e_b^s(n)$$

$$e_b^{(i)}(n) = e_b(n) + k_i\, \xi(n) \qquad i = 1, 2, 3$$

$$\gamma_1(n) = \frac{w\, E_a(n-1)}{E_a(n)}\ \gamma(n-1)$$

$$\gamma(n) = \frac{\gamma_1(n)}{1 - \gamma_1(n)\tilde{m}^*(n)e_b^{(1)}(n)}$$

$$\bar{G}(n) = \bar{M}(n) + \tilde{m}(n)\, B(n-1)$$

$$B(n) = B(n-1) + \gamma(n)\, e_b^{(2)*}(n)\, \bar{G}(n)$$

$$E_b(n) = w\, E_b(n-1) + \gamma(n)\, |e_b^{(3)}(n)|^2$$

atualização do filtro adaptativo

$$e(n) = d(n) - H^H(n-1)X(n)$$

$$\varepsilon(n) = \gamma(n)e(n)$$

$$H(n) = H(n-1) + \varepsilon^*(n)\bar{G}(n)$$

A simulação deste algoritmo, utilizando sinais estacionários e variáveis em ponto flutuante, revelou um desempenho comparável ao do Algoritmo de Botto e Moustakides. Foram observadas milhões de iterações sem que se notasse qualquer anormalidade no comportamento das variáveis. Também, da mesma forma que no Algoritmo de Botto e Moustakides, o método de estabilização falha para valores de w menores que 0,95.

4.3.4 – Algoritmo de Kalman Rápido Estabilizado

Como visto, os métodos de estabilização que preservam a solução LS utilizam-se de formas alternativas para o cálculo de determinada variável. O sinal gerado pela diferença destas variáveis é o responsável pelas correções de erro no algoritmo. O Algoritmo FK, por não utilizar todos os erros de predição, parece não apresentar muitas formas alternativas para o cálculo de suas variáveis. No entanto, em [10], é notado que existe uma forma opcional para a obtenção do erro de predição regressiva a priori, a qual pode ser usada para gerar um sinal de controle de erro para este algoritmo.

Tomando a relação que determina $e_b(n)$ nos algoritmos rápidos computacionalmente mais eficientes

$$e_b(n) = w\,\tilde{m}(n)\,E_b(n-1) \tag{4.46}$$

e substituindo as equações

$$\tilde{m}(n) = \frac{m(n)}{\gamma_1(n)} \tag{4.47}$$

$$\gamma_{1(n)} = \gamma(n-1)\,\frac{w\,E_a(n-1)}{E_a(n)} \tag{4.48}$$

e

$$\gamma(n)\,\frac{E_a(n)}{E_b(n)} = w^N \tag{4.49}$$

que foram obtidas no Capítulo 2, temos,

$$e_b(n) = w^{-N}\,E_a(n)\,m(n) \tag{4.50}$$

Esta equação fornece $e_b(n)$ a partir de variáveis escalares existentes no Algoritmo FK, de forma que permite obter a variável de controle, $\xi(n)$, com um número mínimo de operações matemáticas. A forma de aplicar o sinal de erro ao algoritmo é semelhante à empregada no Algoritmo de Benallal e Gilloire, com a diferença que, neste caso, somente o preditor regressivo é corrigido. Dessa forma

$$B(n) = B(n-1) + [e_b^*(n) + \rho\,\xi(n)]\,G(n) \tag{4.51}$$

onde ρ é uma constante que regula o grau de realimentação imprimido.

A implementação do Algoritmo FK Estabilizado, tal como sugerida em [10], é apresentada na Tabela 4.5. A demanda computacional é de 9N + 3 somas, 10N + 6 multiplicações e 2 divisões, o que representa um acréscimo de apenas 2 somas e 3 multiplicações ao algoritmo original.

Tabela 4.5 - Algoritmo FK Estabilizado

disponível, do instante n-1

$$X(n-1),\ H(n-1),\ A(n-1),\ B(n-1),\ G(n-1),\ E_a(n-1)$$

recebido no instante n

$$x(n),\ d(n)$$

atualização dos preditores

$$e_a(n) = x(n) - A^H(n-1)X(n-1)$$

$$A(n) = A(n-1) + e_a^*(n)\, G(n-1)$$

$$\varepsilon_a(n) = x(n) - A^H(n)X(n-1)$$

$$E_a(n) = w\, E_a(n-1) + \varepsilon_a^*(n)e_a(n)$$

$$\begin{bmatrix} M(n) \\ m(n) \end{bmatrix} = \begin{bmatrix} 0 \\ G(n-1) \end{bmatrix} + \frac{\varepsilon_a(n)}{E_a(n)} \begin{bmatrix} 1 \\ -A(n) \end{bmatrix}$$

$$e_b(n) = x(n-N) - B^H(n-1)X(n)$$

$$G(n) = \frac{1}{1 - m(n)e_b^*(n)} [M(n) + m(n)\, B(n-1)]$$

$$\xi(n) = e_b(n) - w^{-N} E_a(n)\, m(n)$$

$$B(n) = B(n-1) + [e_b^*(n) + \rho\, \xi(n)]\, G(n)$$

$$E_b(n) = w\, E_b(n-1) + \varepsilon_b^*(n)e_b(n)$$

atualização do filtro adaptativo

$$e(n) = d(n) - H^H(n-1)X(n)$$

$$H(n) = H(n-1) + e^*(n)\, G(n)$$

Infelizmente, a pouca complexidade adicionada por este método não representa uma vantagem, dado que a complexidade computacional do algoritmo original é grande (cerca de 2N operações a mais que os algoritmos mais eficientes). Além disso, procurando constatar a eficácia do método, executamos diversas simulações utilizando este algoritmo, e os resultados mostraram-se muito insatisfatórios. Apesar de aumentar levemente a estabilidade do Algoritmo FK, ele não produz o desejável efeito de estabilização. Em praticamente todas as situações testadas o método se mostrou deficiente, mesmo usando diferentes valores para o parâmetro ρ.

4.4 – UM NOVO MÉTODO DE ESTABILIZAÇÃO

Vamos agora apresentar um novo método para a estabilização dos algoritmos rápidos. Este método foi desenvolvido tendo como motivação a busca de um algoritmo que apresentasse um bom desempenho de funcionamento mantendo em um nível mínimo o esforço computacional.

O resultado foi um algoritmo com uma complexidade proporcional a 8N, cujo desempenho é comparável aos demais métodos existentes

O funcionamento do método, que é aplicável ao Algoritmo FTF, apresenta alguma similaridade com o de Botto e Moustakides. Em seu desenvolvimento foram observados os seguintes pontos:

1 - O método deve aplicar-se ao algoritmo FTF ou FAEST, devido à menor complexidade computacional destes algoritmos.

2 - A atuação do método deve concentrar-se na malha que atualiza o preditor regressivo, pois o processo de instabilidade está relacionado com ela.

3 - Todas as variáveis diretamente ligadas a atualização do preditor regressivo, B(n), precisam ser corrigidas uma vez em cada iteração.

4 - É indiferente que determinada variável seja corrigida no início, no meio ou no final do algoritmo. Assim por exemplo, a correção de B(n) pode ocorrer no final de uma dada iteração, de uma forma totalmente equivalente a B(n-1) ser corrigida no início da iteração seguinte.

5 - A correção das variáveis vetoriais não deve acrescentar uma complexidade excessiva ao método, devendo-se evitar somas ou multiplicações vetoriais, que incrementam de N o esforço computacional.

Considerando estes pontos, procuramos reunir as correções das variáveis no final do algoritmo. Dessa forma, todas as passagens se mantém praticamente inalteradas e, ao final de cada iteração, são feitas as correções. Além disso, observamos que existe a necessidade de corrigir unicamente o vetor B(n) e a variável $E_b(n)$, que se enquadram no contexto estabelecido acima, e são transmitidas de uma iteração à outra no Algoritmo FTF. A rigor a variável $\gamma(n)$ também deveria ser corrigida. Entretanto, inúmeras tentativas nesse sentido apresentaram resultados insatisfatórios, não melhorando a estabilidade do algoritmo.

Conforme já discutido anteriormente, a diferença entre as formas de calcular os erros de predição regressiva produzem o sinal que melhor responde às necessidades de correção existentes no Algoritmo FTF. De uma forma similar à utilizada no método de Botto e Moutakides, um sinal deste tipo será utilizado para corrigir as variáveis. Porém, neste caso, ele será definido como

$$\theta(n) = \varepsilon_b(n) - \gamma(n)\, w\, \tilde{m}(n)\, E_b(n-1) \qquad (4.52)$$

Como pode ser observado, este sinal difere de $\xi(n)$, em (4.25), pelo valor de $\gamma(n)$, uma vez que neste caso estão sendo utilizados os erros a posteriori, e não mais os erros a priori. Esta diferença se deve ao fato de que este sinal pretende corrigir o vetor B(n) e não mais B(n-1) como anteriormente. Com isso, o papel desempenhado no método de Botto e Moustakides pela variável $e_b(n)$ é agora substituído por $\varepsilon_b(n)$.

Denotando por $\bar{B}(n)$ o vetor corrigido do preditor regressivo, e $\bar{\theta}(n)$ o valor corrigido da variável $\theta(n)$, temos

$$\bar{B}(n) = B(n) + \bar{\theta}^*(n)\, G(n) \qquad (4.53)$$

Tabela 4.6 - Novo Algoritmo Estável

disponível, do instante n-1

$X(n-1), H(n-1), A(n-1), B(n-1), \bar{G}(n-1), \gamma(n-1), E_a(n-1), E_b(n-1)$

recebido no instante n

$x(n), d(n)$

atualização dos preditores

$$e_a(n) = x(n) - A^H(n-1)X(n-1)$$

$$\begin{bmatrix} \bar{M}(n) \\ \tilde{m}(n) \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{G}(n-1) \end{bmatrix} + \frac{e_a(n)}{w\, E_a(n-1)} \begin{bmatrix} 1 \\ -A(n-1) \end{bmatrix}$$

$$\varepsilon_a(n) = \gamma(n-1)\, e_a(n)$$

$$A(n) = A(n-1) + \varepsilon_a^*(n)\, \bar{G}(n-1)$$

$$E_a(n) = w\, E_a(n-1) + \varepsilon_a^*(n) e_a(n)$$

$$e_b(n) = x(n-N) - B^H(n-1)X(n)$$

$$\gamma_1(n) = \frac{w\, E_a(n-1)}{E_a(n)}\, \gamma(n-1)$$

$$\gamma(n) = \frac{\gamma_1(n)}{1 - \gamma_1(n)\tilde{m}^*(n) e_b(n)}$$

$$\bar{G}(n) = \bar{M}(n) + \tilde{m}(n)\, B(n-1)$$

$$\varepsilon_b(n) = \gamma(n) e_b(n)$$

$$\theta(n) = \varepsilon_b(n) - \gamma(n)\, w\, \tilde{m}(n)\, E_b(n-1)$$

$$\bar{\theta}(n) = \gamma(n)\, \theta(n)$$

$$\bar{B}(n) = B(n-1) + [\varepsilon_b^*(n) + \bar{\theta}^*(n)]\, \bar{G}(n)$$

$$\bar{\varepsilon}_b(n) = \varepsilon_b(n) - \bar{\theta}(n) \left[\frac{1}{\gamma(n)} - 1\right]$$

$$E_b(n) = w\, E_b(n-1) + \varepsilon_b^*(n) e_b(n)$$

atualização do filtro adaptativo

$$e(n) = d(n) - H^H(n-1)X(n)$$

$$\varepsilon(n) = \gamma(n) e(n)$$

$$H(n) = H(n-1) + \varepsilon^*(n)$$

O interessante desta correção vetorial é que ela representa um custo computacional de apenas uma soma. Analogamente ao realizado na Seção 4.3, pode-se mostrar que a equação (4.53) resulta de uma minimização com relação a $\bar{B}(n)$ sobre a função custo

$$\Omega(n) = w\, [\bar{B}(n) - B(n)]^H\, R(n-1)\, [\bar{B}(n) - B(n)] + |\bar{\theta}(n)|^2 \qquad (4.54)$$

Usando (4.53) e (4.32), o erro de predição regressiva a posteriori pode ser corrigido como

$$\bar{\varepsilon}_b(n) = \varepsilon_b(n) - \bar{\theta}(n)\left[\frac{1}{\gamma(n)} - 1\right] \tag{4.55}$$

Usando (4.55) na definição (4.52) temos, após algumas simplificações

$$\bar{\theta}(n) = \gamma(n)\,\theta(n) \tag{4.56}$$

A utilização de $\bar{e}_b(n)$, ao invés de $e_b(n)$ na fórmula que atualiza a energia do erro de predição regressiva, $E_b(n)$, resulta em um valor corrigido para esta variável.

O novo algoritmo estável é apresentado na Tabela 4.6, onde pode ser observada sua grande simplicidade. A introdução das fórmulas (4.52), (4.55) e (4.56) no Algoritmo FTF implicam em um acréscimo computacional de 4 somas, 4 multiplicações e uma divisão por iteração.

Da mesma forma que no método de Botto e Moustakides, seria possível incluir um parâmetro para controlar o nível de correção (a constante ρ). Essa medida, no entanto, traria uma complexidade desnecessária ao método. Em várias simulações ficou demonstrado que esta constante não produz maiores benefícios neste método de estabilização.

Quanto ao desempenho do algoritmo em relação à estabilidade numérica foram simuladas diversas situações, usando principalmente sinais estacionários e fatores de esquecimento de valores razoáveis para a utilidade prática. Em todas as situações o desempenho foi observado por milhões de iterações sem que qualquer tendência de instabilidade fosse observada.

Também foi observado o desempenho do novo método para o caso de valores relativamente baixos para o fator de esquecimento. Assim como os demais métodos existentes, este também não consegue evitar a divergência das variáveis para valores abaixo de 0,95.

A grande vantagem do novo método reside no reduzido esforço computacional adicionado ao Algoritmo FTF-8N original.

Como forma de comprovar a eficiência do novo método de estabilização, foram também realizadas diversas comparações, ao nível experimental, com os métodos já existentes. Os resultados estão apresentados no Capítulo 5.

4.5 – COMENTÁRIO

Neste capítulo foi apresentado um grande leque de possibilidades para a estabilização dos algoritmos rápidos. Dentre os métodos descritos é preciso ressaltar a importância dos métodos que preservam a solução LS pois, além de garantir uma solução ótima, eles apresentam um expressivo desempenho, em termos de estabilidade, quando operando dentro de uma faixa de valores plausíveis para o fator de esquecimento. Apesar de não haver uma plena garantia de estabilidade, as simulações demonstram claramente uma situação de segurança para execuções da ordem de milhões de iterações.

O novo método de estabilização, apresentado na Seção 4.4, representa uma interessante alternativa para aplicações práticas, dado que exige um número reduzido de operações matemáticas para um desempenho comparável aos demais métodos existentes.

O caso de fatores de esquecimentos baixos continua ainda uma barreira intransponível para o problema de instabilidade dos algoritmos rápidos. Os métodos existentes não garantem a estabilidade senão para valores acima de 0,95, o que, entretanto, é suficiente para a maioria das aplicações práticas.

4.6 – REFERÊNCIAS

[1] Cioffi, J. M. and Kailath, T., "Fast, Recursive Least-Squares Transversal Filters for Adaptive Filtering", IEEE Trans. Acoustics, Speech and Signal Processing, vol. 32, pp. 304-337, April 1984.

[2] Lin, D. W., "On Digital Implementation of the Fast Kalman Algorithms", IEEE Trans. Acoustics, Speech and Signal Processing, vol. ASSP-32, pp. 998-1005, October, 1984.

[3] Bellanger, M. G., Adaptive Digital Filters and Signal Analysis, Marcel Dekker Inc., New York, 1987.

[4] Fabre, P. and Guegen, C., "Fast Recursive Least-Squares Algorithms: Preventing Divergence", Proc. of IEEE ICASSP 85, pp 1149-1152, Tampa, 1985.

[5] Botto, J. L. and Moustakides, G. V., "Stabilizing the Fast Kalman Algorithms", IEEE Trans. Acoustics, Speech and Signal Processing, vol. ASSP-37, pp. 1342-1348, September, 1989.

[6] Cioffi, J. M., "Limited-Precision Effects in Adaptive Filtering" IEEE Trans. Circuits and Systems, vol CAS-34, pp. 821-833, July, 1987.

[7] Slock, D. T. M. and Kailath, T., "Numerically Stable Fast Recursive Least-Square Transversal Filters", Proc. of IEEE ICASSP 88, pp 1365-1372, New York, 1988.

[8] Benallal, A. and Gilloire, A., "A New Method to Stabilize Fast RLS Algorithms Based on a First Order Model of the Propagation of Numerical Errors", Proc. of IEEE ICASSP 88, pp. 1373-1376, New York, 1988.

[9] Slock, D. T. M. and Kailath, T., "Numerically Stable Fast Transversal Filters for Recursive Least Squares Adaptive Filtering", IEEE Trans. Acoustics, Speech and Signal Processing, vol. ASSP-39, pp. 92-113, January, 1991.

[10] Benallal, A. "Etude des Algorithmes des Moindres Carres Transversaux Rapides et Application a L'Identification de Responses Impulsionnelles Acoustiques", Thèse présentée devant L'Universite de Rennes I, Janvier, 1989.

CAPÍTULO 5

ANÁLISE EXPERIMENTAL DOS ALGORITMOS ESTABILIZADOS

Neste capítulo vamos analisar experimentalmente alguns dos algoritmos de mínimos quadrados rápidos estabilizados discutidos no Capítulo 4. Serão apresentados os resultados de diversas simulações, procurando sempre comparar os desempenhos dos métodos de estabilização com vistas à aplicação em sistemas práticos.

A exemplo das simulações apresentadas no Capítulo 3, aqui também será utilizada aritmética em ponto flutuante e sinais com características estacionárias bem definidas, gerados sinteticamente.

Serão analisados os comportamentos dos algoritmos rápidos estabilizados para uma vasta gama de situações, variando o fator de esquecimento, as características de estacionaridade do sinal de entrada, o numero de bits usado na implementação das variáveis e o número de coeficientes do filtro transversal.

Os métodos de Botto e Moutakides, de Slock e Kailath, de Benallal e Gilloire, assim como o novo método de estabilização, apresentado na Seção 4.4, receberão uma maior atenção, pois são, a nosso ver, os que apresentam os melhores requisitos para a estabilização dos algoritmos rápidos quando aplicados a sistemas reais.

Em praticamente todas as situações simuladas poderá ser comprovada a eficiência do novo método, cujo desempenho apresenta sempre alguma superioridade em relação aos demais métodos de estabilização. Estes resultados, aliados à pouca complexidade requerida em sua implementação, conferem ao novo método um importante papel no elenco de soluções para o problema de instabilidade numérica dos algoritmos rápidos.

5.1 – ANÁLISE QUANTO AO FATOR DE ESQUECIMENTO

Como já mencionado anteriormente, os métodos de estabilização não funcionam a contento para o caso de fatores de esquecimentos baixos. Ainda que estes casos não tenham grande interesse prático, a operação dos algoritmos rápidos nesta região se presta para testar os métodos de estabilização, pois representa uma condição desfavorável. Também neste sentido, foi privilegiado o emprego de uma precisão baixa, de 23 bits na mantissa, para as variáveis em ponto flutuante.

No primeiro teste foi utilizado um mesmo sinal de excitação para os quatro algoritmos estabilizados mencionados acima, sendo que o fator de esquecimento foi variado em uma região de 0,9 a 0,96 para medir número de iterações, n_D, suportado por eles. Os resultados estão mostrados na Tabela 5.1, onde foi também colocado o n_D obtido com o Algoritmo FTF original. Pode-se notar que os resultados dos quatro métodos de estabilização são comparáveis, e também que todos eles melhoram substancialmente a estabilidade dos algoritmos na medida em que w aumenta. Para o caso de w = 0,96, foram observadas dezenas de milhões de iterações, usando-se os quatro métodos, sem que fosse apresentado qualquer sintoma de divergência. Pode-se afirmar que a partir deste valor, para uma precisão de 23 bits e N = 4, os quatro algoritmos são estáveis.

O sinal utilizado para a excitação do filtro transversal foi, no caso desta experiência, um processo AR, com variância 1 e amostras relativamente correlacionadas. Uma série temporal destas amostras, utilizada em algumas das execuções, está ilustrada na Figura 5.1.

É curioso notar que a estabilização do algoritmo aumenta consideravelmente a variância de n_D para esta faixa de valores de w, o que não acontecia com os algoritmos em sua forma original. Os resultados da Tabela 5.1 foram obtidos a partir de uma média sobre 5 execuções, onde se variou unicamente a semente do gerador randômico. No entanto, as amostras consideradas para o cálculo de n_D apresentavam valores, as vezes, bem diferentes. Este aumento na variância de n_D é explicado pela proximidade do limiar de estabilização.

Tabela 5.1 - Comparação dos algoritmos quanto ao valor de w

w \ Método	FTF-8N	Slock/Kailath	Botto /Moust.	Benallal/Gill.	Novo Método
w = 0,9	418	684	774	503	922
w = 0,91	450	864	920	752	1.006
w = 0,92	479	1.057	1.687	1.041	1.299
w = 0,93	606	1.516	4.190	3.539	3.634
w = 0,94	788	2.952	13.322	16.201	16.907
w = 0,95	890	12.720	268.553	395.281	491.714

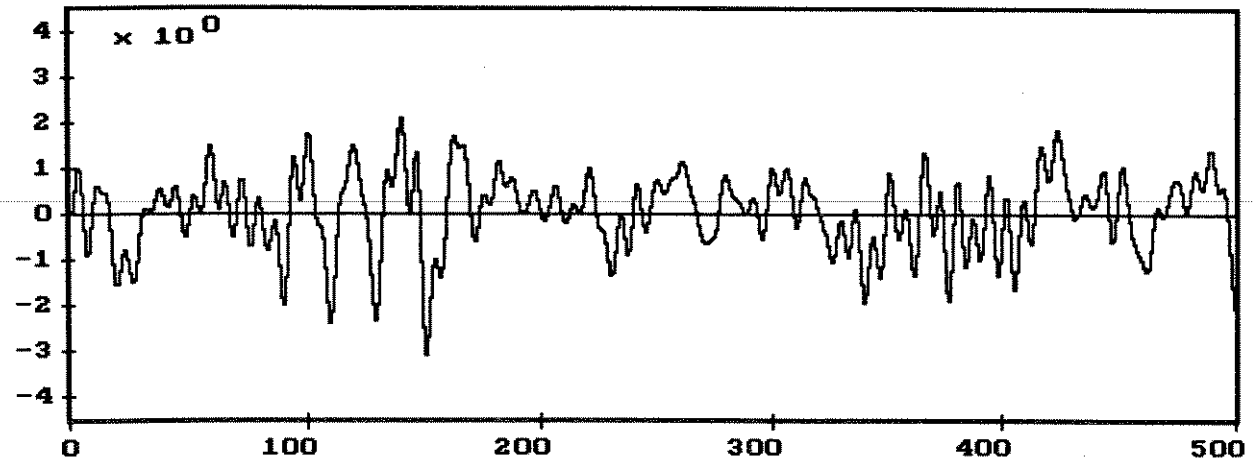


Figura 5.1 - Amostra do sinal utilizado para excitar o filtro transversal nos testes da Tabela 5.1. Este sinal é um processo AR, de variância unitária, de ordem 4, sintetizado com $c_1 = 3{,}08$, $c_2 = -4{,}13$, $c_3 = 2{,}86$ e $c_4 = -0{,}85$

O efeito de limiar pode ser melhor observado no gráfico da Figura 5.2, que foi construído para o caso do novo método de estabilização com os valores da Tabela 5.1 . Nota-se que o gráfico apresenta uma assíntota próxima de w=0.954, que significa o valor limite para a estabilidade do algoritmo. É difícil precisar corretamente este valor para cada um dos algoritmos, mas os resultados sugerem que, neste caso particular, ele é um pouco menor para o novo método de estabilização, sendo seguido pelos algoritmos de Benallal e Gilloire, de Botto e Moutakides, e por último, pelo de Slock e Kailath.

Devido à multiplicidade de constantes permitidas nos métodos de Slock e Kailath, de Benallal e Gilloire, e de Botto e Moustakides, pode-se dizer que estes métodos representam uma família de métodos, estando cada elemento desta família associado a um particular conjunto de constantes. A família correspondente ao método de Benallal e Gilloire, por exemplo, tem três graus de liberdade. Para as simulações deste capítulo preferimos, ao invés de fazer inúmeros experimentos, arbitrando os mais diversos valores para estas constantes, definir um único conjunto de valores e fazer todas as simulações com este particular conjunto.

No caso do Algoritmo de Botto e Moustakides, foi utilizado $\rho = 1$, pois o método se mostra mais eficaz com este valor. No caso do Algoritmo de Benallal e Gilloire, escolhemos fazer $k_i = 1$, $i = 1, 2, 3$, seguindo uma recomendação encontrada em [1]. Para o caso do método de Slock e Kailath, fizemos $k_1 = 1{,}5$, $k_2 = 2{,}5$, $k_3 = 1$, $k_4 = 0$, $k_5 = 1$ e $k_6 = 0$, conforme sugerido em [2]. Pudemos constatar experimentalmente que, de fato, estes valores permitem um bom funcionamento do algoritmo. Além disso, em virtude da escolha de $k_4 = 0$, temos que a complexidade computacional do algoritmo é reduzida de 9N para 8N.

Nas simulações da Tabela 5.1, onde se observa o desempenho dos algoritmos estabilizados para valores baixos de w, foram utilizados vários critérios de parada. Isto porque os algoritmos diferem na forma que apresentam os sintomas de divergência. A técnica de observar o valor de $\gamma(n)$, determinando a parada quando esta variável extrapola sua faixa de validade, somente funciona no novo método de estabilização. Vale ressaltar que em alguns casos este algoritmo

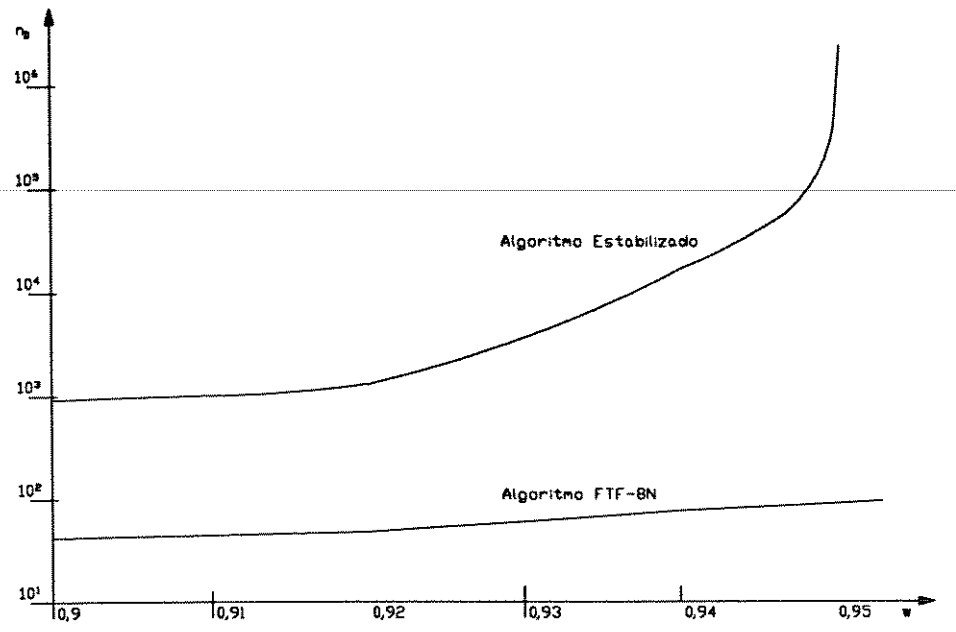


Figura 5.2 - Número de iterações suportadas, em função de w, pelo Algoritmo FTF-8N, e pelo algoritmo estabilizado com o novo método .

continua funcionando de uma forma aparentemente normal, mesmo após haver apresentado um valor acima de 1 para $\gamma(n)$. No cômputo dos valores da Tabela 5.1, entretanto, foi considerada qualquer ocorrência que denotasse uma anormalidade. A Figura 5.3 ilustra uma execução típica do algoritmo estabilizado com o novo método quando são empregados valores baixos para w. Nota-se, neste caso, a precedência da violação de $\gamma(n)$ sobre a divergência das variáveis.

No caso do método de Botto e Moustakides é mais provável que o primeiro sintoma de anormalidade seja uma ocorrência de overflow do que a violação da variável de verossimilhança, ou de qualquer outra variável. Para evitar este evento indesejável foi empregado, conforme sugestão em [3], a condição de alerta:

$$|\xi(n)|^2 < 0{,}01 \qquad (5.1)$$

Esta condição, apesar de bastante relaxada devido ao alto valor arbitrado como limite, é sempre a que determina o instante de parada no algoritmo de Botto e Moustakides, pois ocorre antes que o overflow ou a violação de $\gamma(n)$. A Figura 5.4 mostra um caso típico, onde nota-se que a condição (5.1) previne o overflow apresentado pela variável $E_b(n)$.

No método de Benallal e Gilloire, ao invés da ocorrência de overflow, o algoritmo perde a capacidade de rastreio, ficando os elementos dos preditores com valores fixos, enquanto que algumas variáveis escalares tendem a zero e outras tendem ao infinito. Esta anormalidade ocorre sem que $\gamma(n)$ extrapole sua faixa de validade. Usando o critério de alerta estabelecido em (5.1), este tipo de anormalidade pode ser prevenido. A Figura 5.5 ilustra esta situação.

No caso do Algoritmo de Slock e Kailath, observa-se também a ocorrência de overflow. A variável $\zeta(n)$, no entanto, consegue antecipar esta ocorrência. Vide Figura 5.6.

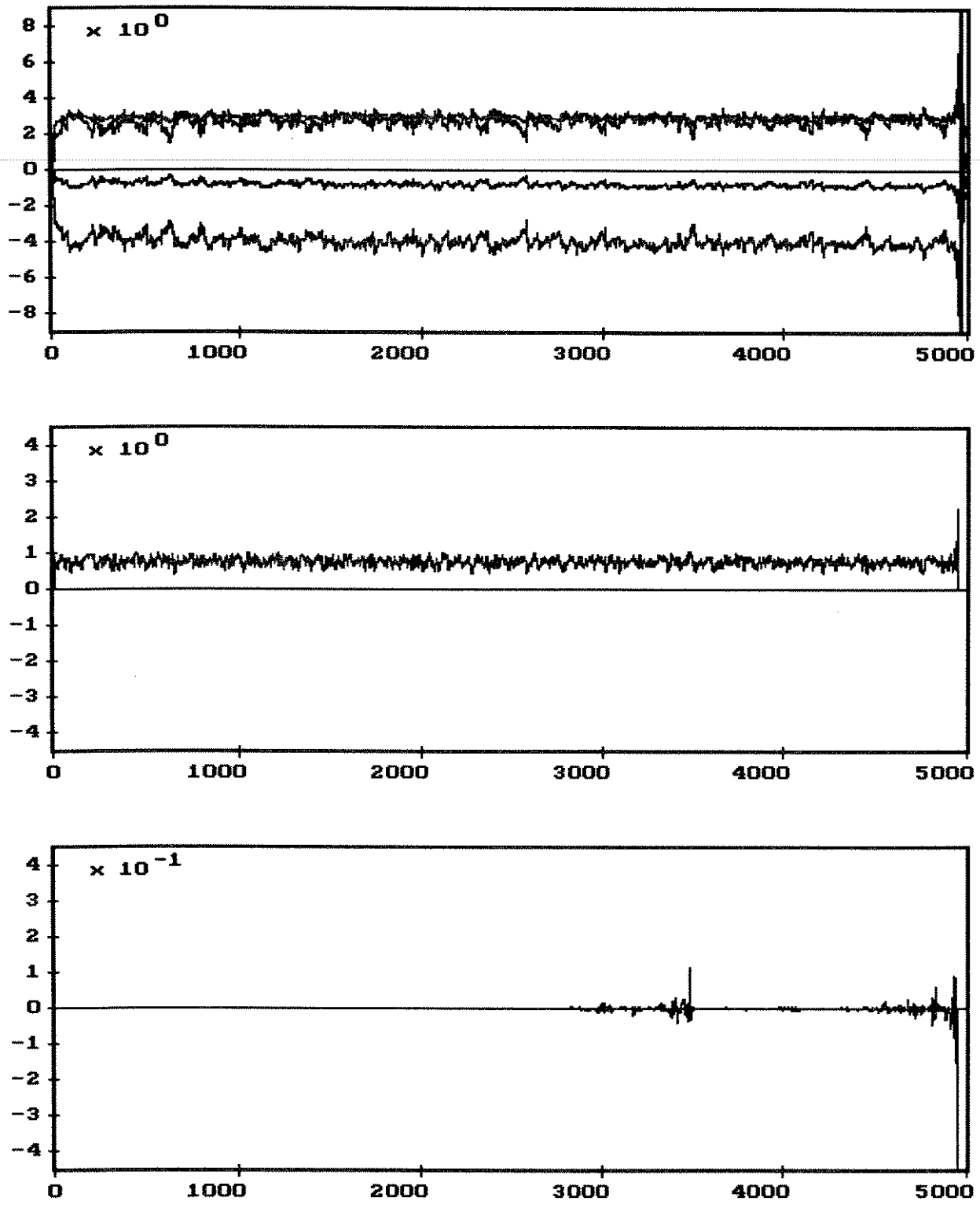


Figura 5.3 - Execução de um processo de predição usando o novo método de estabilização. Foram usados o sinal ilustrado na Figura 5.1 como x(n) e w = 0,934. Em **a** é observada a evolução dos coeficientes do preditor progressivo, em **b**, a variável $\gamma(n)$, e em **c**, a variável de controde de erro $\theta(n)$. A variável $\gamma(n)$ apresentou um valor fora de sua faixa de validade para n = 4929, enquanto que $|\theta(n)|^2 > 0{,}01$ para n = 4993.

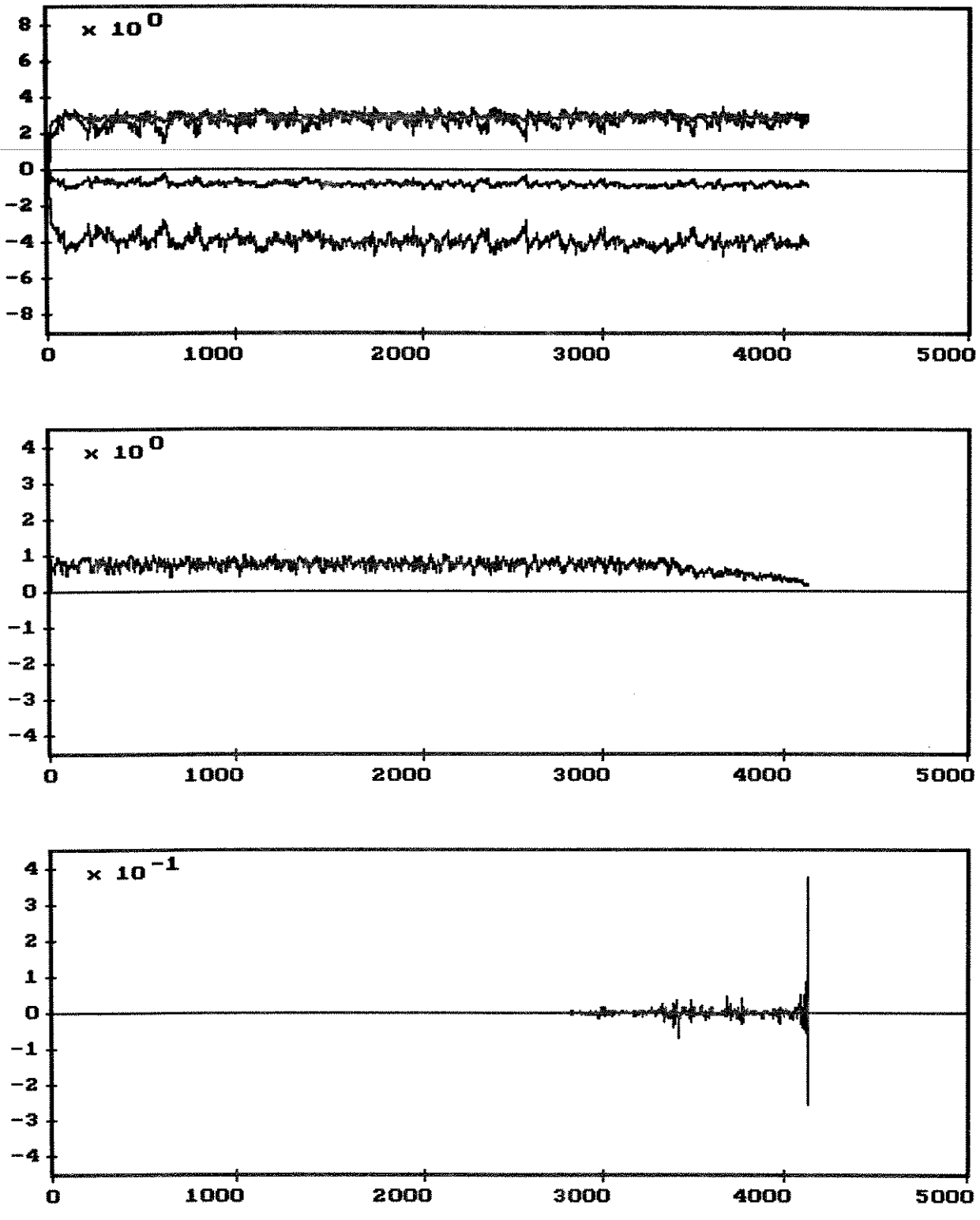


Figura 5.4 - Execução de um processo de predição usando o método de Botto e Moustakides. Foram usadas as mesmas condições da Figura 5.3. Em **a** é observada a evolução dos coeficientes do preditor progressivo, em **b**, a variável $\gamma(n)$, e em **c**, a variável de controle de erro $\xi(n)$. Houve um overflow em n = 4138. A variável $\gamma(n)$, entretanto, não apresentou qualquer valor fora de sua faixa de validade. Por outro lado, tivemos $|\xi(n)|^2 > 0{,}01$ para n = 4128.

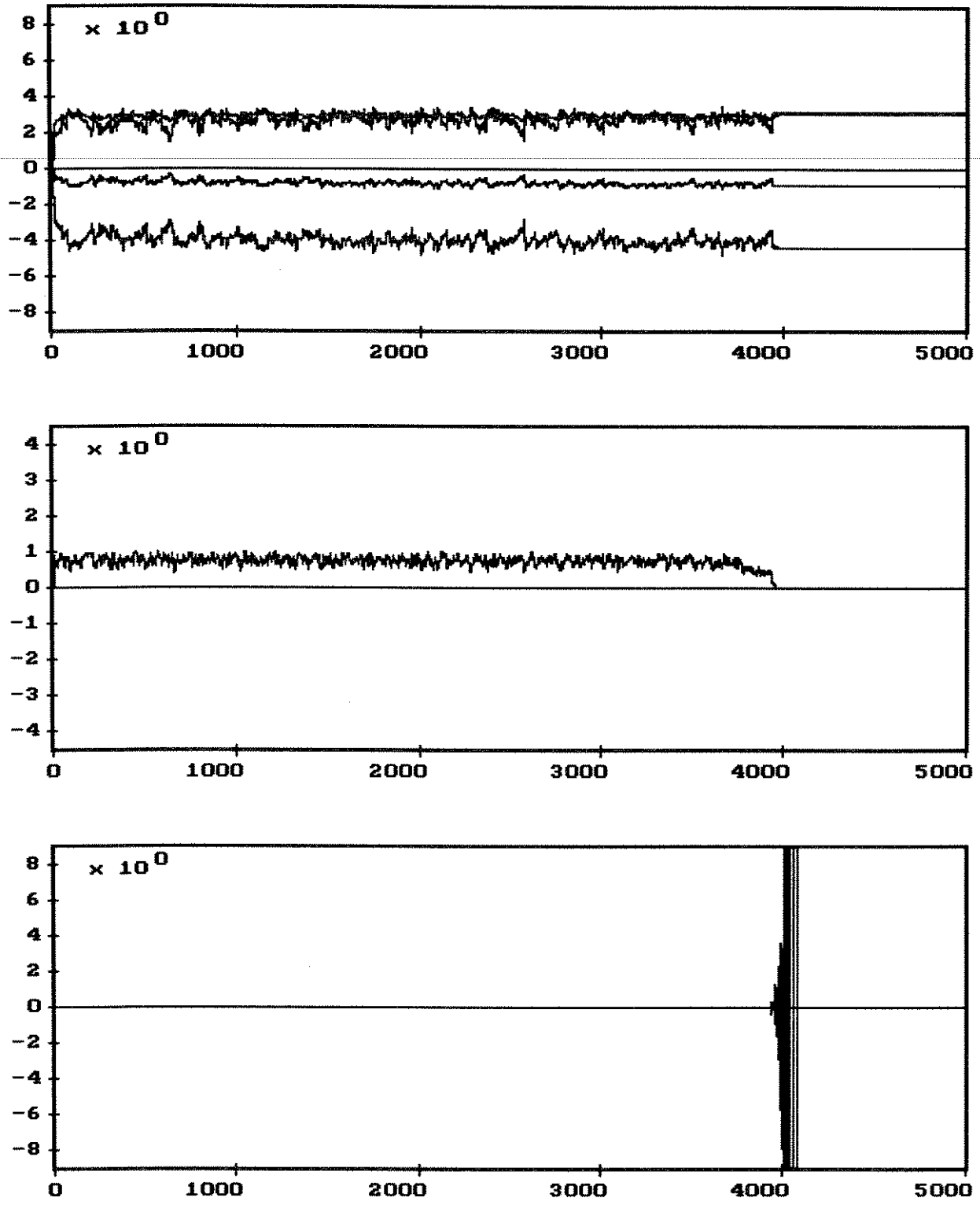
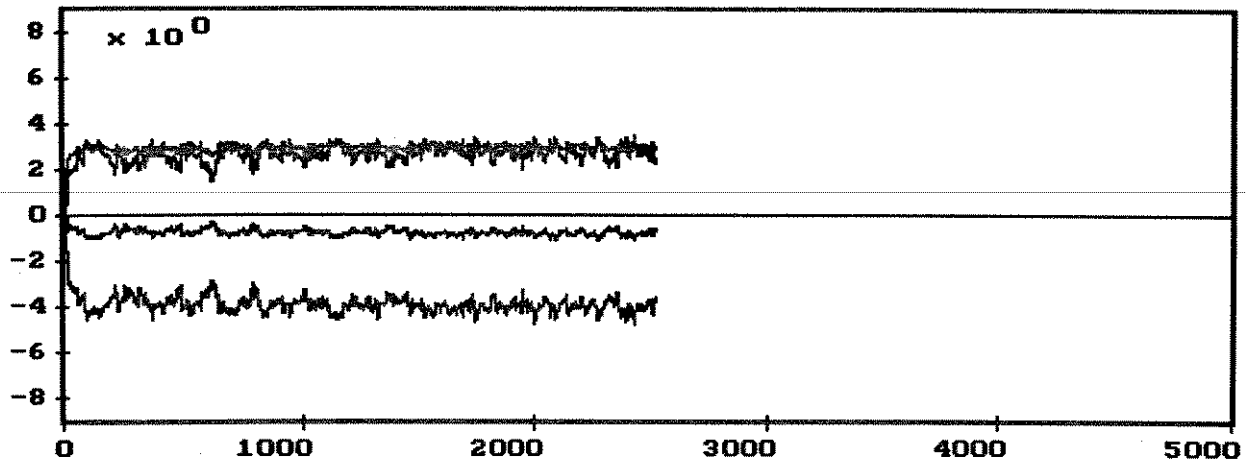
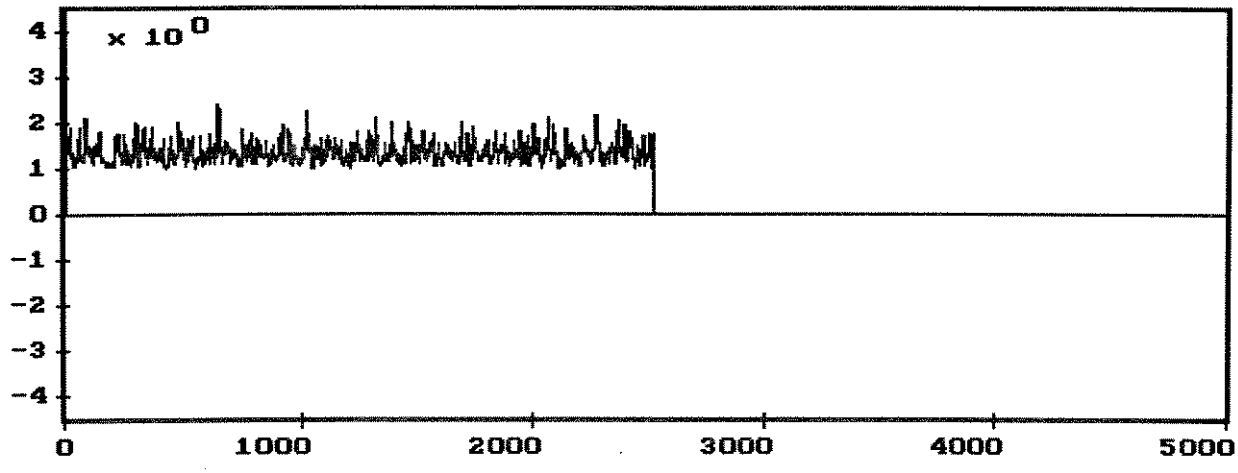


Figura 5.5 - Execução de um processo de predição usando o método de Benallal e Gilloire. Foram usadas as mesmas condições da Figura 5.3. Em **a** é observada a evolução dos coeficientes do preditor progressivo, em **b**, a variável $\gamma(n)$, e em **c**, a variável de controde de erro $\xi(n)$. A variável $\gamma(n)$, não apresentou qualquer valor fora de sua faixa de validade. Por outro lado, tivemos $|\xi(n)|^2 > 0{,}01$ para $n = 3935$.





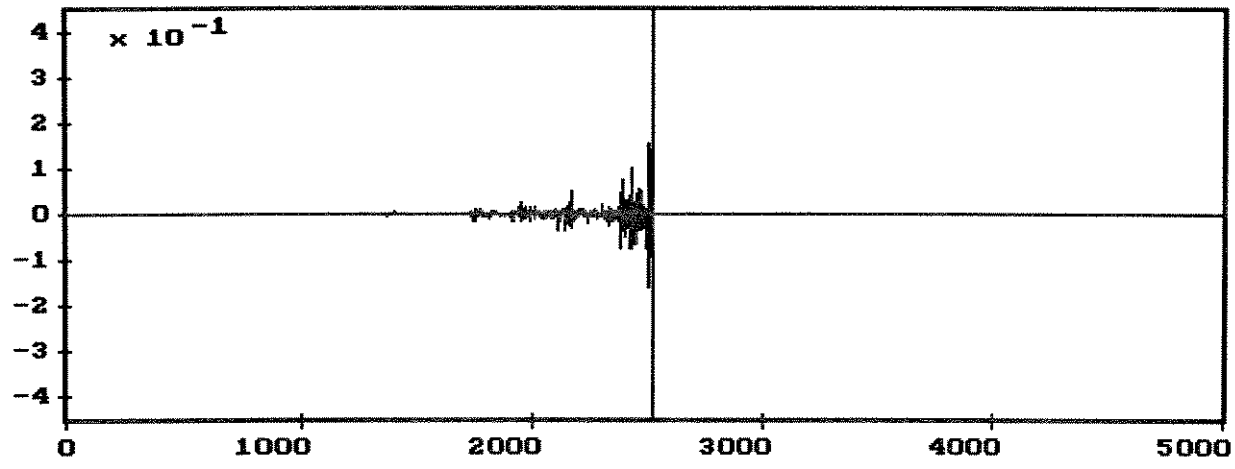


Figura 5.6 - Execução de um processo de predição usando o método de Slock e Kailath. Foram usadas as mesmas condições da Figura 5.3. Em **a** é observada a evolução dos coeficientes do preditor progressivo, em **b**, a variável $\zeta(n)$, e em **c**, a variável de controde de erro $\xi(n)$. Houve um overflow em n = 2524. A variável $\zeta(n)$, apresentou um valor fora de sua faixa de validade para n = 2433.

Nas simulações referidas acima, a condição imposta por (5.1) sobre a variável de controle de erro é bastante tolerante, pois permite que o módulo de $\xi(n)$ atinja até o valor de 0,1. Com isso, a vida do algoritmo é prolongada ao máximo possível. Restringindo a condição sobre $\xi(n)$ [ou $\theta(n)$ no novo método], os números da Tabela 5.1 caem um pouco. Esta alteração privilegia o novo método de estabilização, onde a queda é menor. Isto porque, nas simulações realizadas, o critério que determina o estado de alerta no novo método tem sido o valor de $\gamma(n)$, e não a restrição sobre $\theta(n)$.

Ainda a respeito do valor da variável de verossimilhança, é preciso mencionar que nos quatro métodos experimentados, quando são utilizados fatores de esquecimento muito próximos de 1, é possível ocorrer uma violação desta variável. Observa-se esporadicamente a ocorrência de um valor ligeiramente maior que 1 [$\zeta(n)$ menor que 1 no caso do Algoritmo de Slock e Kailath]. O funcionamento do algoritmo, entretanto, mantém-se inalterado. Foram realizados alguns testes para comprovar a exatidão da resposta destes algoritmos com o critério LS quando da ocorrência destes valores. Estes testes foram efetuados comparando-se as variáveis calculadas com as de um algoritmo implementado com precisão numérica superior. Pudemos comprovar que esta pequena desconformidade no valor de $\gamma(n)$ não influencia a exatidão da solução LS, mantendo-se as variáveis rigorosamente dentro dos valores devidos, especialmente os preditores.

5.2 – ANÁLISE QUANTO À PRECISÃO ARITMÉTICA

A primeira pergunta que surge quando alguma modificação é introduzida no algoritmo é se a resposta do sistema continua sendo a mesma. Ou seja, se no caso dos algoritmos estabilizados, com as modificações introduzidas em seu comportamento dinâmico, as variáveis escalares e vetoriais continuam a apresentar os devidos valores. Seria plausível que o algoritmo se mantivesse estável à custa de alguma alteração que mudasse os valores matematicamente corretos segundo o critério de mínimos quadrados. Com o objetivo de investigar esta possibilidade, executamos diversas vezes os algoritmos estabilizados variando a precisão aritmética empregada. Os resultados foram comparados com os fornecidos por um algoritmo implementado com precisão superior, que para os efeitos práticos do teste, era como se fosse um resultado ideal. Como parâmetro de medida foi utilizada a definição $|\mathbf{B}(n) - \overline{\mathbf{B}}(n)|^2$, onde $\overline{\mathbf{B}}(n)$ representa o preditor regressivo calculado com o algoritmo de maior precisão.

Os resultados estão mostrados na Figura 5.7, onde se observa as primeiras 100.000 iterações do mesmo processo implementado com três diferentes níveis de precisão. Foram usados 23, 39 e 52 bits na mantissa das variáveis em ponto flutuante. Observa-se, nos três casos, que o erro não cresce com o tempo, mantendo-se rigorosamente dentro do limite de precisão empregado; o que demonstra a exatidão da resposta obtida com a estabilização. Apesar de a figura ilustrar somente o erro apresentado pelo algoritmo estabilizado com o novo método, os três outros algoritmos estabilizados também foram testados, e apresentaram exatamente o mesmo nível de erro para cada uma das precisões simuladas.

Ainda que pudéssemos utilizar como referência um algoritmo reconhecidamente estável, como por exemplo o RLS, preferimos usar o próprio algoritmo FTF estabilizado, com uma

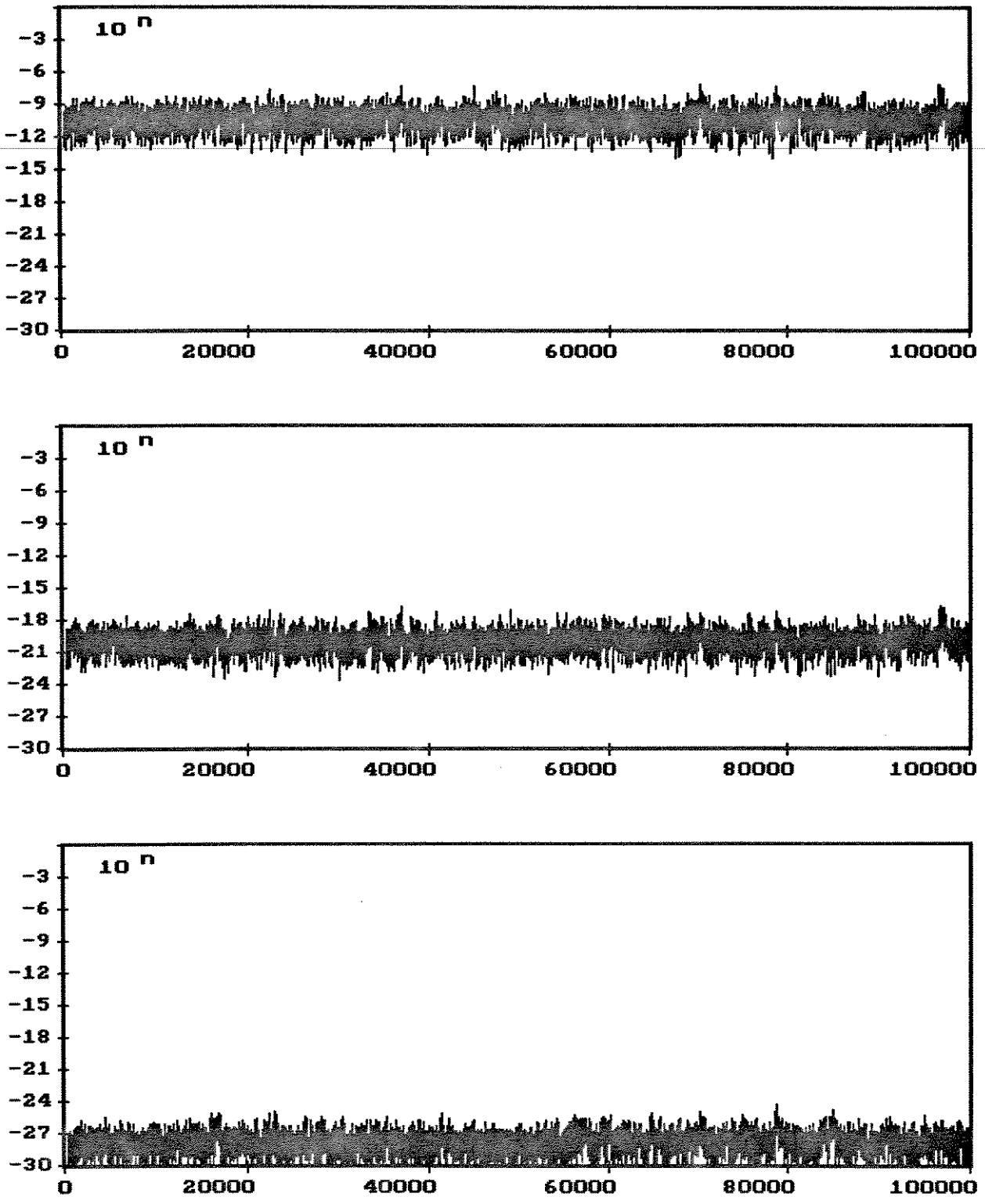


Figura 5.7 - Três diferentes simulações de um mesmo processo de predição usando o novo método de estabilização. Em cada execução foi utilizado um diferente nível de precisão aritmética. Em **a** usamos b = 23, em **b**, b = 39, e em **c**, b = 52. O fator de esquecimento foi w = 0,97 e o sinal de entrada foi o mesmo descrito na Figura 5.1. A variável observada é $|B(n) - \bar{B}(n)|^2$.

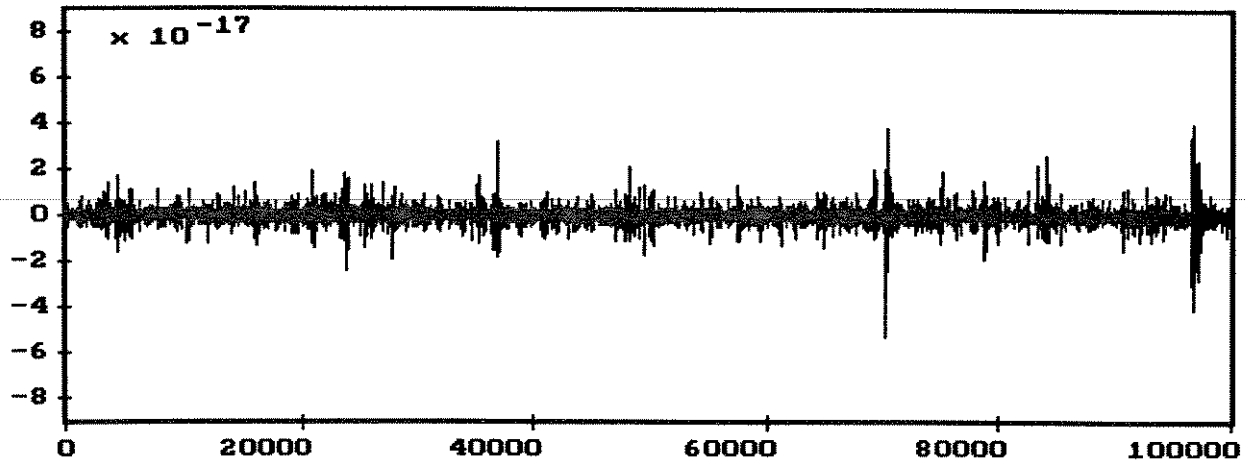


Figura 5.8 - Variável de erro θ(n) no algoritmo estabilizado pelo novo método. O processo simulado foi o mesmo da Figura 5.6, porém com uma precisão de 63 bits na mantissa. É notável a exatidão numérica desta implementação.

precisão de 63 bits na mantissa. Isto porque a resposta deste algoritmo é suficientemente precisa para a comparação desejada. A título de ilustração exibimos na Figura 5.8 a variável de erro, θ(n), para esta implementação. Nota-se que ao longo de 100.000 iterações o valor absoluto desta variável manteve-se praticamente abaixo de 10^{-17}, demonstrando o alto grau de acuidade numérica oferecido pelo algoritmo.

Outra questão a ser considerada, em termos do limite de precisão, diz respeito a quanto o algoritmo ganha de estabilidade com o aumento do número de bits. Para responder a esta questão foram feitas inúmeras simulações, onde procurou-se avaliar o limiar de estabilidade em função do número de bits utilizado na implementação das variáveis. Como resultado destes testes, foi observado que a estabilidade dos algoritmos é muito pouco afetada em decorrência da precisão aritmética empregada. Tomemos como exemplo o caso do algoritmo implementado com o novo método, que para uma precisão de 23 bits apresentou um limiar de estabilidade de w = 0,954, enquanto que para uma precisão de 63 bits apresentou um limiar de w = 0,943. Lembrando que este adicional de 40 bits representa um acréscimo de 12 casas decimais na precisão das variáveis, temos que o aumento de estabilidade obtido é inexpressivo.

Este resultado pode parecer pouco intuitivo à primeira vista. Mas, se levarmos em conta as considerações do Capítulo 3, veremos que não é exatamente o ruído adicionado em cada iteração que provoca a divergência das variáveis, mas sim o comportamento dinâmico do algoritmo. Ou seja, a forma como ele amplifica o ruído acumulado. Da mesma forma que nos algoritmos não estabilizados, o aumento da precisão não proporciona estabilidade. Como exemplo, a Tabela 5.2 ilustra os valores de n_D para o caso do algoritmo estabilizado pelo novo método, com valores de w próximos do limiar de estabilidade. Pode-se observar que, mesmo para esta condição, o aumento do número de bits não prolonga a vida dos processos senão por um fator correspondente ao aumento da precisão.

Concluímos, mais uma vez, que o aumento da precisão aritmética é um procedimento inútil sob o ponto de vista da estabilidade numérica dos algoritmos de mínimos quadrados rápidos.

Tabela 5.2 - Número de iterações suportadas pelo novo método de estabilização

w \ nº de bits	23 bits	39 bits	52 bits	63 bits
w = 0,9	922	1.772	2.528	3.260
w = 0,91	1006	2.182	3.198	4.468
w = 0,92	1.299	4.187	5.445	6.553
w = 0,93	3.634	8.570	12.070	15.035
w = 0,94	16.907	39.283	51.934	68.222
w = 0,95	491.714	estável	estável	estável

Tanto no caso dos algoritmos originais, quanto no caso dos algoritmos estabilizados, o desempenho obtido com este recurso não é satisfatório. Para o caso dos algoritmos estabilizados, com valores de w acima do limiar, o aumento do número de bits representa um maior nível de exatidão numérica nos resultados, o que não se traduz por estabilidade.

5.3 – ANÁLISE QUANTO À ORDEM DO FILTRO TRANSVERSAL

Infelizmente os algoritmos rápidos estabilizados são muito sensíveis ao número de coeficientes do filtro transversal. Em inúmeros experimentos, pudemos comprovar que a instabilidade numérica aumenta substancialmente com o aumento da ordem do filtro.

Nesta seção fizemos simulações usando ruído branco gaussiano e variando a ordem do filtro para os diversos algoritmos rápidos estabilizados. Os efeitos produzidos em cada um deles para um valor elevado de N são muito similares. Como exemplo, tomemos o algoritmo estabilizado pelo novo método. A Figura 5.9 ilustra um efeito típico em um processo de identificação de sistemas para o caso de N grande. Observa-se, na parte **b** desta figura, que a variável de erro, $\theta(n)$, não apresenta qualquer atuação para n menor que 40, a ordem do filtro usada. Esta variável se mantém rigorosamente nula para estas N primeiras iterações. Isto se deve a uma característica inerente aos algoritmos rápidos, que é a nulidade do vetor $B(n)$, e dos erros de predição regressiva para esta faixa inicial de iterações, como pode ser observado na Figura 5.10.

Como conseqüência da não atuação do mecanismo estabilizador, nas N primeiras iterações, observa-se um acúmulo de erro, que se traduz por um crescimento acentuado de $\theta(n)$ a partir do instante n = N, quando esta variável é liberada para operar. Após um certo período de tempo, em que o sinal de erro se mantém em níveis elevados, advém o que podemos chamar de "plena estabilidade do algoritmo", com o erro decrescendo para valores muito baixos.

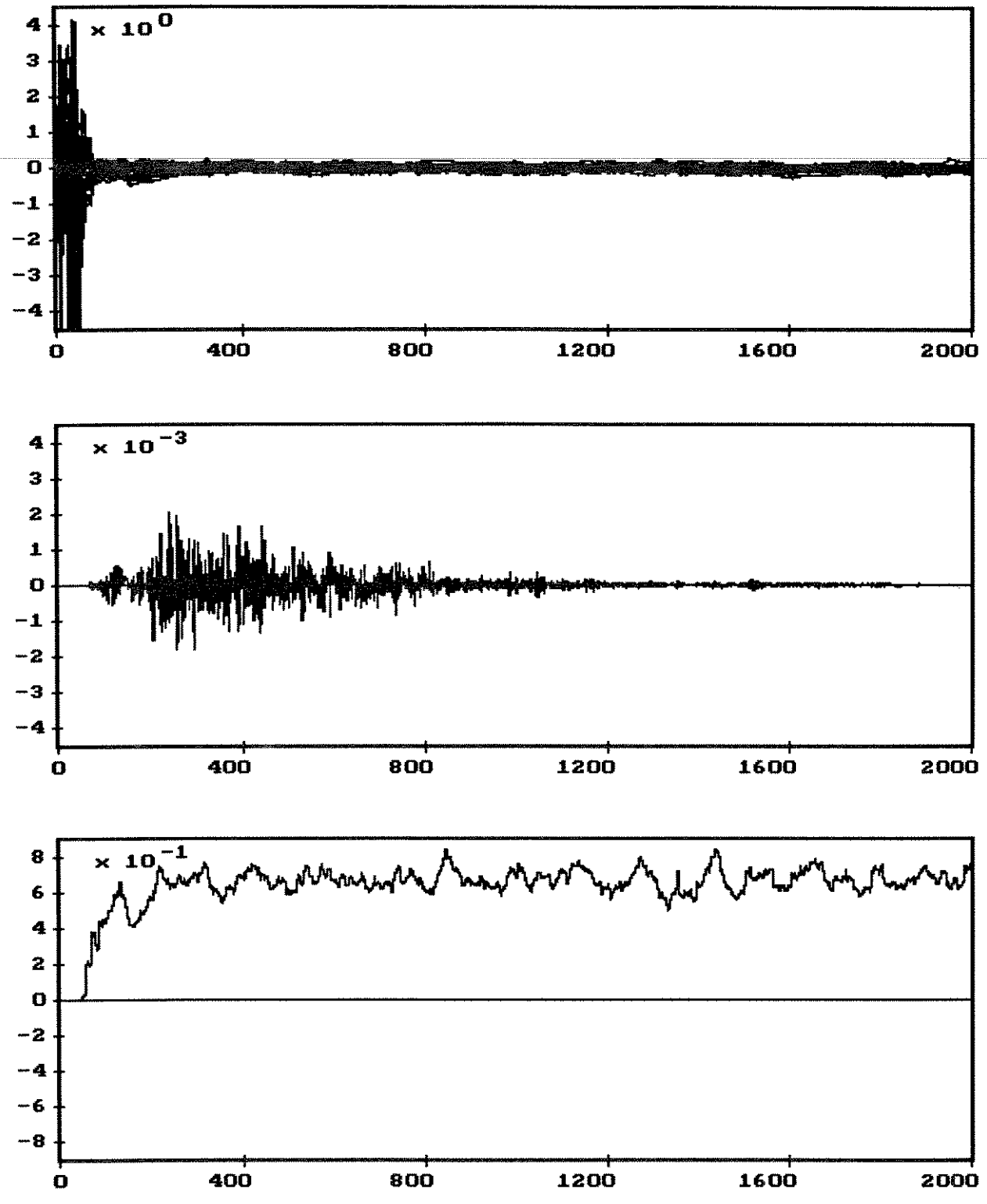


Figura 5.9 - Algoritmo estabilizado pelo novo método, com precisão de 23 bits, N = 40, w = 0,99 e tendo como sinal de entrada um ruído branco, gaussiano e de média nula. Em **a** observa-se os coeficientes do preditor progressivo, em **b**, a variável $\theta(n)$, e em **c**, a variável $\gamma(n)$. Nota-se que para $n < 40$, a ordem do filtro, tanto a variável de erro, quanto a variável de verossimilhança apresentam valores próximos de zero.

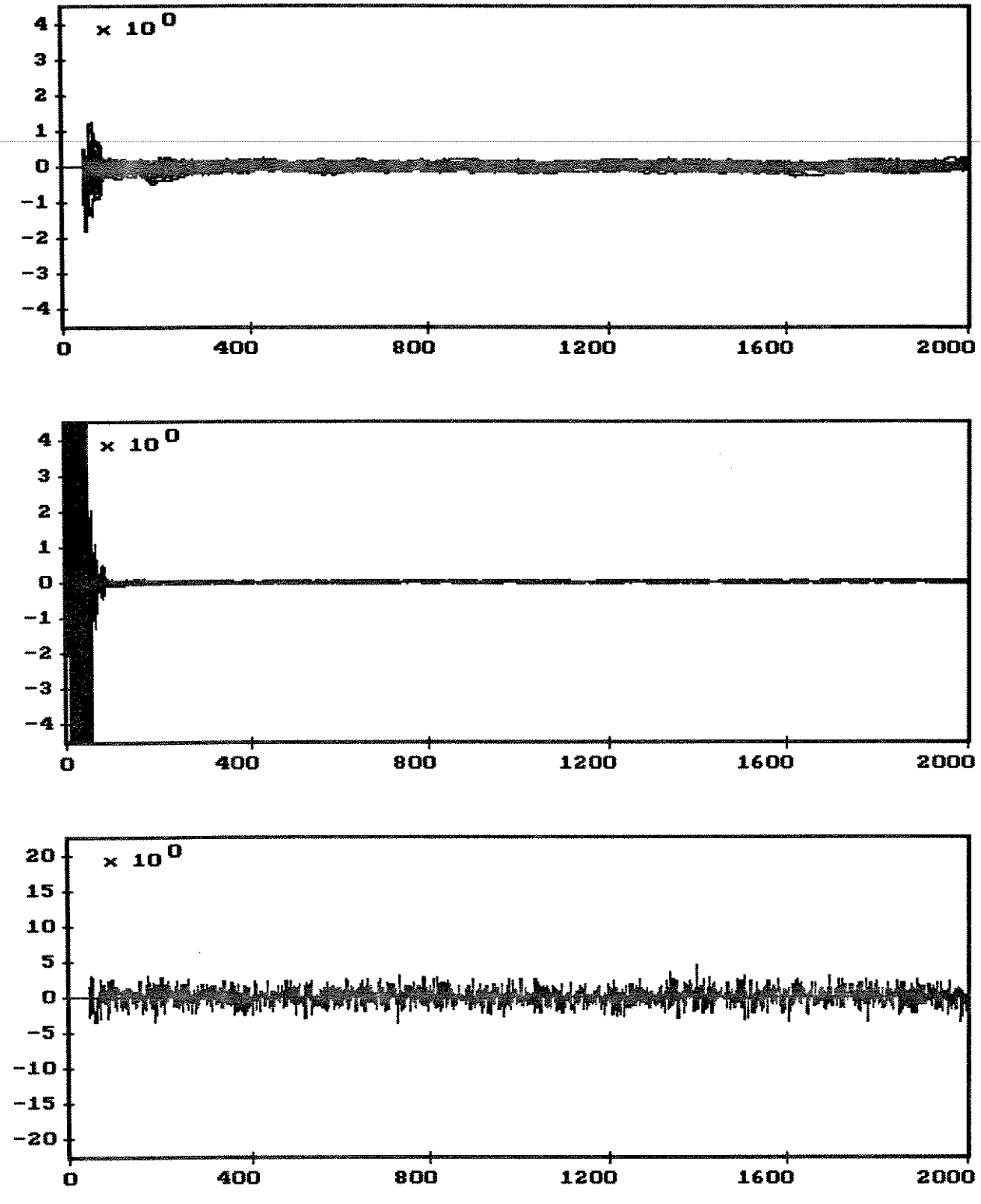


Figura 5.10 - Demais variáveis do processo ilustrado na Figura 5.9. Em **a** observa-se os coeficientes do preditor regressivo, B(n), em **b**, o vetor de ganho, G(n), e em **c**, o erro de predição regressiva a priori. No instante n = N todos os elementos do vetor B(n) assumem valores diferentes de zero.

O efeito observado na Figura 5.9b se repete sistematicamente, para o caso de valores grandes de N, nos três algoritmos estabilizados em questão. O período de atuação inicial da variável $\theta(n)$, onde ela apresenta valores altos, caracteriza uma fase crítica para a estabilidade numérica. É neste momento que o algoritmo encontra-se mais susceptível à manifestação da divergência. De fato, observa-se que quando o algoritmo ultrapassa esta fase sem que ocorra a divergência, o mais provável é que ele se mantenha estável indefinidamente.

Outro fator que contribui para o mau desempenho dos métodos de estabilização para os casos em que N é grande, está relacionado com o próprio algoritmo rápido original, cuja característica de estabilidade é mais deficiente, como visto no Capítulo 3. Por outro lado, a forma como os métodos atuam, somando em cada iteração um vetor com elementos de pequeno valor ao preditor regressivo, B(n), introduz um caráter estocástico ao processo de estabilização. Parece natural que havendo um número maior de coeficientes para serem corrigidos, este processo será dificultado.

A Tabela 5.3 mostra uma série de resultados, onde foram avaliados os limiares de estabilidade para os quatro algoritmos em função do número de coeficientes N. Estes dados foram obtidos para o caso de ruído branco gaussiano com variância unitária. Nota-se, nos quatro casos, uma acentuada queda de desempenho para valores de N acima de uma centena. O novo método de estabilização mostrou um desempenho ligeiramente melhor que os outros três para todos os valores de N. Para o caso de $N = 256$ o Algoritmo de Botto e Moustakides somente conseguiu manter a estabilidade para $w = 1$, enquanto que o Algoritmo de Benallal e Gilloire mostrou-se instável para $N > 32$, suportando, nestes casos, apenas algumas centenas de iterações.

Como forma de averiguar se os algoritmos de Botto e Moutakides, de Benallal e Gilloire e de Slock e Kailath podem atuar com mais eficiência nos casos de N grande, foram também testados outros valores para os parâmetros ρ e k_i, não se obtendo melhores resultados.

Tabela 5.3 - Comparação do limiar de estabilidade quanto à ordem do filtro

N \ Método	Botto /Moustides	Slock/Kailath	Benallal/Gilloire	Novo Método
N = 2	0,788	0,791	0,833	0,756
N = 4	0,876	0,887	0,904	0,875
N = 8	0,937	0,939	0,948	0,935
N = 16	0,968	0,969	0,973	0,967
N = 32	0,985	0,985	1,0	0,984
N = 64	0,994	0,999	instável	0,993
N = 128	0,998	instável	instável	0,996
N = 256	1,0	instável	instável	0,999

5.4 – ANÁLISE QUANTO AO SINAL DE EXCITAÇÃO

Os algoritmos estabilizados também são sensíveis à característica do sinal de entrada. No caso de sinais predizíveis, ainda valem os estudos feitos no Capítulo 3. Sobretudo, com respeito ao valor das energias dos erros de predição, que tendem a zero com o tempo. O anulamento do erro de predição regressiva, $E_b(n)$, impede a atuação do mecanismo de estabilização, pois ele é usado no cálculo de $\xi(n)$, ou $\theta(n)$ no novo método.

A forma mais simples de contornar o problema do anulamento de $E_b(n)$ consiste em adicionar, em cada iteração, uma constante positiva de pequeno valor a esta variável. Ou seja

$$E_b(n) = w\, E_b(n-1) + e_b^*(n)\varepsilon_b(n) + \Delta \tag{5.2}$$

O valor de Δ deve se situar em torno de 10^{-8} para o caso de uma implementação com precisão de 23 bits. Valores maiores podem comprometer o funcionamento do algoritmo, gerando instabilidade. Também a variável $E_a(n)$ deve receber cuidados especiais para evitar o decaimento com o tempo. Caso contrário, a ocorrência de overflow é inevitável. Como forma de impedir o anulamento de $E_a(n)$, sugerimos acrescentar no final do algoritmo a relação

$$E_a(n) = \frac{E_b(n)\, w^N}{\gamma(n)} \tag{5.3}$$

Este recurso, além de evitar o anulamento da energia do erro de predição progressiva, também possui a virtude de contribuir para a exatidão do método LS, uma vez que a variável é recalculada em termos do novo valor atribuído para $E_b(n)$ mantendo, dessa forma, a consistência entre as variáveis.

As equações (5.2) e (5.3) adicionam uma complexidade computacional de duas multiplicações e uma soma ao novo método de estabilização. Esta alteração é suficiente para manter a estabilidade do algoritmo no caso de sinais perfeitamente predizíveis, como por exemplo, uma soma de senóides. As Figuras 5.11 e 5.12 ilustram as variáveis vetoriais e escalares para o caso de uma simulação em que foi empregado um sinal deste tipo para $N = 4$ e $w = 0{,}99$. O evento de overflow, que ocorria após cerca de trinta mil iterações, foi totalmente suprimido quando efetuadas as alterações definidas em (5.2) e (5.3).

Como deve ser esperado, a estabilização dos algoritmos rápidos torna-se mais difícil na medida em que as amostras do sinal de entrada ficam mais correlacionadas. Para os quatro algoritmos testados, o limiar de estabilização piora um pouco com o aumento da correlação. Como exemplo, o algoritmo estabilizado pelo novo método, para o caso de $N = 4$ e $b = 23$, tem seu limiar aumentado de 0,875 para 0,960 enquanto o sinal varia de um ruído branco para um sinal perfeitamente predizível.

Outro fato notável, é que os algoritmos estabilizados apresentam resultados satisfatórios quando excitados por sinais predizíveis do tipo somatória de senóides, ainda que o número K de exponenciais complexas presentes na somatória seja menor que a ordem N do filtro. Neste caso, inclusive, os N - K últimos coeficientes do preditor progressivo tendem a se anular com o tempo, e o algoritmo passa a se comportar, em termos de estabilidade, de uma forma próxima a de um algoritmo com ordem K correspondente. Portanto, com melhor desempenho.

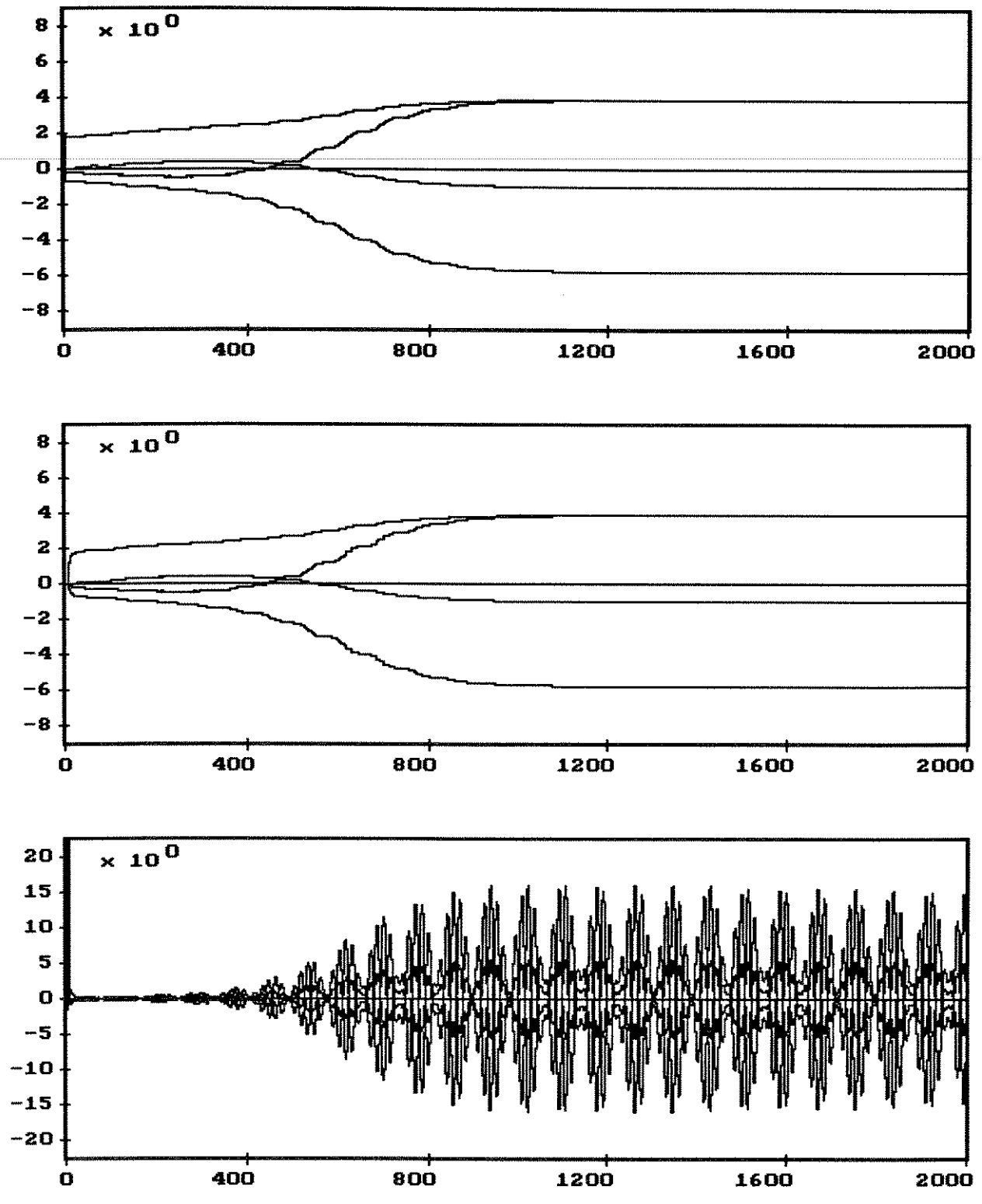


Figura 5.11 - Processo de predição usando o novo método de estabilização para o caso de um sinal de entrada dado por $x(n) = \text{sen}(n\pi/12) + \text{sen}(n\pi/17)$, $N = 4$ e um fator de esquecimento $w = 0{,}99$. Em **a** observa-se o preditor progressivo, A(n), em **b**, o preditor regressivo, B(n), e em **c**, o vetor de ganho G(n)

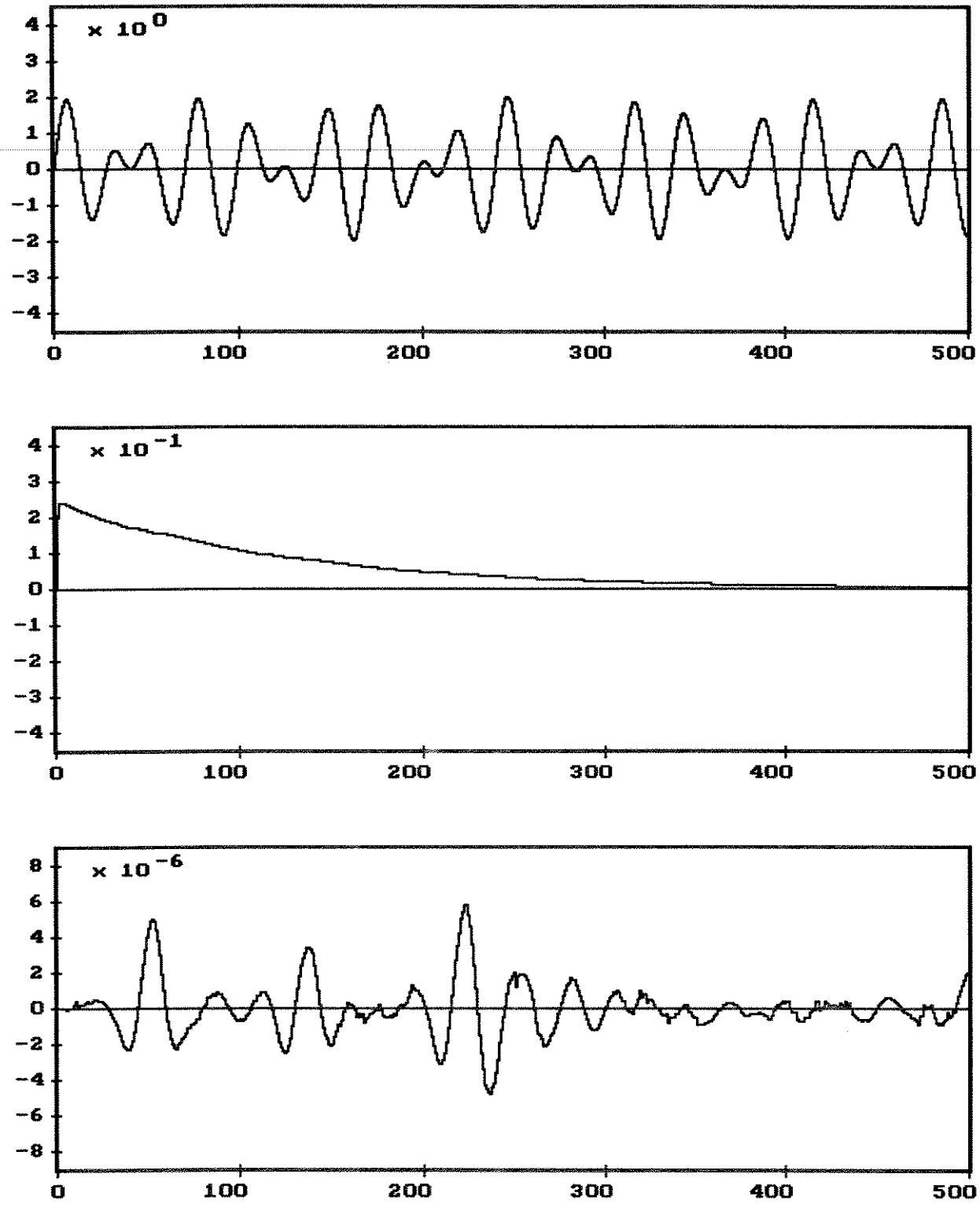


Figura 5.12 - Variáveis escalares para o mesmo processo da Figura 5.11. Em **a** observa-se o sinal de entrada, $x(n)$, em **b**, a energia do erro de predição progressiva, $E_a(n)$, e em **c**, o sinal de controle de erro, $\theta(n)$. O algoritmo utiliza as modificações definidas em (5.2) e (5.3) e executa milhões de iterações sem que haja qualquer manifestação de instabilidade.

5.5 – COMENTÁRIO

Neste capítulo foram testados, sob diversas condições experimentais, o novo método de estabilização dos algoritmos rápidos, apresentado na Seção 4.4, juntamente com dois outros métodos conhecidos. Como forma de trabalho, foi privilegiado o caso de sinais estacionários.

Pudemos constatar que o fator de esquecimento, principal elemento de instabilidade nos algoritmos rápidos originais, também exerce uma grande influência nos algoritmos estabilizados, ressaltando-se a existência de um limiar de estabilidade, cujo valor delimita duas regiões de operação. Na região superior, o algoritmo é numericamente estável. Na região inferior, ele fica ainda sujeito ao fenômeno da divergência, porém, com um notável prolongamento na vida dos processos.

Observamos que o comportamento dos algoritmos rápidos é alterado com a introdução do mecanismo de estabilização, devendo-se adotar novos critérios para a verificação do estado de normalidade do algoritmo. Este é um cuidado especialmente importante no caso dos algoritmos de Botto e Moustakides e de Benallal e Gilloire, onde existe o agravante de ocorrer um overflow ou a perda da capacidade de rastreio - problemas não observados no funcionamento do novo método.

Vimos que o aumento da precisão aritmética na implementação da variáveis, assim como no caso dos algoritmos rápidos originais, não melhora o desempenho quanto à estabilidade numérica, pois este é um fator que está diretamente relacionado com o comportamento dinâmico do algoritmo, e não com a acuidade numérica.

Deparamos com um resultado pouco animador quando constatamos a queda de desempenho dos algoritmos estabilizados com relação a ordem do filtro transversal. Este resultado, infelizmente, inviabiliza a implementação de filtros transversais com centenas de coeficientes - região de valores de N para os quais os algoritmos rápidos poderiam superar com enorme vantagem computacial o Algoritmo RLS.

Os algoritmos LS rápidos estabilizados operam satisfatoriamente quando excitados por sinais correlacionados, ou mesmo, por sinais predizíveis, necessitando para isso algumas pequenas adaptações. Este resultado é muito interessante, pois torna o critério LS recomendável para uma gama de aplicações onde o método do gradiente estocástico apresenta um desempenho deficiente.

Por fim, em todos os resultados comparativos ficou evidenciada a superioridade dos resultados obtidos pelo novo método. Esta constatação, aliada à pouca complexidade computacional exigida em sua implementação, permite que o indiquemos como a melhor solução para o problema de instabilidade numérica dos algoritmos LS rápidos.

5.6 – REFERÊNCIAS

[1] Benallal, A. "Etude des Algorithmes des Moindres Carres Transversaux Rapides et Application a L'Identification de Responses Impulsionnelles Acoustiques", Thèse présentée devant L'Universite de Rennes I, Janvier, 1989.

[2] Slock, D. T. M. and Kailath, T., "Numerically Stable Fast Transversal Filters for Recursive Least Squares Adaptive Filtering", IEEE Trans. on Signal Processing, vol. 39, pp. 92 - 113, January, 1991.

[3] Botto, J. L. and Moustakides, G. V., "Stabilizing the Fast Kalman Algorithms", IEEE Trans. Acoustics, Speech and Signal Processing, vol. ASSP-32, pp. 998 - 1005, October, 1984.

CONCLUSÃO

Inúmeros estudos apontam a superioridade do critério LS para a solução dos problemas de minimização encontrados na área de filtragem adaptativa. A emergência de um algoritmo de complexidade proporcional a N poderia consagrar a aplicação deste critério na aplicação de sistemas que operam em tempo real. Contudo, os algoritmos de mínimos quadrados rápidos, desde sua concepção, estiveram marcados pelo rótulo de serem numericamente instáveis.

Talvez a frase que melhor caracterize a conclusão deste trabalho seja: "Sim, é possível a utilização prática dos algoritmos de mínimos quadrados rápidos". Os subsídios teóricos e experimentais necessários a esta tarefa estão descritos nos cinco capítulos que compõe esta tese.

Uma série de contribuições é também apresentada ao longo de todo o trabalho. No Capítulo 3 é feito um estudo do processo de acumulação de erros nos algoritmos LS, onde procurou-se entender de uma forma simples e objetiva a causa da instabilidade nestes algoritmos. Como resultado, foi apresentado um modelo que descreve de forma satisfatória este processo para o caso de variáveis em ponto flutuante, em função da precisão aritmética, da ordem do filtro e do fator de esquecimento empregado.

Uma conclusão importante, e que foi diversas vezes ressaltada, é que apesar se ser o limite de precisão aritmética o principal causador de instabilidade, não é este o fator que deve ser combatido, e sim o comportamento dinâmico do algoritmo, que amplifica o ruído de arredondamento para níveis insuportáveis. Dessa forma, é improdutivo aumentar a precisão das variáveis na tentativa de obter estabilidade.

A constatação de que os métodos de estabilização não conseguem solucionar o problema de instabilidade numérica para o caso de filtros com ordem grande, não deve ser considerada um desestímulo. Ainda que seja este o caso em que os algoritmos rápidos poderiam substituir com

enormes vantagens o Algoritmo RLS, existe toda uma região de valores, abaixo de uma centena, para os quais os algoritmos rápidos constituem um considerável avanço em relação ao esforço proporcional a N^2 exigido pelo algoritmo RLS.

No Capítulo 4 são apresentados os principais métodos existentes para a estabilização dos algoritmos rápidos, além do que é introduzido um método novo, cujo principal mérito é o de apresentar um desempenho comparável aos já existentes, e muitas vezes superior, com um adicional de complexidade muito pequeno.

Foi também verificado que os métodos de estabilização apresentam regiões de funcionamento, não constituindo uma solução generalizada para o problema. A solução ideal, capaz de atender a quaisquer valores do fator de esquecimento e da ordem do filtro parece algo impraticável. De qualquer forma, o novo método, apresentado na Seção 4.4, apresenta características superiores aos métodos existentes.

No Capítulo 5 foram testados os algoritmos estabilizados sob diversas condições, especialmente quando excitados por sinais estacionários. Como resultado, conclui-se que existem combinações de valores para o fator de esquecimento, e para o número de coeficientes do filtro transversal, que apresentam interesse prático, onde é perfeitamente possível obter um algoritmo estabilizado e de operação confiável.

Para se ter uma idéia da viabilidade do uso dos algoritmos rápidos em sistemas práticos, tomemos como exemplo uma transmissão de dados a uma taxa de transferência de 64 Kbps. Usando uma unidade de processamento que permita executar 10 milhões de operações em ponto flutuante por segundo, que é um recurso plausível em nossa época, teremos uma disponibilidade de aproximadamente 156 operações por amostra, ou por iteração do algoritmo. Com um método que exija um esforço computacional de 8N, como o novo método apresentado na Seção 4.4, teremos que o filtro poderá ter até 18 coeficientes, o que é bastante para o tratamento de 280 μS de sinal. Este é o tempo que um sinal elétrico leva para percorrer 56 Km em um condutor. Como os lances de retransmissão são, em geral, de uns poucos quilômetros, vemos que é perfeitamente possível a aplicação de um algoritmo LS rápido no tratamento de ecos em linhas de transmissão de 64 Kbps. E isto sem explorar as possibilidades de processamento paralelo que o algoritmo oferece.