WEMERSON DELCIO PARREIRA

COMPORTAMENTO ESTOCÁSTICO DO ALGORITMO KERNEL LEAST-MEAN-SQUARE

FLORIANÓPOLIS

2012

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

COMPORTAMENTO ESTOCÁSTICO DO ALGORITMO KERNEL LEAST-MEAN-SQUARE

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Doutor em Engenharia Elétrica

WEMERSON DELCIO PARREIRA

Florianópolis, Abril de 2012

COMPORTAMENTO ESTOCÁSTICO DO ALGORITMO KERNEL LEAST-MEAN-SQUARE

WEMERSON DELCIO PARREIRA

Esta Tese foi julgada adequada para obtenção do Título de Doutor em Engenharia Elétrica, Área de concentração Comunicações e Processamento de Sinais, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.

José Carlos M. Bermudez, PhD
Orientador

Patrick Kuo Peng, Dr
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

José Carlos M. Bermudez, Ph.D., UFSC
Presidente

Clodoaldo Aparecido M. Lima, Dr., USP

Joceli Mayer, Ph.D., UFSC

Leonardo Silva Resende, Dr., UFSC

Márcio Holsbach Costa, Dr., UFSC

Vitor Heloiz Nascimento, Ph.D., USP

Aos meus pais e ao Edson.

AGRADECIMENTOS

A Deus, pela bençãos diárias.

Ao meu orientador, professor José Carlos M. Bermudez, pela confiança, pelo apoio, pela paciência e pela dedicação a minha formação.

Aos professores Cédric Richard – *Université de Nice Sophia-Antipolis*, *Institut Universitaire de France* e Jean-Ives Tourneret – *Université de Toulouse*, CNRS, *France* pelas valiosas contribuições dadas ao desenvolvimento desta tese e a minha formação crítico-científica.

Aos meus pais Antônio D. Parreira e Rosa Maria G. Parreira, pelo amor, carinho e dedicação em todos os momentos de minha vida. Aos meus adoráveis irmãos Anderson D. Parreira e Wesley L. Parreira pelo apoio e incentivo. Aos meus queridos sobrinhos Geovanna e Henrique e à minha madrinha Aparecida pelo amor, carinho e compreensão.

Aos meus familiares que compreenderam todas as minhas ausências e me incentivaram durante todo o tempo.

À todos da família Passig e Barth que me acolheram com amor e carinho, mas principalmente aos queridos Edson Passig, Irma Passig, Adriana Passig, Magrid Passig, *nonna* Elza Barth, Marli P. Barth e Jonathan Barth pelo apoio e pelas positivas palavras nos momentos mais difíceis.

Aos meus amigos João C. Dovicchi e Julcimar B. Archer pela amizade e pelo suporte necessário para o início desta fantástica jornada.

Aos professores Márcio Hosbach Costa e Joceli Mayer, EEL – UFSC, pela grande contribuição à minha formação.

Ao professor Manuel R. O. Lino, INE – UFSC, pelos valiosos ensinamentos.

Aos colegas do LPDS, Daniel M. Montezano, Luiz F. Silva, Marcos H. Maruo, Osmando Pereira Jr., Renata C. Borges, Tales C. O. Imbiriba, Vitor B. Nicolau e Yasmin M. Maluenda pela amizade, pelas valiosas discussões e

construtivas críticas.

Às secretárias do LPDS Fernanda Leal e Ana Catarina e aos funcionários PGEEL Wilson e Marcelo pelo apoio na resolução de questões administrativas.

Ao CNPq pelo apoio financeiro.

E a todos aqueles que de alguma forma contribuíram para o desenvolvimento deste trabalho.

“Our chief want in life, is somebody who shall make us do what we can.”
(“Nosso maior desejo na vida é encontrar alguém que nos leve a fazer o melhor que pudermos.”)

Ralph Waldo Emerson, 1860.

Resumo da Tese apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Doutor em Engenharia Elétrica.

COMPORTAMENTO ESTOCÁSTICO DO ALGORITMO KERNEL LEAST-MEAN-SQUARE

Wemerson Delcio Parreira

Abril / 2012

Orientador: José Carlos M. Bermudez, PhD.
Área de Concentração: Comunicações e Processamento de Sinais.
Palavras-chave: Filtragem Adaptativa, KLMS, análise da convergência, sistema não-linear, kernels reprodutivos.
Número de Páginas: 266

Algoritmos baseados em *kernel* têm-se tornado populares no processamento não-linear de sinais. O processamento não-linear aplicado sobre um sinal pode ser modelado como um processamento linear aplicado a um sinal transformado para um espaço de Hilbert com *kernels* reprodutivos (RKHS). A operação linear no espaço transformado pode ser implementada com baixa complexidade e pode ser melhor estudada e projetada. O algoritmo Kernel Least-Mean-Squares (KLMS) é um algoritmo popular em filtragem adaptativa não-linear devido à sua simplicidade e robustez. Implementações práticas desse algoritmo requerem um modelo de ordem finita do processamento não-linear, o que modifica o comportamento do algoritmo em relação ao LMS simplesmente mapeado para o RKHS. Essa modificação leva à necessidade de novos modelos analíticos para o comportamento do algoritmo. O desempenho do algoritmo é função do passo de convergência e dos parâmetros do *kernel* empregado. Este trabalho estuda o comportamento do KLMS em regimes transitório e permanente para entradas Gaussianas e um modelo de não-linearidade de ordem finita. Dois *kernels* são considerados; o Gaussiano e o Polinomial. Derivamos modelos analíticos recursivos para os comportamentos do vetor médio de erros nos coeficientes e do erro quadrático médio de estimação. As previsões do modelo mostram excelente acordo com simulações de Monte Carlo no transitório e no regime permanente. Isso permite a determinação explícita das condições para a estabilidade, e permite escolher os parâmetros do algoritmo a fim de obter um desempenho desejado. Exemplos de projeto são apresentados para o *kernel* Gaussiano e para o *kernel* Polinomial de segundo grau de forma a validar a análise teórica e ilustrar sua aplicação.

Abstract of Thesis presented to UFSC as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

STOCHASTIC BEHAVIOR OF THE KERNEL LEAST-MEAN-SQUARE ALGORITHM

Wemerson Delcio Parreira

Abr / 2012

Advisor: José Carlos M. Bermudez, PhD.
Area of Concentration: Digital Signal Processing.
Keywords: Adaptive Filtering, KLMS, convergence analysis, nonlinear system, reproducing kernel.
Number of pages: 266

Kernel-based algorithms have become popular in nonlinear signal processing. A nonlinear processing can be modeled as a linear processing applied to a signal transformed to a reproducing kernel Hilbert space (RKHS). The linear operation in the transformed space can be implemented with low computational complexity and can be more easily studied and designed. The Kernel Least-Mean-Squares (KLMS) is a popular algorithm in nonlinear adaptive filtering due to its simplicity and robustness. Practical implementations of this algorithm require a finite order model for the nonlinear processing. This modifies the algorithm behavior as compared to the LMS simply mapped to the RKHS. This modification leads to the need for new analytical models for the algorithm behavior. The algorithm behavior is a function of both the step size and the kernel parameters. This work studies the KLMS algorithm behavior in transient and in steady-state for Gaussian inputs and for a finite order nonlinearity model. Two kernels are considered; the Gaussian and the Polinomial. We derive analytical models for the behavior of both the mean weight error vector and the mean-square estimation error. The model predictions show excellent agreement with Monte Carlo simulations at both the transient and the steady-state. This allows the explicit determination of the stability limits and to design the algorithm parameters to obtain a desired performance. Design examples are presented for the Gaussian and for the second degree Polinomial kernels to validate the analysis and to illustrate its application.

LISTA DE FIGURAS

2.1 Sistema não-linear. 40

2.2 Estimação de sistemas baseado no algoritmos adaptativo KLMS. 45

2.3 Filtro linear com vetores das características no espaço das características. 46

2.4 Filtro linear de ordem finita. 48

3.1 Estimação de sistemas baseado em algoritmos adaptativos "kernelizados". 56

5.1 Diagrama em blocos das diretrizes para um projeto de filtro usando o algoritmo KLMS para um dado kernel. 100

5.2 Modelo teórico e simulações (Monte Carlo) do KLMS gaussiano usando diferentes parâmetros para o Exemplo 1. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) e (3.57). 105

5.3 Resultados do KLMS gaussiano em regime permanente para o Exemplo 1. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente. 106

5.4 Modelo teórico e simulações (Monte Carlo) do KLMS polinomial usando diferentes parâmetros para o Exemplo 1. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) e (3.57). 107

5.5 Resultados do KLMS polinomial em regime permanente para o Exemplo 1. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente. 108

5.6 Modelo teórico e simulações (Monte Carlo) do KLMS gaussiano usando diferentes parâmetros para o Exemplo 2. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) e (3.57). 115

5.7 Resultados do KLMS gaussiano em regime permanente para o Exemplo 2. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente. 116

5.8 Modelo teórico e simulações (Monte Carlo) do KLMS polinomial usando diferentes parâmetros para o Exemplo 2. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) and (3.57). 117

5.9 Resultados do KLMS polinomial em regime permanente para o Exemplo 2. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente. 118

5.10 Modelo teórico e simulações (Monte Carlo) do KLMS gaussiano usando diferentes parâmetros para o Exemplo 3. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) e (3.57). 119

5.11 Resultados do KLMS gaussiano em regime permanente para o Exemplo 3. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente. 120

5.12 Modelo teórico e simulações (Monte Carlo) do KLMS polinomial usando diferentes parâmetros para o Exemplo 3. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) e (3.57). 121

5.13 Resultados do KLMS polinomial em regime permanente para o Exemplo 3. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente. 122

A.1 Conjunto X de sinais 134

F.1 Comportamento do MSE mínimo com $J_{\min}(\xi) < J_{\max}$. . . . 161

F.2 Comportamento do comprimento médio do dicionário em que $J_{\min}(\xi) < J_{\max}$. 162

F.3 Comportamento do regime permanente do MSE em que $J_{ms}(\infty) < J_{\max}$. 162

F.4 Comportamento do regime permanente do MSE em excesso em que $J_{ms}(\infty) < J_{\max}$. 162
F.5 Número de iterações necessárias para a convergência em que $J_{ms}(\infty) < J_{\max}$. 163
F.6 Comportamento do KLMS para diferentes parâmetros do *kernel* gaussiano. A curva em vermelho representa o modelo teórico e a curva em azul representa as simulações de 500 realizações de Monte Carlo. 163
F.7 Comportamento do modelo teórico do KLMS gaussiano. A curva contínua em vermelho representa o MSE mínimo, a curva contínua em azul representa MSE, a curva em preto representa o MSE em excesso e a curvas pontilhadas representa os seus respectivos regime permanente. 164

LISTA DE TABELAS

3.1 Restrições em η obtidas a partir de (3.45b) 71
3.2 Restrições em η obtidas a partir de (3.52b) 73
4.1 Restrições em η obtidas a partir de (4.41) e (4.44) 94
5.1 Limites de estabilidade do Exemplo 1 usando KLMS gaussiano 102
5.2 Limites de estabilidade do Exemplo 1 usando KLMS polinomial 103
5.3 Resumo dos resultados simulados para o Exemplo 1 usando o kernel gaussiano. 104
5.4 Resumo dos resultados simulados para o Exemplo 1 usando o kernel polinomial. 104
5.5 Limites de estabilidade do Exemplo 2 usando KLMS gaussiano 109
5.6 Limites de estabilidade do Exemplo 2 usando KLMS polinomial. 109
5.7 Resumo dos resultados simulados para o Exemplo 2 usando o kernel gaussiano. 110
5.8 Resumo dos resultados simulados para o Exemplo 2 usando o kernel polinomial. 110
5.9 Limites de estabilidade do Exemplo 3 usando KLMS gaussiano. 111
5.10 Limites de estabilidade do Exemplo 3 usando KLMS polinomial. 112
5.11 Resumo dos resultados simulados para o Exemplo 3 usando o kernel gaussiano. 112
5.12 Resumo dos resultados simulados para o Exemplo 3 usando o kernel polinomial. 113
F.1 Resumo dos resultados apresentados 161

LISTA DE ABREVIATURAS E SIGLAS

ALD	Dependência Linear Aproximada.
BP	*Basis Pursuit.*
DSP	Processador digital de sinais (*digital signal processor*).
i.i.d.	Independente e identicamente distribuído.
KAPA	*Kernelized Affine Projection* .
KLMS	*Kernel Least-Mean-Square.*
KNLMS	KLMS Normalizado.
KRLS	*Kernel Recursive-Least-Mean-Square.*
LMS	*Least Mean-Squares.*
MIA	Hipótese da independência modificada (*modified independence assumption*).
MP	*Matching Pursuit.*
MSE	Erro Quadrático Médio (*Mean-Square Error*).
OMP	*Orthogonal Matching Pursuit.*
RKHS	Espaço de Hilbert de *kernel* reprodutivo.
RNAs	Redes Neurais Artificiais.
SVMs	*Support Vector Machines.*
w.s.	Problema sem solução (*without solution*).
WGA	*Weak Greedy Algorithms.*

LISTA DE SÍMBOLOS

$\varphi(.)$	Função não-linear
$\kappa(.,.)$	Funções *kernel*
$\mathcal{U}$	Subespaço compacto de $\mathbb{R}^q$
$z(n)$	Ruído aditivo
$\boldsymbol{u}(n)$	Sinal de entrada
$d(n)$	Saída desejada
$\psi(\cdot)$	Função baseada em kernel
α	Vetor de coeficientes
$\boldsymbol{K}$	Matriz Gram do *kernel*
$\mathcal{D}$	Dicionário de funções *kernel*
$\#\mathcal{D}$	Cardinalidade do dicionário
ϵ_0	Nível de esparsidade do modelo
ε_0	Nível de coerência do dicionário do dicionário
$\mathcal{C}$	Número de ciclos do processador
ξ	Parâmetros do *kernel* gaussiano e laplaciano
β	Grau do kernel polinomial
a	Termo constante do *kernel* polinomial
$\mathcal{H}$	Espaço de Hilbert
$e(n)$	Erro de estimação
η	Passo do algoritmo adaptativo
$\varphi(\boldsymbol{u}(n))$	Saída do sistema não-linear
$\boldsymbol{R_{uu}}$	Matriz de autocorrelação de $\boldsymbol{u}(n)$
$\kappa_\omega(n)$	Vetor de *kernel* no instante n
$\kappa(\boldsymbol{u}(.,\omega_i))$	*i*-ésima componente do dicionário
$\alpha(n)$	Vetor de coeficientes do KLMS
$d(n)$	Sinal desejado
$\hat{d}(n)$	Estimativa do sinal desejado

$J_{ms}(n)$	Erro Quadrático Médio – MSE
$\boldsymbol{R}_{\kappa\kappa}$	Matriz de autocorrelação da entra $\boldsymbol{u}(n)$ "kernelizada"
$\boldsymbol{p}_{\kappa d}$	Vetor de correlação cruzada entre $\kappa_\omega(n)$ and $d(n)$
r_{md}	Componente da diagonal principal de $\boldsymbol{R}_{\kappa\kappa}$
r_{od}	Componente fora da diagonal principal de $\boldsymbol{R}_{\kappa\kappa}$
α_{opt}	Vetor de peso ótimo
J_{min}	Mínimo MSE
$\boldsymbol{v}(n)$	Vetor de erro nos coeficientes
e_0	Erro de estimação ótima
$\boldsymbol{C_v}(n)$	Matriz de correlação do vetor de erro nos coeficientes
$J_{ex}(n)$	MSE em excesso
μ_k	Momentos de quarta ordem
$\boldsymbol{H}^{ij}$	(i,j)-ésima representante de uma família de funções
$\boldsymbol{G}$	Matriz transição de estado
$\boldsymbol{h}^{\ell p}$	Representação lexicográfica de $\boldsymbol{H}^{\ell p}$
$\boldsymbol{c_v}(n)$	Representação lexicográfica de $\boldsymbol{C_v}(n)$
$r_{\kappa\kappa}$	Representação lexicográfica de $\boldsymbol{R}_{\kappa\kappa}$
$\boldsymbol{c_v}(\infty)$	Regime permanente do vetor $\boldsymbol{c_v}(n)$
n_ϵ	Número de iterações necessárias para convergência
ϵ	Parâmetro de projeto que representa acurácia da estimação do tempo de convergência
$\boldsymbol{C_v}(\infty)$	Matriz correlação do vetor de erro nos coeficientes em regime permanente
$\boldsymbol{T}(\infty)$	Regime permanente da matriz $\boldsymbol{T}(n)$
$J_{ms}(\infty)$	Regime permanente do erro quadrático médio
trace$\{\boldsymbol{R}_{\kappa\kappa}\,\boldsymbol{C_v}(\infty)\}$	Traço da matriz $\{\boldsymbol{R}_{\kappa\kappa}\,\boldsymbol{C_v}(\infty)\}$
$J_{ex}(\infty)$	Regime permanente do erro quadrático médio em excesso
c_{md}	Componente da diagonal principal de $\boldsymbol{C_v}(\infty)$
c_{od}	Componente fora da diagonal principal de $\boldsymbol{C_v}(\infty)$

ξ	Parâmetro do *kernel* gaussiano
$\|\cdot\|$	Norma ℓ_2 ou Euclidiana
$\boldsymbol{I}$	Matriz identidade($q \times q$)
$\boldsymbol{O}$	Matriz nula de ordem ($q \times q$)
$\boldsymbol{y}$	Vetor com distribuição gaussiana de média nula
$\boldsymbol{R}_y$	Matriz covariância de $\boldsymbol{y}$
Φ_z	Função geradora de momentos
$\boldsymbol{R}_\ell$	Matriz ($\ell q \times \ell q$) de correlação do vetor $\boldsymbol{y}_\ell$
$\boldsymbol{I}_\ell$	Matriz identidade de ordem($\ell q \times \ell q$)
det$\{\cdot\}$	Determinante da matriz
$J_{\max}$	Máximo MSE
$\eta_{\max}$	Máximo passo do algoritmo KLMS
ε_0^{GK}	Nível de coerência para o *kernel* gaussiano
$\mathcal{P}_{GK}$	Conjunto de parâmetros do *kernel* gaussiano
ε_0^{PK}	Nível de coerência para o *kernel* polinomial
$\mathcal{P}_{PK}$	Conjunto de parâmetros do *kernel polinomial*
$\eta_{\max}^{GD}$	Passo máximo do KLMS obtido a partir da análise dos discos de Gerschgorin
$\eta_{\max}^{EG}$	Passo máximo do KLMS obtido a partir da análise dos autovalores de $\boldsymbol{G}$
$n_{\epsilon_{GK}}$	Número de iterações necessárias para a convergência do KLMS gaussiano
$n_{\epsilon_{PK}}$	Número de iterações necessárias para a convergência do KLMS polinomial

SUMÁRIO

Introdução 33

1.1 Aplicações envolvendo os kernels 34

1.2 Desenvolvimento do trabalho 36

1.3 Trabalhos publicados 38

Problema de Estimação de sistemas não-lineares 39

2.1 Introdução 39

2.2 Redes Neurais 40

2.3 Filtros Polinomiais 42

2.4 Estimação não-linear usando kernel 44

2.4.1 Esparsificação de dicionários 48

2.4.2 Algoritmos adaptativos “kernelizados” 51

2.4.3 Exemplos de kernels 52

2.5 Conclusão e proposta de tese 53

Modelagem estocástica do algoritmo KLMS 55

3.1 Introdução 55

3.2 Solução de Wiener 56

3.3 Análise do comportamento do transitório do algoritmo KLMS 59

3.3.1 Hipóteses estatísticas simplificadoras 60

3.3.2 Comportamento médio dos coeficientes 60

3.3.3 Erro quadrático médio – MSE 61

3.3.4 Comportamento dos momentos de segunda ordem do vetor de erros 61

3.3.5 Algumas desigualdades úteis 65

3.4 Análise de convergência do KLMS para kernels positivos . . 66

3.4.1 Condições de convergência para kernels de valores positivos 68
3.4.2 Análise do algoritmo em regime permanente 74
3.5 Conclusões 76

Análise estatística dos kernels gaussiano e polinomial 79
4.1 Introdução 79
4.2 Análise estatística do kernel gaussiano 79
4.2.1 Momentos de segunda ordem 80
4.2.2 Momentos de quarta ordem 81
4.2.3 Relação entre os momentos de quarta ordem do kernel gaussiano 83
4.3 Análise estatística do kernel polinomial 84
4.3.1 Momento de primeira ordem 84
4.3.2 Momentos de ordens superiores 86
4.4 Análise de convergência do KLMS para o caso geral 89
4.4.1 Estudo do sinal dos momentos de segunda e quarta ordem do kernel 89
4.4.2 Condições para convergência quando $\mu_2 < 0$ 92
4.5 Conclusões 94

Diretrizes de projeto, simulações e resultados 97
5.1 Introdução 97
5.2 Diretrizes de projeto do KLMS usando kernel gaussiano e polinomial 98
5.3 Simulações e resultados 101
5.3.1 Exemplo 1 101
5.3.2 Exemplo 2 104
5.3.3 Exemplo 3 110
5.4 Conclusões 113

Conclusões e propostas de continuidade da pesquisa 123
6.1 Conclusões e contribuições 123
6.2 Propostas de continuidade da pesquisa 125

Anexo A – Representação de dados usando kernel 133

Anexo B – Kernel definido positivo e propriedades 137
B.1 Origem do termo kernel 138
B.2 Kernel como produto interno 139
B.2.1 O “truque” do kernel 146
B.3 Espaço Hilbertiano de kernel reprodutivo – RKHS 147

Anexo C – Matriz de autocorrelação do kernel é uma matriz definida positiva 149

Anexo D – Análise do sinal das componentes da matriz $\boldsymbol{G}$ 151

Anexo E – Análise da estabilidade nos casos em que $M = 1$ e $M = 2$ . . 153

Anexo F – Exemplo de estimação de sistema com estatísticas conhecidas 155
F.1 Descrição do Sistema 155
F.2 Cálculo das estatísticas da saída desejada – $d(n)$ 155
F.3 Condições de projeto e Resultados 159

Anexo G – Análise estatística do kernel polinomial do segundo grau 165
G.1 Momento de segunda ordem 165
G.1.1 Autocorrelação 166
G.1.2 Correlação cruzada 167
G.2 Momentos de quarta ordem 168
G.2.1 Cálculo de μ_1 168
G.2.2 Cálculo de μ_2 172
G.2.3 Cálculo de μ_3 174

G.2.4 Cálculo de μ_4 175
G.2.5 Cálculo de μ_5 177

Anexo H – Programas usados 179
H.1 Kernel gaussiano e polinomial 179
H.2 Kernel gaussiano 201
H.3 Kernel polinomial 234

1 INTRODUÇÃO

A filtragem adaptativa tem sido largamente utilizada nas últimas décadas em diversas áreas de aplicação. Sua principal característica é a possibilidade de ajuste ótimo dos parâmetros de filtragem na ausência de uma informação estatística prévia dos sinais envolvidos (SAYED, 2003) e (HAYKIN, 2002). Modelos lineares são ainda rotineiramente usados devido a sua inerente simplicidade, do ponto de vista conceitual e de implementação. Entretanto, em muitas situações práticas, tais como, identificação, predição e controle aplicados a áreas de comunicações e engenharia biomédica, um processamento não-linear do sinal se faz necessário. Uma extensa bibliografia (GIANNAKINS; SERPEDIN, 2001) dedicada à teoria de sistemas não-lineares. Diferentemente do caso de sistemas lineares e invariantes no tempo, os quais são completamente caracterizados por suas respostas ao impulso, não existe um contexto matemático simples para descrever os sistemas não-lineares. Filtros polinomiais, comumente chamados de séries de Volterra (MATHEWS; SICURANZA, 2000), e redes neurais (HAYKIN, 1994), são os mais populares e estudados modelos não-lineares para filtragem adaptativa.

De maneira alternativa (ARONSZAJN, 1950), (AIZERMAN; BRAVERMAN; ROZONOER, 1964) e (KIMELDORF; WAHBA, 1971), verificaram progressos em métodos de aproximação de funções baseados em espaços de Hilbert de *kernels* reprodutivos (RKHS)[1] (HERBRICH, 2002), (SCHÖLKOPF; SMOLA, 2002) incluindo, por exemplo, *Support Vector Machines* (SMOLA; SCHÖLKOPF, 1998). Uma característica comum dos métodos baseados em *kernels*, entretanto, é a utilização de matrizes e modelos cujas dimensões são iguais ao número de dados avaliados. Os métodos baseados em *kernel*, em particular as *Support Vector Machines* (SVMs), são aplicados em muitos problemas do mundo real, sendo considerados “estados

[1] Veja definição de Espaços RKHS no Apêndice B.3

da arte" em vários domínios (VERT; TSUDA; SCHÖLKOPF, 2004). Apenas recentemente, entretanto, tais métodos têm-se tornado uma tendência em *machine learning* e inferência bayesiana.

1.1 Aplicações envolvendo os kernels

Os *kernels* são aplicados com sucesso em um vasto número de problemas reais e são considerados estado da arte em vários domínios. A história dos métodos baseados em *kernels* definidos positivos pode ser contada voltando algumas décadas, com Aronszajn (1950) e Parzen (1962) com alguns empregos de *kernels* em estatística, (ARONSZAJN, 1950) e (PARZEN, 1962). Posteriormente, Aizerman e outros (1964) usaram *kernels* definidos positivos num contexto mais próximo do que a comunidade científica denomina hoje de "truque do *kernel*" (AIZERMAN; BRAVERMAN; ROZONOER, 1964). Boser, trinta anos mais tarde, usou o "truque do *kernel*" para construir as SMVs (*Support Vector Machines*), uma generalização do assim chamado algoritmo do hiperplano ótimo (BOSER; GUYON; VAPNIK, 1992). Inicialmente, eles pensavam que a ação principal das SMVs comparada ao algoritmo do hiperplano ótimo estava no fato delas permitirem o uso de uma classe maior de medidas de similaridade. Na época, usavam apenas dados vetoriais, como no caso dos hiperplanos ótimos.

Em 1997, Schölkopf observou que a aplicação de *kernels* não só aumentava a classe de medidas de similaridades mas também poderia ser estendida a dados não vetoriais (SCHÖLKOPF, 1997). Isto é devido ao fato do *kernel* fornecer uma representação vetorial dos dados no espaço das características. Os primeiros exemplos destes *kernels* não triviais foram apresentados por Haussler (HAUSSLER, 1999) e Watkins (WATKINS, 2000). Além disso, Schölkopf, mostrou que os *kernels* podem ser usados para construir generalização de alguns algoritmos que podem ser escritos em termos de produto interno (SCHÖLKOPF; SMOLA; MÜLLER, 1998). Nos últimos anos

muitos trabalhos utilizando algoritmos “kernelizados” foram apresentados à comunidade científica (WESTON et al., 2002; GIROLAMI, 2002; KUSS; GRAEPEL, 2003).

A ideia básica dos algoritmos baseados em *kernels* é transformar um dado $\boldsymbol{x}$ do espaço de entrada em um vetor $\phi(\boldsymbol{x})$ pertencente a um espaço denominado espaço das características. Nesse espaço o produto interno pode ser calculado usando uma função *kernel* definida positiva, que satisfaz a condição de Mercer, (SCHÖLKOPF; SMOLA, 2002):

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$$

em que $\langle \, ., . \, \rangle$ denota o produto interno usual. Essa ideia simples nos permite encontrar versões não-lineares de algoritmos lineares que possam ser expressos em termos de produtos internos sem mesmo conhecer o mapeamento ϕ. Uma propriedade interessante do espaço das características é que o RKHS, o espaço gerado pelas funções $\{\kappa(., \boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X}\}$, define um único funcional no espaço de Hilbert. Essa propriedade dá origem à propriedade reprodutiva dos *kernels* (SCHÖLKOPF; SMOLA, 2002).

A aplicação de algoritmos “kernelizados” em aprendizagem em tempo real ou filtragem adaptativa impõe a utilização de modelos de ordem finita. Diversos algoritmos foram propostos para limitar a dimensão dos modelos específicos no contexto de filtragem adaptativa baseada em *kernels* (ENGEL; MANNOR; MEIR, 2004), (DODD; KADIRKAMANATHAN; HARRISON, 2003), (DODD; MITCHINSON; HARRISON, 2003). Cada um desses consiste na adição de um procedimento extra de esparsificação ao processo de filtragem adaptativa. Esses algoritmos de duas etapas levam a uma complexidade ainda excessiva para o processamento em tempo real. Técnicas de baixo custo computacional baseadas na função coerência foram recentemente propostas (HONEINE; RICHARD; BERMUDEZ, 2007).

Escolher o algoritmo e os parâmetros do modelo não-linear (parâme-

tros do *kernel*) para alcançar um desempenho prescrito era uma questão em aberto, e exige uma extensa análise do comportamento estocástico do algoritmo. Nosso trabalho traz uma nova contribuição para a discussão sobre filtragem adaptativa baseada em *kernel*, fornecendo uma primeira análise de desempenho do algoritmo KLMS.

1.2 Desenvolvimento do trabalho

Este trabalho apresenta inicialmente a definição do problema de estimação de sistemas não-lineares abordando de maneira geral as principais técnicas utilizadas atualmente, que formam o estado da arte. Abordamos de maneira mais detalhada a utilização do *kernel* em estimação não-linear. Para o desenvolvimento deste trabalho vamos considerar um modelo de ordem finita. Na Seção 2.4 apresentamos as justificativas para essa abordagem (modelo de ordem finita). Na Subseção 2.4.1 descrevemos as principais técnicas utilizadas para otimizar a estimação não-linear quando é utilizado o modelo de ordem finita. Finalizamos o Capítulo 2 com exemplos dos principais *kernels* utilizados em processamento de sinais e a proposta de tese.

No Capítulo 3 apresentamos a análise do comportamento estocástico do algoritmo adaptativo *Kernel Least-Mean-Square* (KLMS). Apresentamos a solução de Wiener na Seção 3.2. Fazemos a análise do comportamento transitório do algoritmo KLMS, período de adaptação dos coeficientes, na Seção 3.3. Algumas hipóteses simplificadoras foram utilizadas para que fosse possível modelar matematicamente o comportamento estocástico.

Com o objetivo de obter um conjunto de equações que estabeleçam os limites de estabilidade do algoritmo, utilizamos a técnica de projeção lexicográfica. Algumas desigualdades obtidas a partir dos momentos de quarta ordem permitiram simplificar e estudar as condições de estabilidade do KLMS deduzidas na Seção 3.4 e posteriormente na Seção 4.4. Na Seção 3.4.2, finalizamos o conjunto de equações que representam o comportamento

estocástico para qualquer *kernel*, definindo as expressões para regime permanente do erro quadrático médio e o erro quadrático médio em excesso.

Escolhemos duas funções *kernel* para avaliar o comportamento estocástico e obter os resultados necessários para estabelecer uma comparação e optar pelo melhor *kernel*[2]. Optamos pelo emprego *kernels* gaussiano e polinomial devido ao seu frequente uso em processamento de sinais e por possuírem características diferentes. Dentre esses dois, o *kernel* gaussiano tem capacidade de aproximação universal, isto é, a combinação linear de funções *kernel* aplicadas em determinados pontos pode aproximar qualquer função contínua (PÉREZ-CRUZ; BOUSQUET, 2004). Já a capacidade de aproximação de um *kernel* polinomial de ordem β está limitada a aproximação de uma função polinomial (ou que possa ser razoávelmente aproximada por um polinômio) de grau menor ou igual a β (LIU; PRINCIPE; HAYKIN, 2010). No Capítulo 4 particularizamos a análise estatística para os casos desses dois *kernels*, gaussiano e polinomial, a fim de apresentar os momentos necessários para análise do comportamento estocástico do KLMS gaussiano e polinomial.

Usando as equações deduzidas nos Capítulos 3 e 4 foi possível estabelecer um conjunto de diretrizes de projeto de um filtro adaptativo KLMS usando os *kernels* gaussiano e polinomial. Avaliamos o KLMS (gaussiano e polinomial) para alguns parâmetros e verificamos a qualidade de estimação e o tempo necessário para convergência em cada um dos casos apresentados. O modelo de predição mostrou excelente concordância com as simulações de Monte Carlo, tanto no período transitório quanto em regime permanente.

Finalizamos o trabalho com as conclusões, sugestões de pesquisas futuras e as considerações finais obtidas a partir da análise do comportamento estocástico do algoritmo KLMS (gaussiano e polinomial).

[2]Entenda por melhor *kernel* aquele que aplicado ao algoritmo LMS produz uma estimação com uma qualidade suficiente com a menor quantidade de iterações.

1.3 Trabalhos publicados

A seguir apresentamos as primeiras publicações geradas a partir do estudo realizado.

1. **Artigos publicados em congressos científicos internacionais**

   - PARREIRA, Wemerson Delcio; BERMUDEZ, José Carlos Moreira; RICHARD, Cédric e TOURNERET, Jean-Ives. *Stochastic behavior analysis of the Gaussian Kernel Least Mean Square algorithm*. In Proc. IEEE ICASSP'11. Prague, Czech Republic, 2011. 4116–4119.

   - PARREIRA, Wemerson Delcio; BERMUDEZ, José Carlos Moreira; RICHARD, Cédric e TOURNERET, Jean-Ives. *Steady–state behavior and design of the Gaussian KLMS algorithm*. In 19th European Signal Processing Conference (EUSIPCO – 2011), Barcelone, Espagne. EURASIP, 2011. 121–125.

2. **Artigo publicado em periódico internacional**

   - PARREIRA, Wemerson Delcio; BERMUDEZ, José Carlos Moreira; RICHARD, Cédric e TOURNERET, Jean-Ives. *Stochastic Behavior Analysis of the Gaussian Kernel Least-Mean-Square algorithm*. IEEE TRANSACTIONS ON SIGNAL PROCESSING. VOL 60, NO. 5. MAY 2012. 2208–2222.

2 PROBLEMA DE ESTIMAÇÃO DE SISTEMAS NÃO-LINEARES

2.1 Introdução

Muitas aplicações práticas (ex. bioengenharia, comunicação, equalização de canais entre outros) requer um processamento não-linear do sinal. Considere um sistema não-linear caracterizado pela função não-linear $\varphi(.)$ e representado na Figura 2.1 em que $\boldsymbol{u}(n) \in \mathbb{R}^q$ é a entrada deste sistema no instante n e $\varphi(\boldsymbol{u}(n)) \in \mathbb{R}$ é a saída deste sistema. Comumente a saída $\varphi(\boldsymbol{u}(n))$ é corrompida por um ruído $z(n)$. Com o objetivo de estimar $\varphi(\boldsymbol{u}(n))$ a partir da saída ruidosa,

$$d(n) = \varphi(\boldsymbol{u}(n)) + z(n), \tag{2.1}$$

muito autores recorrem a técnicas de processamento não-linear.

Os sistemas não-lineares podem ser caracterizados por representações que vão desde as estatísticas de ordem superior aos métodos de expansão em séries (SCHETZEN, 1980). As redes neurais artificiais (HAYKIN, 1994, e suas referências) e a filtragem polinomial (MATHEWS; SICURANZA, 2000, e suas referências) baseada nas séries de Volterra foram durante décadas opções bastante atrativas para resolução do problema de estimação não-linear. Nas próximas seções apresentamos uma breve caracterização dessas duas técnicas.

Nas últimas décadas, a estimação de sistemas não-lineares a partir dos métodos baseados no espaço Hilbertiano de *kernels* reprodutivos – RKHS ganharam popularidade junto à comunidade científica (KIMELDORF; WAHBA, 1971; DUTTWEILER; KAILATH, 1973). Recentemente, a filtragem adaptativa baseada em *kernel* tem sido reconhecida com uma solução atraente para esse problema. A utilização do RKHS permite o uso de estruturas lineares para resolver problemas de estimação não-linear (LIU; PRINCIPE; HAYKIN, 2010). Na Seção 2.4 descrevemos como é feita a

estimação de $d(n)$ no RKHS. Essa técnica é o objeto de estudo desta tese.

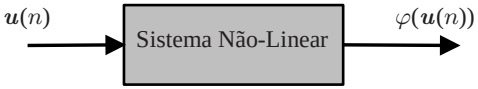


Figura 2.1: Sistema não-linear.

2.2 Redes Neurais

As redes neurais artificiais (RNAs) são amplamente usadas para modelar sistemas não-lineares usando interconexões de estruturas não-lineares mais simples denominadas neurônios artificiais. Visto que os neurônios artificiais são tipicamente sistemas com múltiplas entradas e única saída, a estimativa do sinal desejado $d(n)$ usando apenas uma função de ativação é dada pela relação:

$$\hat{d}(n) = f\left(\sum_{i=1}^{N} \alpha_i u_i(n) - \theta\right) \tag{2.2}$$

em que α_i é um peso associado à i-ésima entrada $u_i(n)$ do dispositivo e θ é um termo constante que controla o ponto de operação da não-linearidade f. Muitos tipos diferentes de funções de ativação podem ser empregados e algumas das funções mais comuns são (HAYKIN, 1994):

1. Função de Heaviside:

$$f(x) = \begin{cases} 1, & \text{se } x \geq 0 \\ 0, & \text{caso contrário} \end{cases} \tag{2.3}$$

2. Função Linear Por Partes:

$$f(x) = \begin{cases} 1, & \text{se } x \geq 1/2 \\ x, & \text{se } -1/2 < x < 1/2 \\ 0, & \text{se } x \leq -1/2 \end{cases} \tag{2.4}$$

3. Função Sigmoidal: é definida como uma função estritamente crescente que exibe um fator de alisamento e propriedades assintóticas. Um exemplo de função sigmoidal é a função logística

$$f(x) = \frac{1}{1 + \exp(-ax)} \tag{2.5}$$

em que a é o parâmetro de inclinação da função sigmoidal. Um outro exemplo de função Sigmóide é a função Sinal:

$$f(x) = \begin{cases} 1, & \text{se } x > 0 \\ 0, & \text{se } x = 0 \\ -1, & \text{se } x < 0 \end{cases} \tag{2.6}$$

Dentre esses três exemplos de funções, a função Sigmoide é a função de ativação mais comumente utilizada na construção de redes neurais artificiais.

Pode ser demonstrado que interconexões apropriadas dessas estruturas não-lineares simples (neurônios artificiais) são capazes de produzir aproximações eficientes de uma grande classe de sistemas não-lineares (MATHEWS; SICURANZA, 2000). Em aplicações práticas, os pesos α_i usados nas redes neurais são selecionados a partir do treinamento usando dados representativos do problema. O treinamento é realizado usando um algoritmo de adaptação, como por exemplo o algoritmo *backpropagation*. Maiores detalhes a respeito desse algoritmo podem ser obtidos dentre outras referências em (HAYKIN, 1994).

De um modo geral, o projeto de uma rede neural é uma tentativa de se imitar o funcionamento do sistema nervoso. O sistema nervoso é formado por interconexões massivamente paralelas de um número muito grande de neurônios. Essa rede de interligações, que formam o sistema nervoso, é capaz de realizar o reconhecimento e tomar decisões em tarefas que não podem ser acompanhadas por até mesmo os supercomputadores mais poderosos existentes no momento.

A característica mais importante das RNAs é a sua capacidade de modelar os sistemas não-lineares genéricos. Sua potencialidade para o tratamento de sistemas complexos, o processamento paralelo, a aprendizagem, a adaptabilidade e a generalização são grandes motivadoras do emprego das RNAs no processamento não-linear de sinais. No entanto, uma desvantagem para aplicações que requerem treinamento em tempo real é que as RNAs levam a problemas de otimização não-convexos, com existência de mínimos locais. Outra desvantagem das redes neurais é que a convergência global de algoritmos de treinamento, como o *backpropagation*, não é garantida (MATHEWS; SICURANZA, 2000). Dependendo da arquitetura usada, as RNAs podem exigir um número demasiadamente grande de neurônios artificiais para realizar a modelagem adequada. As desvantagens já citadas, a dificuldade de estabelecer metodologias sistemáticas de projeto, uma vez que são difíceis de analisar, e a tendência ao alto custo computacional tornam as redes neurais uma opção pouco atrativa para tratamento de problemas em tempo real.

2.3 Filtros Polinomiais

Uma abordagem construtiva e versátil para o problema de estimação não-linear é utilizar a estrutura do filtro baseada nas Séries de Volterra. Faremos nessa subseção apenas uma breve abordagem desta técnica, maiores detalhes podem ser encontrados em (MATHEWS; SICURANZA, 2000, e suas

referências).

Os filtros polinomiais de tempo discreto causais estimam a saída desejada, $d(n)$, usando

$$\hat{d}(n) = \sum_{i=0}^{P} f_i(u(n), u(n-1), \ldots, u(n-N), y(n-1), \ldots, y(n-M)), \quad (2.7)$$

em que a função $f_i(.)$ é um polinômio de ordem i com variáveis $u(n), u(n-1), \ldots, u(n-N), y(n-1), \ldots$ e $y(n-M)$. Essa definição contém os filtros lineares já que Equação (2.7) é linear se $f_i(.) = 0$ para todo $i \neq 1$. Se (2.7) é estável no sentido BIBO (entrada limitada implica em saída limitada), ela admite uma expansão em Séries de Volterra convergente na forma

$$\begin{aligned}\hat{d}(n) =& h_0 + \sum_{m_1=0}^{N_1-1} h_1(m_1)u(n-m_1) \\ &+ \sum_{m_1=0}^{N_2-1}\sum_{m_2=0}^{N_2-1} h_2(m_1,m_2)u(n-m_1)u(n-m_2) + \ldots + \sum_{m_1=0}^{N_r-1}\sum_{m_2=0}^{N_r-1} \\ &\ldots \sum_{m_r=0}^{N_r-1} h_r(m_1,m_2,\ldots,m_r)u(n-m_1)u(n-m_2)\ldots u(n-m_r) \\ &+ \ldots,\end{aligned} \quad (2.8)$$

em que $h_r(m_1, m_2, \ldots, m_r)$ representa o polinômio de Volterra da r-ésima ordem do sistema não-linear. Esses filtros são amplamente aplicados, por exemplo, em identificação de sistemas não-lineares. A seguir listaremos as vantagens e desvantagens do uso dos filtros polinomiais em aplicações de filtragem adaptativa não-linear:

- Vantagens (MATHEWS; SICURANZA, 2000):

    1. Muitos dos resultados obtidos a partir da análise e do projeto de filtros lineares podem ser usados em filtros polinomiais, pelo fato

dos filtros polinomiais se tratarem de uma extensão dos filtros lineares.

2. É possível mostrar que, usando um número finito de coeficientes, os sistemas polinomiais são capazes de representar uma classe relativamente grande de sistemas não-lineares.
3. Levam a problemas de otimização convexos, geralmente com alguma redução da capacidade de generalização.

- Desvantagens (MATHEWS; SICURANZA, 2000):

1. Requerem um grande número de coeficientes para modelar adequadamente os sistemas não-lineares provenientes de aplicações práticas.
2. Difíceis de serem analisados quando a ordem é superior a dois.
3. As propriedades de estabilidade dos sistemas polinomiais recursivos não são completamente compreendidas neste momento. Para muitos dos modelos, só podemos desenvolver condições suficientes para a estabilidade que tendem a ser muito restritivas.
4. Tendem a ter elevada complexidade computacional, sendo pouco atraentes para tratamento de problemas em tempo real.

2.4 Estimação não-linear usando kernel

Os *kernels* são funções não-lineares representadas por $\kappa(.,.)$, que oferecem estruturas gerais para representação de dados. No Apêndice A apresentamos um exemplo do uso do *kernel* para representar dados por medida de similaridade. Os *kernels* possuem determinadas propriedades matemáticas que são úteis para aplicação em processamento de sinais. No Apêndice B apresentamos a definição de *kernel* e descrevemos algumas dessas propriedades. A Figura 2.2 representa um diagrama de blocos do problema de

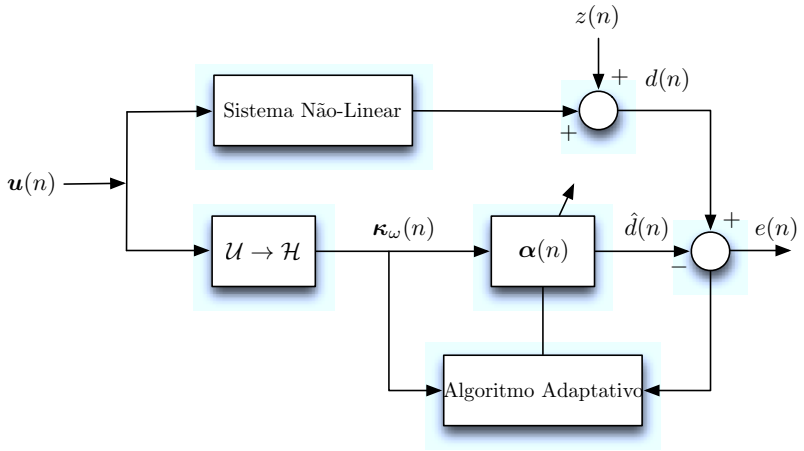


Figura 2.2: Estimação de sistemas baseado no algoritmos adaptativo KLMS.

estimação de sistemas não-lineares usando algoritmos adaptativos "kernelizados". Aqui, $\mathcal{U}$ representa um subespaço compacto de $\mathbb{R}^q$, $\kappa : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ é um *kernel* reprodutivo (ver Definição B.4), $(\mathcal{H}, \langle \cdot, \cdot \rangle_{\mathcal{H}})$ é o produto interno induzido no RKHS e $z(n)$ é um ruído aditivo descorrelacionado de qualquer outro sinal.

O problema de estimação do sinal desejado $d(n)$ pode ser reduzido ao problema de minimização quadrática, em que devemos encontrar uma função $\psi \in \mathcal{H}$ tal que

$$\psi = \text{argmin}_{\psi \in \mathcal{H}} \sum_{n=1}^{N} (d(n) - \psi(\boldsymbol{u}(n)))^2. \tag{2.9}$$

A solução deste problema é obtida a partir do Teorema da Representação (KIMELDORF; WAHBA, 1971) que estabelece: dados N vetores de entrada $\boldsymbol{u}(n)$ e a saída desejada do sistema $d(n)$, $n = 1, \ldots, N$, a função $\psi(\cdot) \in \mathcal{H}$ que

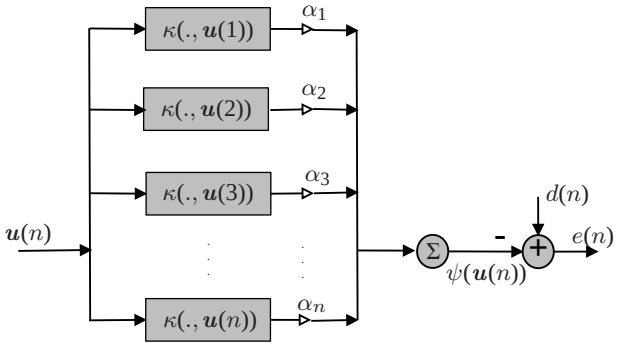


Figura 2.3: Filtro linear com vetores das características no espaço das características.

minimiza o erro de estimação quadrático dada por:

$$\sum_{n=1}^{N} (d(n) - \psi(\boldsymbol{u}(n)))^2,$$

pode ser escrita como uma expansão em função do *kernel*,

$$\psi(\boldsymbol{u}(n)) = \sum_{k=1}^{n} \alpha_k \, \kappa(\boldsymbol{u}(n), \boldsymbol{u}(k)).$$

Essa expansão é devida à propriedade reprodutiva dos *kernels* (veja em detalhes esta propriedade no Apêndice B, Equação (B.19)). Então, dadas $n = N$ observações, a expansão em função dos *kernels* reduz o problema de estimação à determinação de $\alpha = [\alpha_1, \ldots, \alpha_N]^\top$ o qual minimiza $\| \boldsymbol{d} - \boldsymbol{K}\alpha \|^2$, em que $\boldsymbol{K}$ é a matriz *Gram* com (n, ℓ)-ésima componente dada por $\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\ell))$, e $\boldsymbol{d} = [d(1), \ldots, d(N)]^\top$. Desta maneira, a ordem do filtro é igual a N, como é exibido na Figura 2.3. Neste contexto a utilização dos algoritmos adaptativos "kernelizados" seria inviável, em tempo real (N muito grande).

Assim, uma alternativa é utilizar um modelo de ordem finita (usando um número finito de observações), obtido a partir de um **dicionário de**

funções kernel. O dicionário de funções *kernel* é definido como:

Definição 2.1 (Dicionário) *: Seja $\kappa : \mathbb{R}^q \times \mathbb{R}^q \to \mathbb{R}$ uma função kernel, $M \in \mathbb{N}^*$ e $\boldsymbol{u}(n) \in \mathbb{R}^q$ as observações feitas. Definimos dicionário um conjunto $\mathcal{D}$ de funções kernel $\kappa(.,\boldsymbol{u}(\omega_j))$ dado por:*

$$\mathcal{D} := \{\kappa(.,\boldsymbol{u}(\omega_1)),\ \kappa(.,\boldsymbol{u}(\omega_2)),\ \kappa(.,\boldsymbol{u}(\omega_3)),\ \ldots,\kappa(.,\boldsymbol{u}(\omega_M))\}.$$

Na Definição 2.1, o parâmetro $M \leq N$ representa a ordem (cardinalidade de $\mathcal{D}$, denotado por $\#\mathcal{D}$) e ω_j pertence a um conjunto de índices $\mathcal{I} \subset \{1,2,\ldots,N\}$, usado para renomear as observações passadas de $\boldsymbol{u}(n)$, escolhidas para compor o dicionário.

Considerando um dicionário $\mathcal{D}$, $\alpha_j \in \mathbb{R}$ e $M \in \mathbb{N}^*$, o modelo de ordem finita para estimação de $d(.)$ será dado por:

$$\psi(\boldsymbol{u}(n)) = \sum_{j=1}^{M} \alpha_i \kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_j)). \tag{2.10}$$

Usando um modelo de ordem finita, como o representado pela Figura 2.4, a estimação de uma função não-linear no instante $n > M$ é calculada usando uma medida de similaridade das M observações passadas $\boldsymbol{u}(\omega_j)$ (aquelas selecionadas para compor o dicionário) com $\boldsymbol{u}(n)$. Desta maneira, com um modelo de ordem finita, $\psi(\boldsymbol{u}(n))$ representará uma estimativa do sinal desejado $d(n)$, ou seja, $\hat{d}(n)$.

Utilizando este tipo de modelagem, temos a vantagem de poder estudar o comportamento estocástico do algoritmo adaptativo "kernelizado". Porém, o uso de um modelo de ordem finita gera um erro na estimação. Alguns autores têm utilizado técnicas para aumentar a representatividade na ordem M do dicionário. Essas técnicas são baseadas na redução da redundância destes elementos. Uma revisão a respeito das técnicas existentes para selecionar as funções *kernel* de (2.10) foi apresentada em (LIU; PRINCIPE; HAYKIN, 2010). Estas técnicas são comumente denominadas **Técnicas de**

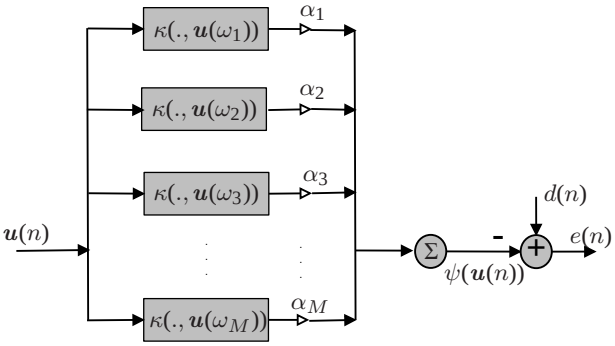


Figura 2.4: Filtro linear de ordem finita.

Esparsificação de Dicionários. Na próxima subseção, apresentaremos uma visão geral das técnicas comumente empregadas.

2.4.1 Esparsificação de dicionários

Um tipo básico de esparsificação consiste em fazer uma avaliação aproximada da dependência linear, ou seja, avaliar se uma função *kernel* "candidata" ao dicionário pode ser representada por uma combinação linear dos elementos que já pertencem ao conjunto $\mathcal{D}$. Ou ainda, encontrar a melhor representação tal que $\mathcal{D}$ tenha a menor quantidade de elementos. A seguir, descreveremos algumas dessas técnicas.

Alguns métodos de esparsificação trabalham com o objetivo de encontrar a melhor combinação linear de M funções de um dicionário redundante. Os algoritmos *Greedy* buscam uma solução localmente ótima para o problema de aproximação esparsa como uma possibilidade de encontrar a solução globalmente ótima. Tropp propõe o uso dos algoritmos *Greedy* para resolver o problema de aproximação esparsa e descreve alguns tipos desses algoritmos (TROPP, 2004):

- *Matching Pursuit* (MP): Introduzido por Mallat e Zhang em 1993, este

algoritmo decompõe um sinal em termos de uma combinação linear de funções que compõem um dicionário (MALLAT; ZHANG, 1993). Se estas funções não formam uma base ortonormal, então este algoritmo seleciona a melhor combinação para estimar o sinal em questão, ou seja, este algoritmo escolhe a cada iteração uma forma de onda que melhor se adapte à aproximação do sinal (sinal estimado) usando as funções do dicionário.

- *Orthogonal Matching Pursuit* (OMP): Este algoritmo proposto por Davis e outros usa mínimos quadrados com o objetivo de resolver o problema de minimização em cada iteração do algoritmo MP para obter melhor aproximação dos elementos do dicionário escolhido (DAVIS; MALLAT; ZHANG, 1994). Essa modificação gera significativa melhora no algoritmo.

- *Weak Greedy Algorithms* (WGA): Possui uma estrutura que é uma variação do OMP. Por isso é também denominado *Weak* OMP (WOMP) (TROPP, 2004). Esse algoritmo em vez de selecionar o elemento ótimo em cada etapa, seleciona um elemento "quase" ótimo, ou seja, próximo ao ótimo.

- *Basis Pursuit* (BP): A ideia fundamental deste método é usar o fato de que o número de termos usados na representação de um sinal (ou função) pode ser aproximado pela soma absoluta dos coeficientes. Esta soma é uma função convexa que pode ser minimizada (TROPP, 2004). Este tipo de aproximação é considerado mais sofisticado que a simples troca do problema de aproximação esparsa por um problema de programação linear.

A maior vantagem do OMP com relação ao BP está na simplicidade da implementação. Porém, alguns pesquisadores sugerem que o desempenho do

algoritmo BP é melhor que do algoritmo OMP (CHEN; DONOHO; SAUNDERS, 1999; TROPP, 2004). Maiores detalhes desses métodos podem ser vistos também em Mallat (MALLAT, 2008).

A proposta para redução da redundância do dicionário, envolvendo funções *kernel*, de Schölkopf e outros (SCHÖLKOPF; SMOLA; MÜLLER, 1998), parte das observações do espaço nulo da matriz *Gram*, $\boldsymbol{K}_{ij} = \langle \boldsymbol{u}(\omega_i), \boldsymbol{u}(\omega_j) \rangle$, com $i, j = 1,\ 2,\ ...,\ L$. Considere a existência de um autovetor $\theta \neq \boldsymbol{0}$ associado a um auto-valor nulo de $\boldsymbol{K}$, representado por:

$$\sum_{i=1}^{L} \kappa(\boldsymbol{u}(\omega_j), \boldsymbol{u}(\omega_i))\theta_i = 0 \tag{2.11}$$

Então, se a igualdade (2.11) for verdadeira para todo $j = 1,\ 2,\ ...,\ L$, é possível escrever algum $\kappa(., \boldsymbol{u}(\omega_j))$ como combinação linear de outros. Portanto podemos excluir o termo $\kappa(., \boldsymbol{u}(\omega_j))$ do dicionário. Caso não seja possível encontrar um autovalor nulo, Schölkopf oferece a alternativa de resolver um problema de minimização que envolve os autovalores da matriz *Gram* ($\boldsymbol{K}$).

Um outro exemplo é o critério de dependência linear aproximada (ALD) (ENGEL; MANNOR; MEIR, 2004), que consiste em incluir uma função kernel $\kappa(\cdot, \boldsymbol{u}(\ell))$ no dicionário se ela satisfaz

$$\min_{\gamma} \| \kappa(\cdot, \boldsymbol{u}(\ell)) - \sum_{j} \gamma_j\, \kappa(\cdot, \boldsymbol{u}(\omega_j)) \|^2_{\mathcal{H}} > \epsilon_0 \tag{2.12}$$

em que ϵ_0 é um parâmetro que determina o nível de esparsidade do modelo.

Para controlar a ordem do modelo com redução da complexidade computacional pode ser usada a regra de esparsificação baseada no critério de coerência (HONEINE; RICHARD; BERMUDEZ, 2007; RICHARD; BERMUDEZ; HONEINE, 2009). De acordo com esse critério, o *kernel* $\kappa(\cdot, \boldsymbol{u}(\ell))$ é inserido no dicionário se

$$\max_{j} |\kappa(\boldsymbol{u}(\ell), \boldsymbol{u}(\omega_j))| \leq \varepsilon_0 \tag{2.13}$$

com ε_0 sendo um fator que determina a coerência do dicionário. É possível mostrar que o dicionário criado a partir da Regra (2.13) é finito mesmo com n grande (RICHARD; BERMUDEZ; HONEINE, 2009).

A complexidade computacional do teste (2.13) é em geral bastante reduzida em relação à dos demais métodos mencionados anteriormente. Usando um processador digital de sinais (DSP) convencional de 32 *bits*, como por exemplo um *Blackfin ADSP 537* operando com precisão de 16 *bits*, o número de ciclos do processador necessários para executar (2.13) é de cerca de

$$\mathcal{C} = \frac{6M+1}{2} \tag{2.14}$$

em que M é a cardinalidade do dicionário $\mathcal{D}$. Note que os valores dos *kernels* $\kappa(\boldsymbol{u}(\ell), \boldsymbol{u}(\omega_j)$ que são necessários para o teste (2.13) já foram calculados na iteração anterior. Assim, seu cálculo não impacta a complexidade computacional do teste, que é determinada basicamente pelos cálculos das magnitudes e pelas comparações.

2.4.2 Algoritmos adaptativos "kernelizados"

Algoritmos adaptativos "kernelizados" baseados no gradiente estocástico com entrada não-linear têm sido muito utilizados. O emprego dessas estruturas está relacionado ao fato de podermos escrever uma função não-linear $\psi(\boldsymbol{u}(.))$ em termos de um produto interno, utilizando o conhecido truque do *kernel*, que é descrito no Apêndice B.2.1.

A teoria descrita na subseção anterior nos permite inferir que, com uma regra de esparsificação adequada, os problemas não-lineares de filtragem adaptativa podem ser formulados como um problema linear de ordem finita em que o sinal de entrada em $\mathcal{U}$ foi previamente mapeado de forma não-linear por um *kernel* em um espaço de Hilbert $\mathcal{H}$. Algoritmos desenvolvidos usando estas ideias incluem o algoritmo *Kernel Least-Mean-Square* (KLMS) (LIU; POKHAREL; PRINCIPE, 2008; BOUBOULIS; THEODORIDIS, 2011), o

Kernel Recursive-Least-Mean-Square (KRLS) (ENGEL; MANNOR; MEIR, 2004), o KLMS Normalizado (KNLMS) e o algoritmo de Projeções Afins baseado em kernel (KAPA) (HONEINE; RICHARD; BERMUDEZ, 2007; SLAVAKIS; THEODORIDIS, 2008; RICHARD; BERMUDEZ; HONEINE, 2009). Liu *et al.* faz uma compilação desses algoritmos "kernelizados" em (LIU; PRINCIPE; HAYKIN, 2010).

Além da escolha dos usuais parâmetros do filtro adaptativo linear, projetar filtros adaptativos "kernelizados" requer a escolha do *kernel* e de seus parâmetros. A escolha desses parâmetros para alcançar uma eficiência desejada ainda era uma questão em aberto, e exige uma extensa análise do comportamento estocástico algoritmo. Nosso trabalho traz uma nova contribuição para a discussão sobre a filtragem adaptativa baseada em *kernels*, fornecendo uma primeira análise do comportamento estocástico do algoritmo KLMS, bem como de suas propriedades de convergência.

No próximo capítulo apresentamos a derivação das expressões recursivas para o comportamento médio do vetor de pesos e para o erro quadrático médio (MSE) para entradas gaussianas. Definimos os modelos analíticos para o comportamento transitório dos momentos de primeira e segunda ordem dos pesos adaptativos.

2.4.3 Exemplos de kernels

Como descrito anteriormente, os algoritmos adaptativos são responsáveis pelo ajuste do vetor de pesos nos coeficientes para uma estimativa de $d(n)$ a partir de funções *kernel* (Teorema da Representação). No próximo capítulo veremos com mais detalhes como essas funções *kernel* podem alterar o comportamento do algoritmo em termos da qualidade da estimativa e da velocidade de convergência, uma vez que o sinal de entrada é alterado pela transformação não-linear imposta pelo *kernel*. A seguir, apresentaremos alguns exemplos de funções *kernel* que são as estruturas responsáveis pela

estimação da parte não-linear do sinal desejado $d(n)$.

Exemplo 2.1 (Kernel gaussiano) *: também denominado função kernel de base radial gaussiana, (SCHÖLKOPF; SMOLA, 2002) é dado por:*

$$\kappa(\boldsymbol{u}, \boldsymbol{u}') = \exp\left(\frac{-||\boldsymbol{u} - \boldsymbol{u}'||^2}{2\xi^2}\right) \tag{2.15}$$

em que $\boldsymbol{u}, \boldsymbol{u}' \in \mathbb{R}^q$ *e* $\xi > 0$.

Exemplo 2.2 (Kernel laplaciano) *: também conhecido como função kernel de base radial laplaciana, (SCHÖLKOPF; SMOLA, 2002) é dado por:*

$$\kappa(\boldsymbol{u}, \boldsymbol{u}') = \exp\left(\frac{-||\boldsymbol{u} - \boldsymbol{u}'||}{\xi}\right) \tag{2.16}$$

em que $\boldsymbol{u}, \boldsymbol{u}' \in \mathbb{R}^q$ *e* $\xi > 0$.

Exemplo 2.3 (Kernel polinomial) *: o denominado kernel polinomial de* β*-ésimo grau é dado por: (SCHÖLKOPF; SMOLA, 2002)*

$$\kappa(\boldsymbol{u}, \boldsymbol{u}') = (a + \boldsymbol{u}^\top \boldsymbol{u}')^\beta \tag{2.17}$$

em que $\boldsymbol{u}, \boldsymbol{u}' \in \mathbb{R}^q$, $\beta \in \mathbb{N}^*$ *e* $a \geq 0$

Note nos exemplos que cada um dos *kernels* possui parâmetros ajustáveis e estes parâmetros afetam a qualidade da estimação e a velocidade de convergência dos algoritmos adaptativos. Nos próximos capítulos, apresentamos, de maneira mais clara, essa relação.

2.5 Conclusão e proposta de tese

Neste capítulo descrevemos o problema de estimação de sistemas não-lineares. Abordamos brevemente as técnicas de filtragem polinomial (filtros por série de Volterra) e as redes neurais, que são as comumente utilizadas pela comunidade científica. Listamos algumas das características em que a

utilização dessas técnicas pode ser vantajosa ou desvantajosa. De maneira mais detalhada, descrevemos o uso do *kernel* em estimação não-linear.

A partir das propriedades dos *kernels*, dos *kernels* nos RKHS e do "truque" do *kernel* é possível estimar um sinal não-linear usando uma combinação linear de funções *kernel* em $\mathcal{H}$. Notamos que o estudo do comportamento estocástico dos algoritmos adaptativos que proverá o ajuste dos coeficientes (pesos), bem como a aplicação desses métodos de estimação em tempo real, necessita de uma aproximação de ordem finita. Para isto, é construído o dicionário de funções *kernel*. O dicionário pode ser otimizado (tornado menos redundante) utilizando técnicas de esparsificação, as quais já foram apresentadas com bons resultados à comunidade científica.

Usando um algoritmo adaptativo linear podemos prover a adaptação dos coeficientes deste filtro. A entrada $\boldsymbol{u}(n)$ "kernelizada" pode ser obtida a partir dos *kernels* apresentados nos Exemplos 2.1, 2.2 e 2.3, ou ainda outros presentes em (SCHÖLKOPF; SMOLA, 2002). Até hoje não existia um critério de escolha do *kernel* ou do parâmetro do *kernel* de forma a otimizar o desempenho do algoritmo adaptativo. Isto não podia ser feito porque não havia um modelo de comportamento estocástico do algoritmo em função desses parâmetros.

Sendo assim, a nossa proposta é encontrar um conjunto de equações que modelem o comportamento do KLMS. Para isso devemos estabelecer um conjunto de equações que comporão um projeto para um filtro adaptativo não-linear usando o KLMS. Essas equações devem relacionar as estatísticas do sinal de entrada, e os parâmetros ajustáveis do *kernel*. Escolhemos o *kernel* gaussiano (Ex. 2.1) e o *kernel* polinomial (Ex. 2.3) para estudo de suas propriedades estatísticas e para estabelecer as condições de "melhor" convergência em regime permanente e de tempo de ajuste dos coeficientes (regime transitório).

3 MODELAGEM ESTOCÁSTICA DO ALGORITMO KLMS

3.1 Introdução

Desenvolvido por Widrow 1960, o algoritmo *Least Mean-Squares* (LMS) é muito utilizado para a minimização do erro quadrático médio (MSE) de estimação. Este algoritmo implementa uma versão estocástica do algoritmo *steepest descent*, em que substitui o gradiente exato da função custo J_α pela estimativa instantânea dada por:

$$\nabla \hat{J}_\alpha(n) = -e(n)\boldsymbol{u}(n),$$

em que $\boldsymbol{u}(n)$ é o sinal de entrada e $e(n)$ é o erro de estimação.

A equação de atualização dos coeficientes $\alpha(n)$, para um dado passo η, do LMS é dada por:

$$\alpha(n+1) = \alpha(n) + \eta e(n)\boldsymbol{u}(n). \tag{3.1}$$

O algoritmo KLMS, ou LMS "Kernelizado", consiste em avaliar (3.1) no espaço das características. Para isto assumimos que um dado $\kappa(.,.) \in \mathcal{H}$ mapeia o sinal de entrada $\boldsymbol{u}(n)$ em um espaço de entrada $\kappa(\boldsymbol{u}(.),.)$, que é o espaço das características do *kernel*. Esta transformação do espaço de entrada no espaço das características é mais comumente utilizada para sinais não-lineares. Desde que o espaço das características seja linear, $\kappa(.,\boldsymbol{u}(n)) = [\kappa(\boldsymbol{u}(1),\boldsymbol{u}(n)),\kappa(\boldsymbol{u}(2),\boldsymbol{u}(n)),...,\kappa(\boldsymbol{u}(n),\boldsymbol{u}(n))]^\top$ pode ser considerado um vetor coluna do Espaço de Hilbert (POKHAREL; LIU; PRINCIPE, 2007).

Apresentaremos nesta seção uma análise do comportamento estocástico do KLMS modificado para uma ordem finita (2.10), já que a utilização prática do algoritmo KLMS no espaço das características, que teria o comportamento regular do LMS no RKHS de (POKHAREL; LIU; PRINCIPE, 2007) e (LIU; PRINCIPE; HAYKIN, 2010), é inviável. Vamos assumir M, o

comprimento do dicionário, o qual define a ordem do filtro adaptativo, como conhecido, fixo e finito em nossa análise. Mais tarde faremos considerações sobre como determinar essa ordem em uma situação de projeto.

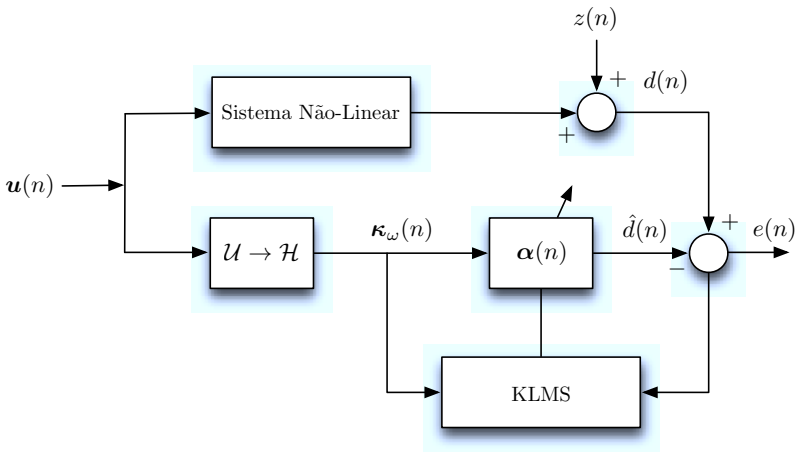


Figura 3.1: Estimação de sistemas baseado em algoritmos adaptativos "kernelizados".

3.2 Solução de Wiener

Considere o problema de estimação não-linear de sistemas representado pela Figura 3.1, e o modelo de ordem finita (2.10) baseado em um *kernel*, $\kappa(.,.)$.

Assumimos que o evento é estacionário, ou seja, $\varphi(\boldsymbol{u}(n))$ é estacionário para $\boldsymbol{u}(n)$ estacionário. Alguns modelos práticos, tais como, sistemas sem memória, modelos de Wiener e Hammerstein satisfazem esta condição. Consideramos também que a entrada do sistema $\boldsymbol{u}(n)$ é um vetor $(q \times 1)$ de variáveis aleatórias gaussianas, de média nula, independente e identicamente distribuído (i.i.d.), isto é, $E\{\boldsymbol{u}(n-i)\,\boldsymbol{u}^{\top}(n-j)\} = \mathbf{0}$ para $i \neq j$. As

componentes do vetor $\boldsymbol{u}(n)$ podem, de alguma maneira, ser correlacionadas. Portanto, a matriz de autocorrelação de $\boldsymbol{u}(n)$, $\boldsymbol{R}_{\boldsymbol{uu}} = E\{\boldsymbol{u}(n)\,\boldsymbol{u}^{\top}(n)\}$, pode não ser uma matriz diagonal.

Para um dicionário de comprimento M considere $\kappa_{\omega}(n)$ o vetor de *kernels* no instante $n > M$[1], representado por

$$\kappa_{\omega}(n) = [\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_1)), \ldots, \kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_M))]^{\top} \tag{3.2}$$

em que $\kappa(\boldsymbol{u}(.,\omega_i))$ é a i-ésima componente do dicionário, com $\boldsymbol{u}(\omega_i) \neq \boldsymbol{u}(n)$ para $i = 1, \ldots, M$. Aqui consideramos que os vetores $\boldsymbol{u}(\omega_i)$, $i = 1, \ldots, M$ podem mudar após um procedimento de atualização. A única limitação imposta na seguinte análise é que $\boldsymbol{u}(\omega_i) \neq \boldsymbol{u}(\omega_j)$ para $i \neq j$ em cada instante de tempo, de modo que os vetores que são argumentos de entradas diferentes de $\kappa_{\omega}(n)$ são estatisticamente independentes. Para tornar a notação mais simples, contudo, não explicitamos a dependência de ω_i em n e representamos $\boldsymbol{u}(\omega_i(n))$ como $\boldsymbol{u}(\omega_i)$ para todo i.

A partir do modelo de ordem finita (2.10), a estimação da saída do sistema representado na Figura 3.1 é

$$\hat{d}(n) = \alpha^{\top}(n)\,\kappa_{\omega}(n) \tag{3.3}$$

com $\alpha(n)$= $[\alpha_1(n), \ldots, \alpha_M(n)]^{\top}$.

Neste caso, o erro de estimação é definido como:

$$e(n) = d(n) - \hat{d}(n). \tag{3.4}$$

Elevando ao quadrado ambos os lados da igualdade de (3.4) e tomando o valor esperado obtemos a expressão do Erro Quadrático Médio (MSE –

[1]Se o dicionário de dimensão M é adaptado em tempo real, assumimos que n é suficientemente grande de tal maneira que a dimensão M não aumentará.

Mean Square Error)

$$J_{ms}(n) = E\{e^2(n)\} = E\{d^2(n)\} - 2\boldsymbol{p}_{\kappa d}^{\top}\alpha(n) + \alpha^{\top}(n)\,\boldsymbol{R}_{\kappa\kappa}\,\alpha(n) \qquad (3.5)$$

em que $\boldsymbol{R}_{\kappa\kappa} = E\{\kappa_\omega(n)\,\kappa_\omega^{\top}(n)\}$ é a matriz de autocorrelação da entrada $\boldsymbol{u}(n)$ "kernelizada" e $\boldsymbol{p}_{\kappa d} = E\{d(n)\,\kappa_\omega(n)\}$ é o vetor correlação cruzada entre $\kappa_\omega(n)$ e $d(n)$.

Podemos escrever as componentes da matriz de autocorrelação do *kernel*, como sendo

$$[\boldsymbol{R}_{\kappa\kappa}]_{ij} = \begin{cases} E\{\kappa^2(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i))\}, & \text{se } i = j \\ E\{\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i))\,\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_j))\}, & \text{se } i \neq j \end{cases} \qquad (3.6)$$

com $1 \leq i, j \leq M$. Devido ao fato de $\boldsymbol{u}(\omega_i)$ e $\boldsymbol{u}(\omega_j)$ serem i.i.d., as componentes da diagonal principal $[\boldsymbol{R}_{\kappa\kappa}]_{ii}$ são todas iguais e vamos denominá-las r_{md}. Pelo mesmo motivo as componentes fora da diagonal $[\boldsymbol{R}_{\kappa\kappa}]_{ij}$ também são todas iguais e vamos denominá-las r_{od}.

Como $\boldsymbol{R}_{\kappa\kappa}$ é definida positiva[2], o vetor de pesos ótimo é:

$$\alpha_{\text{opt}} = \boldsymbol{R}_{\kappa\kappa}^{-1}\,\boldsymbol{p}_{\kappa d}. \qquad (3.7)$$

Com isso, o correspondente MSE mínimo é:

$$J_{\text{min}} = E\{d^2(n)\} - \boldsymbol{p}_{\kappa d}^{\top}\,\boldsymbol{R}_{\kappa\kappa}^{-1}\,\boldsymbol{p}_{\kappa d}. \qquad (3.8)$$

Estas expressões são conhecidas como solução de Wiener e Mínimo MSE, em que o sinal de entrada $\boldsymbol{u}(n)$ é substituído pelo vetor entrada "kernelizado" $\kappa_\omega(n)$.

Note em (3.8) que além dos parâmetros convencionais do LMS, es-

[2]A matriz de autocorrelação $\boldsymbol{R}_{\kappa\kappa}$ é por definição semi-definida positiva. Comumente, na análise do comportamento estocástico de algoritmos de filtragem adaptativa, essa matriz é assumida definida positiva. Porém, no caso do algoritmo KLMS com $\boldsymbol{u}(n)$ i.i.d. gaussiana, é possível mostrar, usando resultados apresentados nas próximas seções, que $\boldsymbol{R}_{\kappa\kappa}$ é definida positiva, isso é feito no Apêndice C.

tatísticas do sinal de entrada, números de coeficientes e estatísticas do sinal desejado, a função *kernel* juntamente com seus parâmetros são responsáveis pela qualidade da estimação do KLMS. No Apêndice F um exemplo fornece maiores esclarecimentos a respeito dessa observação.

Para determinar o vetor de pesos ótimo α_{opt} e o mínimo MSE J_{min}[3], precisamos determinar a matriz $\boldsymbol{R}_{\kappa\kappa}$, dada pelas propriedades estatísticas de $\boldsymbol{u}(n)$ e do *kernel*. O cálculo das componentes de $\boldsymbol{R}_{\kappa\kappa}$ usando o *kernel* gaussiano e o *kernel* polinomial é apresentado no próximo capítulo.

3.3 Análise do comportamento do transitório do algoritmo KLMS

A equação de atualização dos coeficientes usando o algoritmo KLMS para o sistema apresentado na Figura 3.1 é (LIU; PRINCIPE; HAYKIN, 2010)

$$\alpha(n+1) = \alpha(n) + \eta\, e(n)\, \kappa_{\omega}(n). \tag{3.9}$$

Definindo o vetor erro nos coeficientes como:

$$\boldsymbol{v}(n) = \alpha(n) - \alpha_{\text{opt}}, \tag{3.10}$$

a respectiva equação recursiva de atualização para o vetor de erro nos coeficientes é

$$\boldsymbol{v}(n+1) = \boldsymbol{v}(n) \; + \eta\, e(n)\, \kappa_{\omega}(n). \tag{3.11}$$

A partir de (3.3), (3.4) e da definição de $\boldsymbol{v}(n)$, a equação para o erro é

$$e(n) = d(n) - \kappa_{\omega}^{\top}(n)\, \boldsymbol{v}(n) - \kappa_{\omega}^{\top}(n)\, \alpha_{\text{opt}} \tag{3.12}$$

e o erro de estimação ótimo,

$$e_0(n) = d(n) - \kappa_{\omega}^{\top}(n)\, \alpha_{\text{opt}}. \tag{3.13}$$

[3]Diferentemente do LMS convencional, em que o Mínimo MSE é igual à potência do ruído, no caso do KLMS isso nem sempre acontece.

Substituindo (3.12) em (3.11) a equação para atualização do vetor de erros nos coeficientes pode ser reescrita como

$$\boldsymbol{v}(n+1) = \boldsymbol{v}(n) + \eta\, d(n)\, \kappa_\omega(n) - \eta\, \kappa_\omega^\top(n)\, \boldsymbol{v}(n)\, \kappa_\omega(n) - \eta\, \kappa_\omega^\top(n)\, \alpha_{\text{opt}}\, \kappa_\omega(n). \tag{3.14}$$

3.3.1 Hipóteses estatísticas simplificadoras

As hipóteses simplificadoras são necessárias para possibilitar que o modelo do comportamento estocástico de $\boldsymbol{v}(n)$ seja construído. Uma importante hipótese simplificadora usada em toda a análise que apresentaremos a seguir é a independência estatística entre $\kappa_\omega(n)\kappa_\omega^\top(n)$ e $\boldsymbol{v}(n)$. Essa hipótese está justificada detalhadamente em (MINKOFF, 2001) e tem sido sucessivamente empregada em várias análises de filtros adaptativos. Denominaremos essa hipótese, para futuras referências, de "hipótese da independência modificada" (MIA)[4].

Outras hipóteses estatísticas serão necessárias para completar nossa análise. Por motivos didáticos, essas hipóteses serão discutidas e justificadas à medida em que forem apresentadas ao longo do texto.

3.3.2 Comportamento médio dos coeficientes

Calculando o valor esperado em ambos os lados de (3.14) e usando MIA obtemos

$$E\{\boldsymbol{v}(n+1)\} = (\boldsymbol{I} - \eta\, \boldsymbol{R}_{\kappa\kappa})\, E\{\boldsymbol{v}(n)\}\,. \tag{3.15}$$

Note que a Equação (3.15) representa o comportamento médio dos coeficientes do LMS quando a entrada é o vetor $\kappa_\omega(n)$.

[4] Esta hipótese é mostrada como sendo menos restritiva que a hipótese da independência clássica (MINKOFF, 2001)

3.3.3 Erro quadrático médio – MSE

Usando (3.12) e MIA, os momentos de segunda ordem dos pesos são relacionados com MSE a partir de (SAYED, 2003)

$$J_{ms}(n) = J_{\min} + \text{trace}\{\boldsymbol{R}_{\kappa\kappa}\, \boldsymbol{C}_{\boldsymbol{v}}(n)\} \tag{3.16}$$

em que $\boldsymbol{C}_{\boldsymbol{v}}(n) = E\{\boldsymbol{v}(n)\,\boldsymbol{v}^{\top}(n)\}$ é a matriz de autocorrelação de $\boldsymbol{v}(n)$, $J_{\min} = E\{e_0^2(n)\}$ o mínimo MSE e $\text{trace}\{\boldsymbol{R}_{\kappa\kappa}\, \boldsymbol{C}_{\boldsymbol{v}}(n)\} = J_{ex}(n)$. O estudo do comportamento do MSE (3.16) requer um modelo para $\boldsymbol{C}_{\boldsymbol{v}}(n)$. Este modelo é altamente afetado pela transformação imposta pelo *kernel* no sinal de entrada $\boldsymbol{u}(n)$. Um modelo analítico para o comportamento de $\boldsymbol{C}_{\boldsymbol{v}}(n)$ está deduzido na próxima subseção.

3.3.4 Comportamento dos momentos de segunda ordem do vetor de erros

Usando (3.13) e (3.14), a atualização do vetor de erro nos coeficientes é

$$\boldsymbol{v}(n+1) = \boldsymbol{v}(n) + \eta\, e_0(n)\, \kappa_{\omega}(n) - \eta\, \kappa_{\omega}(n)\, \kappa_{\omega}^{\top}(n)\, \boldsymbol{v}(n). \tag{3.17}$$

Pós-multiplicando por (3.17) por sua transposta e calculando o valor esperado, obtemos uma expressão para a matriz de correlação do vetor de erro nos coeficientes, que é

$$\begin{aligned}
\boldsymbol{C}_{\boldsymbol{v}}(n+1) = \boldsymbol{C}_{\boldsymbol{v}}(n) &- \eta\, E\{\kappa_{\omega}(n)\, \kappa_{\omega}^{\top}(n)\, \boldsymbol{v}(n)\, \boldsymbol{v}^{\top}(n)\} \\
&- \eta\, E\{\boldsymbol{v}(n)\, \boldsymbol{v}^{\top}(n)\, \kappa_{\omega}(n)\, \kappa_{\omega}^{\top}(n)\} \\
&+ \eta^2\, E\{e_0^2(n)\, \kappa_{\omega}(n)\, \kappa_{\omega}^{\top}(n)\} + \eta\, E\{e_0(n)\, \kappa_{\omega}(n)\, \boldsymbol{v}^{\top}(n)\} \\
&- \eta^2\, E\{e_0(n)\, \kappa_{\omega}(n)\, \boldsymbol{v}^{\top}(n)\, \kappa_{\omega}(n)\, \kappa_{\omega}^{\top}(n)\} \\
&+ \eta\, E\{e_0(n)[\kappa_{\omega}(n)\, \boldsymbol{v}^{\top}(n)]^{\top}\} \\
&- \eta^2\, E\{e_0(n)[\kappa_{\omega}(n)\, \boldsymbol{v}^{\top}(n)\, \kappa_{\omega}(n)\, \kappa_{\omega}^{\top}(n)]^{\top}\} \\
&+ \eta^2\, E\{\kappa_{\omega}(n)\, \kappa_{\omega}^{\top}(n)\, \boldsymbol{v}(n)\, \boldsymbol{v}^{\top}(n)\, \kappa_{\omega}(n)\, \kappa_{\omega}^{\top}(n)\}.
\end{aligned} \tag{3.18}$$

Devemos calcular agora cada um dos valores esperados que compõem (3.18).

Usando MIA, os dois primeiros valores esperados serão dados por:

$$\begin{aligned} E\{\kappa_\omega(n)\,\kappa_\omega^\top(n)\,\boldsymbol{v}(n)\,\boldsymbol{v}^\top(n)\} &= \boldsymbol{R}_{\kappa\kappa}\,\boldsymbol{C}_{\boldsymbol{v}}(n) \\ E\{\boldsymbol{v}(n)\,\boldsymbol{v}^\top(n)\,\kappa_\omega(n)\,\kappa_\omega^\top(n)\} &= \boldsymbol{C}_{\boldsymbol{v}}(n)\,\boldsymbol{R}_{\kappa\kappa}. \end{aligned} \tag{3.19}$$

Para obter o valor do terceiro momento, assumiremos que o modelo de ordem finita provê uma aproximação suficientemente próxima do modelo de ordem infinita com mínimo MSE, de modo que $E\{e_0(n)\} \approx 0$. Também iremos considerar que $e_0(n)$ e $\kappa_\omega(n)\,\kappa_\omega^\top(n)$ são descorrelacionados. Esta última hipótese é garantida pelos argumentos que sustentam a hipótese MIA (MINKOFF, 2001). Então,

$$\begin{aligned} E\{e_0^2(n)\,\kappa_\omega(n)\,\kappa_\omega^\top(n)\} &\approx E\{e_0^2(n)\}\,E\{\kappa_\omega(n)\,\kappa_\omega^\top(n)\} \\ &= \boldsymbol{R}_{\kappa\kappa}\,J_{\min}. \end{aligned} \tag{3.20}$$

O quarto e o sexto valor esperado podem ser aproximados usando MIA, como

$$E\{e_0(n)\,\kappa_\omega(n)\,\boldsymbol{v}^\top(n)\} \approx E\{e_0(n)\,\kappa_\omega(n)\}\,E\{\boldsymbol{v}^\top(n)\} = 0 \tag{3.21}$$

já que $E\{e_0(n)\,\kappa_\omega(n)\} = 0$ pelo princípio da ortogonalidade (SAYED, 2003).

O cálculo do quinto e do sétimo valor esperado requer simplificações adicionais para tratabilidade matemática. Uma razoável aproximação que preserva o efeito de $\kappa_\omega(n)$ até os seus momentos de segunda ordem é assumir que $e_0(n)\,\kappa_\omega(n)$ e $\kappa_\omega(n)\,\kappa_\omega^\top(n)$ são descorrelacionados[5]. Assumindo ambos, essa aproximação e MIA, temos

$$\begin{aligned} E\{e_0(n)\,\kappa_\omega(n)\,\boldsymbol{v}^\top(n)\,\kappa_\omega(n)\,\kappa_\omega^\top(n)\} &= E\{\boldsymbol{v}^\top(n)\,e_0(n)\,\kappa_\omega(n)\,\kappa_\omega(n)\,\kappa_\omega^\top(n)\} \\ &\approx E\{\boldsymbol{v}^\top(n)\}\,E\{e_0(n)\,\kappa_\omega(n)\}\,E\{\kappa_\omega^\top(n)\,\kappa_\omega(n)\} = 0 \end{aligned} \tag{3.22}$$

[5] Usando essa aproximação, basicamente negligenciamos as flutuações de $\kappa_\omega(n)\,\kappa_\omega^\top(n)$ sobre sua média $E\{\kappa_\omega(n)\,\kappa_\omega^\top(n)\}$.

em que a igualdade a zero é proveniente do princípio da ortogonalidade.

Usando (3.19)–(3.22) em (3.18) obtemos a expressão recursiva para matriz correlação do vetor de erro nos coeficientes

$$\boldsymbol{C_v}(n+1) \approx \boldsymbol{C_v}(n) - \eta(\boldsymbol{R}_{\kappa\kappa}\,\boldsymbol{C_v}(n) + \boldsymbol{C_v}(n)\,\boldsymbol{R}_{\kappa\kappa}) + \eta^2\,\boldsymbol{T}(n) + \eta^2\,\boldsymbol{R}_{\kappa\kappa}\,J_{\min} \tag{3.23a}$$

com

$$\boldsymbol{T}(n) = E\{\kappa_\omega(n)\,\kappa_\omega^\top(n)\,\boldsymbol{v}(n)\,\boldsymbol{v}^\top(n)\,\kappa_\omega(n)\,\kappa_\omega^\top(n)\}. \tag{3.23b}$$

A estimação do valor esperado (3.23b) é um importante passo na análise. Na análise clássica do LMS (HAYKIN, 1991), o sinal de entrada é assumido como gaussiano de média nula. Então o valor esperado em (3.23b) podia ser aproximado usando o teorema da fatoração de momentos para variáveis gaussianas (MILLER, 1963). Na presente análise, $\kappa_\omega(n)$ é uma transformação não-linear de uma função quadrática do vetor de entrada gaussiano $\boldsymbol{u}(n)$, assim $\kappa_\omega(n)$ não é de média nula e nem tem distribuição gaussiana.

Usando MIA para determinar a (i,j)-ésima componente de $\boldsymbol{T}(n)$ em (3.23b) temos

$$[\,\boldsymbol{T}(n)]_{ij} \approx \sum_{\ell=1}^{M}\sum_{p=1}^{M} E\{\kappa_{\omega_i}(n)\,\kappa_{\omega_\ell}(n)\,\kappa_{\omega_p}(n)\,\kappa_{\omega_j}(n)\}\,[\,\boldsymbol{C_v}(n)]_{\ell p} \tag{3.24}$$

em que $\kappa_{\omega_m}(n) = \kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_m))$, com $m = i, j, p, \ell$. Na próxima seção vamos apresentar uma maneira para obter cada um dos momentos de (3.24). Dependendo de i, j, ℓ e p, temos cinco diferentes momentos de quarta ordem μ_k, $k = 1, \ldots, 5$, à saber:

Caso $i = j = p = \ell$, definimos

$$\mu_1 := E\{\kappa_{\omega_i}^4(n)\}\,. \tag{3.25}$$

Caso $i = j = p \neq \ell$, definimos

$$\mu_2 := E\{\kappa^3_{\omega_i}(n)\,\kappa_{\omega_\ell}(n)\}\,. \tag{3.26}$$

Caso $i = j \neq p = \ell$, definimos

$$\mu_3 := E\{\kappa^2_{\omega_i}(n)\,\kappa^2_{\omega_\ell}(n)\}\,. \tag{3.27}$$

Caso $i = j \neq p \neq \ell$, definimos

$$\mu_4 := E\{\kappa^2_{\omega_i}(n)\,\kappa_{\omega_\ell}(n)\,\kappa_{\omega_p}(n)\}\,. \tag{3.28}$$

Caso $i \neq j \neq p \neq \ell$, o momento de quarta ordem será dado por:

$$\mu_5 := E\{\kappa_{\omega_i}(n)\,\kappa_{\omega_\ell}(n)\,\kappa_{\omega_p}(n)\,\kappa_{\omega_j}(n)\}\,. \tag{3.29}$$

Usando estes momentos, os elementos de $\boldsymbol{T}(n)$ são finalmente calculados por

$$\begin{aligned}[\boldsymbol{T}(n)]_{ii} =& \mu_1[\boldsymbol{C_v}(n)]_{ii} + \sum_{\substack{\ell=1\\ \ell\neq i}}^{M}\Big\{2\mu_2[\boldsymbol{C_v}(n)]_{i\ell} + \mu_3[\boldsymbol{C_v}(n)]_{\ell\ell} \\ &+ \mu_4 \sum_{\substack{p=1\\ p\neq\{i,\ell\}}}^{M}[\boldsymbol{C_v}(n)]_{\ell p}\Big\}\end{aligned} \tag{3.30}$$

e, para $j \neq i$,

$$\begin{aligned}[\boldsymbol{T}(n)]_{ij} =& \mu_2\left([\boldsymbol{C_v}(n)]_{ii} + [\boldsymbol{C_v}(n)]_{jj}\right) + 2\mu_3[\boldsymbol{C_v}(n)]_{ij} \\ &+ \sum_{\substack{\ell=1\\ \ell\neq\{i,j\}}}^{M}\Big\{2\mu_4[\boldsymbol{C_v}(n)]_{j\ell} + 2\mu_4[\boldsymbol{C_v}(n)]_{i\ell} + \mu_4[\boldsymbol{C_v}(n)]_{\ell\ell} \\ &+ \mu_5 \sum_{\substack{p=1\\ p\neq\{i,j,\ell\}}}^{M}[\boldsymbol{C_v}(n)]_{\ell p}\Big\}\end{aligned} \tag{3.31}$$

o que completa a expressão $\boldsymbol{T}(n)$ em (3.23b). Substituindo o resultado em (3.23a) obtemos a expressão recursiva para entradas da matriz de autocorrelação $\boldsymbol{C}_{\boldsymbol{v}}(n)$:

$$\begin{aligned}[\boldsymbol{C}_{\boldsymbol{v}}(n+1)]_{ii} =& (1-2\eta r_{\text{md}}+\eta^2\mu_1)[\boldsymbol{C}_{\boldsymbol{v}}(n)]_{ii} + \eta^2\mu_3 \sum_{\substack{\ell=1\\ \ell\neq i}}^{M} [\boldsymbol{C}_{\boldsymbol{v}}(n)]_{\ell\ell} \\ &+ (2\eta^2\mu_2 - 2\eta r_{\text{od}}) \sum_{\substack{\ell=1\\ \ell\neq i}}^{M} [\boldsymbol{C}_{\boldsymbol{v}}(n)]_{i\ell} + \eta^2\mu_4 \sum_{\substack{\ell=1\\ \ell\neq i}}^{M} \sum_{\substack{p=1\\ p\neq\{i,\ell\}}}^{M} [\boldsymbol{C}_{\boldsymbol{v}}(n)]_{\ell p} \\ &+ \eta^2\, r_{\text{md}}\, J_{\min}\end{aligned} \tag{3.32}$$

e, para $j \neq i$,

$$\begin{aligned}[\boldsymbol{C}_{\boldsymbol{v}}(n+1)]_{ij} =& (1-2\eta r_{\text{md}}+2\eta^2\mu_3)[\boldsymbol{C}_{\boldsymbol{v}}(n)]_{ij} + \eta^2\mu_4 \sum_{\substack{\ell=1\\ \ell\neq\{i,j\}}}^{M} [\boldsymbol{C}_{\boldsymbol{v}}(n)]_{\ell\ell} \\ &+ (\eta^2\mu_2 - \eta r_{\text{od}})([\boldsymbol{C}_{\boldsymbol{v}}(n)]_{ii} + [\boldsymbol{C}_{\boldsymbol{v}}(n)]_{jj}) \\ &+ (2\eta^2\mu_4 - \eta r_{\text{od}}) \sum_{\substack{\ell=1\\ \ell\neq\{i,j\}}}^{M} ([\boldsymbol{C}_{\boldsymbol{v}}(n)]_{i\ell} + [\boldsymbol{C}_{\boldsymbol{v}}(n)]_{j\ell}) \\ &+ \eta^2\mu_5 \sum_{\substack{\ell=1\\ \ell\neq\{i,j\}}}^{M} \sum_{\substack{p=1\\ p\neq\{i,j,\ell\}}}^{M} [\boldsymbol{C}_{\boldsymbol{v}}(n)]_{\ell p} + \eta^2\, r_{\text{od}}\, J_{\min}\end{aligned} \tag{3.33}$$

em que $r_{\text{md}} = [\boldsymbol{R}_{\kappa\kappa}]_{ii}$ e $r_{\text{od}} = [\boldsymbol{R}_{\kappa\kappa}]_{ij}$ com $j \neq i$, como definido em (3.6).

3.3.5 Algumas desigualdades úteis

Antes de concluir esta seção, vamos derivar algumas desigualdades que relacionam os momentos de quarta ordem μ_i e as componentes da matriz

$\boldsymbol{R}_{\kappa\kappa}$. Para variáveis aleatórias reais X e Y, lembrando que a desigualdade de Hölder diz (GRIMMETT; STIRZAKER, 2001)

$$E\{|XY|\} \leq E\{|X|^p\}^{\frac{1}{p}} E\{|Y|^q\}^{\frac{1}{q}} \tag{3.34}$$

em que p em q estão em $(1, +\infty)$ com $\frac{1}{p} + \frac{1}{q} = 1$. Como mostrado a seguir, isto gera a seguinte desigualdade

$$\mu_3 \leq \mu_1. \tag{3.35}$$

A desigualdade (3.35) pode ser obtida diretamente da Equação (3.34) com $X = \kappa^2_{\omega_i}(n)$, $Y = \kappa^2_{\omega_\ell}(n)$, $p = q = 2$,

$$\mu_3 = E\{\kappa^2_{\omega_i}(n)\kappa^2_{\omega_\ell}(n)\} \leq E\{\kappa^4_{\omega_i}(n)\}^{\frac{1}{2}} E\{\kappa^4_{\omega_\ell}(n)\}^{\frac{1}{2}} = E\{\kappa^4_{\omega_i}(n)\} = \mu_1 \tag{3.36}$$

em que a penúltima desigualdade segue da estacionaridade de $\kappa_\omega(n)$.

Uma outra desigualdade bastante útil envolve a componente r_{md} da matriz de autocorrelação $\boldsymbol{R}_{\kappa\kappa}$. Pela desigualdade de Chebyshev (PAPOULIS, 1991), temos

$$r^2_{\text{md}} \leq \mu_3. \tag{3.37}$$

Na próxima seção, vamos usar a expressão recursiva de (3.32) e (3.33) da matriz de autocorrelação $\boldsymbol{C}_{\boldsymbol{v}}(n)$, e as desigualdades (3.35) e (3.37), para estudar o comportamento do regime permanente do algoritmo KLMS.

3.4 Análise de convergência do KLMS para kernels positivos

Vamos agora determinar as condições de convergência para o modelo do algoritmo KLMS obtido na Seção 3.3. Seja $\boldsymbol{c}_{\boldsymbol{v}}(n)$ a representação lexicográfica de $\boldsymbol{C}_{\boldsymbol{v}}(n)$, ou seja, cada coluna da matriz $\boldsymbol{C}_{\boldsymbol{v}}(n)$ é empilhada formando o vetor coluna $\boldsymbol{c}_{\boldsymbol{v}}(n)$. Considere a família de matrizes $\boldsymbol{H}^{ij}$, $1 \leq$

$i, j \leq M$, da ordem $(M \times M)$, as quais têm suas componentes dadas por

$$\text{se } (i = j): \begin{cases} [\boldsymbol{H}^{ii}]_{ii} = 1 - 2\eta r_{\text{md}} + \eta^2 \mu_1, & \\ [\boldsymbol{H}^{ii}]_{pp} = \eta^2 \mu_3, & p \neq i \\ [\boldsymbol{H}^{ii}]_{ip} = \eta^2 \mu_2 - \eta r_{\text{od}} = [\boldsymbol{H}^{ii}]_{pi}, & p \neq i \\ [\boldsymbol{H}^{ii}]_{p\ell} = \eta^2 \mu_4, & \text{caso contrário,} \end{cases} \tag{3.38}$$

$$\text{se } (i \neq j): \begin{cases} [\boldsymbol{H}^{ij}]_{ij} = [\boldsymbol{H}^{ij}]_{ji} = \frac{1}{2}(1 - 2\eta r_{\text{md}} + 2\eta^2 \mu_3) & \\ [\boldsymbol{H}^{ij}]_{pp} = \eta^2 \mu_4, & p \neq i, j \\ [\boldsymbol{H}^{ij}]_{ii} = [\boldsymbol{H}^{ij}]_{jj} = \eta^2 \mu_2 - \eta r_{\text{od}}, & \\ [\boldsymbol{H}^{ij}]_{ip} = [\boldsymbol{H}^{ij}]_{pi} = \frac{1}{2}(2\eta^2 \mu_4 - \eta r_{\text{od}}), & p \neq i, j \\ [\boldsymbol{H}^{ij}]_{pj} = [\boldsymbol{H}^{ij}]_{jp} = \frac{1}{2}(2\eta^2 \mu_4 - \eta r_{\text{od}}), & p \neq i, j \\ [\boldsymbol{H}^{ij}]_{p\ell} = \eta^2 \mu_5, & \text{caso contrário.} \end{cases} \tag{3.39}$$

Finalmente, definimos a matriz $\boldsymbol{G}$, $(M^2 \times M^2)$, como

$$\boldsymbol{G} = \left[\boldsymbol{h}^{11}\ \boldsymbol{h}^{12}\ \ldots\ \boldsymbol{h}^{1M}\ \ldots\ \boldsymbol{h}^{MM}\right] \tag{3.40}$$

com $\boldsymbol{h}^{\ell p}$ um vetor $(M^2 \times 1)$ que é a representação lexicográfica de $\boldsymbol{H}^{\ell p}$. Usando estas definições, podemos mostrar que a representação lexicográfica da recursão (3.23a) pode ser escrita como:

$$\boldsymbol{c_v}(n+1) = \boldsymbol{G}\ \boldsymbol{c_v}(n) + \eta^2 J_{\min}\, r_{\kappa\kappa} \tag{3.41}$$

em que $\boldsymbol{c_v}(n)$ é a representação lexicográfica de $\boldsymbol{C_v}(n)$ e $r_{\kappa\kappa}$ é a representação lexicográfica de $\boldsymbol{R}_{\kappa\kappa}$.

Podemos usar a Expressão (3.41) para obter as condições de estabilidade do algoritmo KLMS. Mas antes de encerrarmos esta seção, devemos

deixar claro que $\boldsymbol{G}$ é simétrica. Isto pode ser mostrado a partir das Expressões (3.38)-(3.39) usando $[\boldsymbol{H}^{ij}]_{\ell p} = [\boldsymbol{G}]_{(i-1)M+j,(\ell-1)M+p}$, e observando que $[\boldsymbol{H}^{ij}]_{\ell p} = [\boldsymbol{H}^{\ell p}]_{ij}$. Isto implica que $\boldsymbol{G}$ pode ser diagonalizável por uma transformação unitária[6], e todos os seus autovalores são reais.

3.4.1 Condições de convergência para kernels de valores positivos

Para o desenvolvimento desta subseção vamos considerar os *kernels* de valores positivos. Entenda por essa classe de *kernel* aqueles que produzem $\kappa(\boldsymbol{x}, \boldsymbol{x}') > 0$ quaisquer que sejam $\boldsymbol{x}$ e $\boldsymbol{x}'$ de um espaço vetorial $\mathcal{X}$. Como exemplo podemos citar o *kernel* gaussiano, *kernel* laplaciano e *kernel* polinomial de grau par, entre outros.

Uma condição necessária e suficiente para convergência de $\boldsymbol{c_v}(n)$ em (3.41) é que todos os autovalores de $\boldsymbol{G}$ devem pertencer ao intervalo aberto $(-1, 1)$ (LUENBERGER, 1979, Seção 5.9). Assim, o limite de estabilidade para η pode ser numericamente determinado para dados valores de M e parâmetro do *kernel*. A seguir, definiremos um conjunto de condições suficientes que podem ser também usadas nas propostas de projeto.

Pela bem conhecida teoria dos discos de Gerschgorin, sabemos que os autovalores de $\boldsymbol{G}$ estão dentro da união dos discos de Gerschgorin (GOLUB; LOAN, 1996). Cada um destes discos está centrado em uma componente da diagonal de $\boldsymbol{G}$ e tem um raio dado pela soma dos valores absolutos das demais componentes da mesma linha. Uma condição suficiente para estabilidade de (3.41) é dada por

$$|[\boldsymbol{G}]_{ii}| + \sum_{\substack{\ell=1 \\ \ell \neq i}}^{M^2} |[\boldsymbol{G}]_{i\ell}| < 1, \quad \text{for } i = 1, \ldots, M^2. \tag{3.42}$$

As Equações (3.38)–(3.40) mostram que as linhas de $\boldsymbol{G}$ têm apenas duas

[6] Transformações unitárias são do tipo $\boldsymbol{G} = \boldsymbol{U}\boldsymbol{G}'\boldsymbol{U}^H$, é tal que $\boldsymbol{U}^{-1} = \boldsymbol{U}^H$ em que $\boldsymbol{U}^H$ é a conjugada transposta de $\boldsymbol{U}$.

formas distintas, no sentido de que cada linha de $\boldsymbol{G}$ é definida como uma das duas linhas distintas. Isto implica que somente dois discos distintos de Gerschgorin serão definidos. Note que, exceto $[\boldsymbol{G}]_{i\ell} = \eta^2\mu_2 - \eta r_{\text{od}}$ e $[\boldsymbol{G}]_{i\ell} = \frac{1}{2}(2\eta^2\mu_4 - \eta r_{\text{od}})$, todas as demais componentes de $\boldsymbol{G}$ são positivas. Veja no Apêndice D a prova de que a Expressão (3.42) define apenas duas desigualdades, em que para o caso $M \geq 3$ são:

$$(1 - 2\eta r_{\text{md}} + \eta^2\mu_1) + (M-1)\eta^2\mu_3 + 2(M-1)|\eta^2\mu_2 - \eta r_{\text{od}}| + (M-1)(M-2)\eta^2\mu_4 < 1, \tag{3.43a}$$

$$(1 - 2\eta r_{\text{md}} + 2\eta^2\mu_3) + 2|\eta^2\mu_2 - \eta r_{\text{od}}| + (M-2)\eta^2\mu_4 + 2(M-2)|2\eta^2\mu_4 - \eta r_{\text{od}}| + (M-2)(M-3)\eta^2\mu_5 < 1. \tag{3.43b}$$

O estudo dos valores de η que satisfaçam ambas desigualdades (3.43a) e (3.43b) define o passo máximo para a convergência. A interseção destes limites é uma condição suficiente para estabilidade. A seguir, apresentaremos a análise em que $M \geq 3$. Os resultados para o estudo da estabilidade nos casos $M = 1$ e $M = 2$ estão apresentados no Apêndice E.

Para determinar os limites impostos por (3.43a), vamos reescrevê-la como

$$2(M-1)|\eta\mu_2 - r_{\text{od}}| < 2r_{\text{md}} - \eta\mu_1 - (M-1)\eta\mu_3 - (M-1)(M-2)\eta\mu_4. \tag{3.44}$$

Assim, as seguintes condições devem ser satisfeitas

$$2(M-1)(\eta\mu_2 - r_{\text{od}}) < 2r_{\text{md}} - \eta\mu_1 - (M-1)\eta\mu_3 - (M-1)(M-2)\eta\mu_4 \tag{3.45a}$$

$$2(M-1)(\eta\mu_2 - r_{\text{od}}) > -2r_{\text{md}} + \eta\mu_1 + (M-1)\eta\mu_3 + (M-1)(M-2)\eta\mu_4. \tag{3.45b}$$

Como $\mu_i > 0$[7] para todo $1 \leq i \leq 5$, a condição (3.45a) produz

$$\eta < \frac{2r_{\text{md}} + 2(M-1)r_{\text{od}}}{\mu_1 + 2(M-1)\mu_2 + (M-1)\mu_3 + (M-1)(M-2)\mu_4} := \eta_1 \tag{3.46}$$

Note que o numerador e o denominador de η_1 são ambos positivos.

Em contrapartida, a desigualdade (3.45b) nos leva a uma condição da forma $\theta_2\eta < \theta_1$ com

$$\theta_1 = 2r_{\text{md}} - 2(M-1)r_{\text{od}} \tag{3.47}$$

$$\theta_2 = \mu_1 - 2(M-1)\mu_2 + (M-1)\mu_3 + (M-1)(M-2)\mu_4. \tag{3.48}$$

A seguir, denotaremos por $\eta_2 := \frac{\theta_1}{\theta_2}$ o limite resultante em η. Então, resolvendo (3.45b) temos quatro possíveis casos que são apresentados na Tabela 3.1, dependendo dos sinais de θ_1 e θ_2. No caso (ii), note que não existe $\eta > 0$ que satisfaça a desigualdade (3.45b) quando $\theta_1 < 0$ e $\theta_2 > 0$. Esta situação surge porque a condição (3.43a) é uma condição suficiente que impõe que todos os discos de Gerschgorin definidos por (3.42) estejam completamente no interior do círculo unitário no plano z. Esta condição não é, obviamente, necessária para que todos os autovalores de $\boldsymbol{G}$ estejam no círculo unitário. O limite inferior para o caso (iv) da Tabela 3.1 é também devido a esta forte restrição, pois o algoritmo é certamente estável para $\eta = 0$. Será mantido aqui para que seja completa a análise da estabilidade, mas ela deve ser descartada na prática. Combinando as possíveis soluções de (3.45a) e (3.45b) e descartando o limite inferior obtemos os seguintes limites de

[7] Esta relação envolvendo os momentos de quarta ordem só é óbvia para *kernel* positivos.

Tabela 3.1: Restrições em η obtidas a partir de (3.45b)

Caso	Sinal de θ_1 (3.47)	Sinal de θ_2 (3.48)	Restrição	Observação
(i)	+	+	$0 \leq \eta < \eta_2$	
(ii)	+	−	$0 < \eta$	Sem restrição adicional
(iii)	−	+	$\eta < \eta_2 < 0$	Não existe $\eta > 0$
(iv)	−	−	$0 < \eta_2 < \eta$	

estabilidade para η:

$$\begin{cases} \eta < \min\{\eta_1, \eta_2\}, & \text{para o caso } (i) \text{ na Tabela 3.1} \\ \eta < \eta_1, & \text{para o caso } (ii) \text{ e } (iv) \text{ na Tabela 3.1.} \end{cases} \tag{3.49}$$

Se o caso (iii) ocorrer, o qual pode ser testado de imediato, as condições de estabilidade devem ser determinadas numericamente a partir dos autovalores de $\boldsymbol{G}$.

Tendo determinado os limites para (3.43a) tal que $\eta > 0$, e assumindo que o caso (iii) na Tabela 3.1 não ocorre, procedemos para determinar as restrições extras impostas em η pela desigualdade (3.43b). Primeiramente, multiplicaremos (3.43b) por $(M-1)$ e dividiremos por η, reescrevendo-a como

$$\begin{aligned} 2(M-1)\,|\mu_2\eta - r_{\text{od}}| < 2(M-1)\,r_{\text{md}} - 2(M-1)\,\mu_3\eta - (M-1)(M-2)\,\mu_4\eta \\ - 2(M-1)(M-2)\,|2\,\mu_4\eta - r_{\text{od}}| \\ - (M-1)(M-2)(M-3)\,\eta\,\mu_5. \end{aligned} \tag{3.50}$$

Agora, dada que a condição (3.44) já tenha sido cumprida, trocaremos o lado esquerdo de (3.50) com o lado direito de (3.44). Após reorganizar os termos,

temos a nova condição

$$2(M-1)(M-2)\,|2\,\mu_4\,\eta - r_{\text{od}}| < 2(M-2)\,r_{\text{md}} + \mu_1\,\eta - (M-1)\,\mu_3\,\eta - (M-1)(M-2)(M-3)\,\mu_5\,\eta \tag{3.51}$$

a qual nos leva às duas restrições a seguir

$$4(M-1)(M-2)\,\mu_4\,\eta - 2(M-1)(M-2)\,r_{\text{od}} < 2(M-2)\,r_{\text{md}} + \mu_1\,\eta - (M-1)\,\mu_3\,\eta - (M-1)(M-2)(M-3)\,\mu_5\,\eta \tag{3.52a}$$

$$-2(M-2)\,r_{\text{md}} - \mu_1\,\eta + (M-1)\,\mu_3\,\eta + (M-1)(M-2)(M-3)\,\mu_5\,\eta < 4(M-1)(M-2)\,\mu_4\,\eta - 2(M-1)(M-2)\,r_{\text{od}}. \tag{3.52b}$$

Por um lado, a desigualdade (3.52a) gera duas diferentes restrições definidas por

$$\begin{cases} \eta < \dfrac{2(M-2)\,(r_{\text{md}} + (M-1)\,r_{\text{od}})}{\theta_3} := \eta_3, & \text{se } \theta_3 > 0 \\ \eta > 0, & \text{se } \theta_3 < 0 \end{cases} \tag{3.53}$$

em que

$$\theta_3 = -\mu_1 + (M-1)\,\mu_3 + 4(M-1)(M-2)\,\mu_4 + (M-1)(M-2)(M-3)\,\mu_5. \tag{3.54}$$

Por outro lado, a desigualdade (3.52b) nos leva a uma condição na forma

Tabela 3.2: Restrições em η obtidas a partir de (3.52b)

Caso	Sinal de θ_1 (3.47)	Sinal de θ_4 (3.55)	Restrição	Observação
(v)	+	+	$0 \leq \eta < \eta_4$	
(vi)	+	−	$0 < \eta$	Sem restrição adicional
(vii)	−	+	$\eta < \eta_4 < 0$	Não existe $\eta > 0$
$(viii)$	−	−	$0 < \eta_4 < \eta$	

$\theta_4\,\eta < (M-2)\,\theta_1$, em que θ_1 já foi definido em (3.47) e

$$\theta_4 = -\mu_1 + (M-1)\,\mu_3 - 4(M-1)(M-2)\,\mu_4 + (M-1)(M-2)(M-3)\,\mu_5. \tag{3.55}$$

Resolvendo (3.52b) temos quatro possíveis casos a considerar, apresentados na Tabela 3.2, em que $\eta_4 := \frac{(M-2)\,\theta_1}{\theta_4}$.

Combinando as possíveis soluções de (3.52a) e (3.52b), e novamente descartando os limites inferiores, obtemos os seguintes limites em η

$$\text{Se } \theta_3 > 0, \begin{cases} \eta < \min\{\eta_3, \eta_4\}, & \text{para o caso } (v) \text{ na Tabela 3.2} \\ \eta < \eta_3, & \text{para o caso } (vi) \text{ e } (viii) \text{ na Tabela 3.2} \end{cases}$$
$$\text{Se } \theta_3 < 0, \begin{cases} \eta < \eta_4, & \text{para o caso } (v) \text{ na Tabela 3.2} \\ \eta > 0, & \text{para o caso } (vi) \text{ e } (viii) \text{ na Tabela 3.2.} \end{cases} \tag{3.56}$$

Finalmente, exceto para os casos (iii) na Tabela 3.1 e (vii) na Tabela 3.2, os quais devem ser testados de imediato, a condição de estabilidade suficiente será dada pela interseção das condições (3.49) e (3.56).

Na próxima subseção, apresentaremos a dedução da expressão da matriz correlação do vetor de erro nos coeficientes $\boldsymbol{C}_v(n)$ em regime permanente. Essa expressão será utilizada no cálculo do regime permanente do MSE e do MSE em excesso.

3.4.2 Análise do algoritmo em regime permanente

A forma fechada da solução de (3.41) pode ser escrita como (LUENBERGER, 1979):

$$\boldsymbol{c_v}(n) = \boldsymbol{G}^n\,[\,\boldsymbol{c_v}(0) - \boldsymbol{c_v}(\infty)] + \boldsymbol{c_v}(\infty) \tag{3.57}$$

em que $\boldsymbol{c_v}(\infty)$ denota o vetor $\boldsymbol{c_v}(n)$ em regime permanente, e é dado por

$$\boldsymbol{c_v}(\infty) = \eta^2\, J_{\min}\,(\boldsymbol{I} - \boldsymbol{G})^{-1}\, r_{\kappa\kappa}. \tag{3.58}$$

Assumindo convergência, podemos definir o tempo para convergência como o número n_ϵ de iterações necessárias para que (3.57) satisfaça a condição

$$\|\,\boldsymbol{c_v}(n) - \boldsymbol{c_v}(n_\epsilon)\| \leq \epsilon \tag{3.59}$$

em que ϵ é um parâmetro de projeto selecionado pelo usuário.

Neste ponto, é importante notar que $\boldsymbol{c_v}(\infty)$ é único se o sistema em consideração é estável. De fato, a matriz $(\boldsymbol{G} - \boldsymbol{I})$ tem apenas autovalores não nulos porque satisfaz as restrições (3.43a)–(3.43b), e pode ser então invertida. Seja $\boldsymbol{C_v}(\infty)$, a qual tem a representação lexicográfica dada por $\boldsymbol{c_v}(\infty)$. Esta representação é única e satisfaz a seguinte expressão que foi obtida a partir da Equação (3.23a) com $n \to \infty$

$$\boldsymbol{R}_{\kappa\kappa}\,\boldsymbol{C_v}(\infty) + \boldsymbol{C_v}(\infty)\,\boldsymbol{R}_{\kappa\kappa} - \eta\,\boldsymbol{T}(\infty) = \eta\,\boldsymbol{R}_{\kappa\kappa}\,J_{\min}. \tag{3.60}$$

Em (3.60), $\boldsymbol{T}(\infty)$ representa o regime permanente da matriz $\boldsymbol{T}(n)$. É interessante notar que (3.58) é a contra-representação lexicográfica de (3.60), como (3.41) é a representação lexicográfica de (3.23a).

A partir de (3.16), o regime permanente do MSE é dado por

$$J_{ms}(\infty) = J_{\min} + \text{trace}\{\boldsymbol{R}_{\kappa\kappa}\,\boldsymbol{C_v}(\infty)\} \tag{3.61}$$

no qual, trace$\{\boldsymbol{R}_{\kappa\kappa}\,\boldsymbol{C_v}(\infty)\}$ é o regime permanente do MSE em excesso,

$$J_{ex}(\infty) = \text{trace}\{\boldsymbol{R}_{\kappa\kappa}\,\boldsymbol{C_v}(\infty)\} \tag{3.62}$$

Para obter estes valores, devemos agora calcular as componentes de $\boldsymbol{C_v}(\infty)$. A fim de fazê-lo, começamos justificando que $\boldsymbol{C_v}(\infty)$ é uma matriz com as mesmas propriedades estruturais de $\boldsymbol{R}_{\kappa\kappa}$, a saber, todas as componentes da diagonal principal são iguais entre si, e com este valor aqui denotado por c_{md}, e todas as demais componentes de $\boldsymbol{C_v}(\infty)$ também são iguais entre si e com valor denotado por c_{od}. Então, determinamos c_{md} e c_{od}, de modo que $\boldsymbol{C_v}(\infty)$ seja a solução de (3.60), a qual já sabemos que é única. É fácil ver que o lado esquerdo da Expressão (3.60) é também uma matriz com as mesmas propriedades estruturais de $\boldsymbol{R}_{\kappa\kappa}$, já que é igual a $\eta\,\boldsymbol{R}_{\kappa\kappa}\,J_{\text{min}}$.

Uma forma de $\boldsymbol{C_v}(\infty)$ ter a estrutura proposta seria a de que $\boldsymbol{R}_{\kappa\kappa}\,\boldsymbol{C_v}(\infty)$, $\boldsymbol{C_v}(\infty)\,\boldsymbol{R}_{\kappa\kappa}$ e $\boldsymbol{T}(\infty)$ terem todas a mesma estrutura. É fácil mostrar que as matrizes $\boldsymbol{R}_{\kappa\kappa}\,\boldsymbol{C_v}(\infty)$ e $\boldsymbol{C_v}(\infty)\,\boldsymbol{R}_{\kappa\kappa}$ têm essa estrutura se, e somente se, $\boldsymbol{C_v}(\infty)$ tem esta estrutura. Se este for o caso, uma consequência direta de (3.60) seria que $\boldsymbol{T}(\infty)$ também tem essa estrutura. Usando c_{md} e c_{od}, respectivamente, para componentes na diagonal principal e componentes fora da diagonal principal de $\boldsymbol{C_v}(\infty)$ em (3.30) e (3.31) obtemos para $M > 1$,

$$[\,\boldsymbol{T}(\infty)]_{ij} = \begin{cases} (\mu_1 + (M-1)\,\mu_3)\,c_{\text{md}} + (2(M-1)\,\mu_2 \\ \qquad\qquad + (M-1)(M-2)\,\mu_4)\,c_{\text{od}}, & i = j \\ (2\,\mu_2 + (M-2)\,\mu_4)\,c_{\text{md}} + (4(M-2)\,\mu_4 + 2\,\mu_3 \\ \qquad\qquad + (M-3)(M-2)\,\mu_5)\,c_{\text{od}}, & i \neq j. \end{cases} \tag{3.63}$$

Para $M = 1$, $\boldsymbol{T}(\infty) = \mu_1\,c_{\text{md}}$. Escrevendo, para simplificar a notação, $[\,\boldsymbol{T}(\infty)]_{ii} = t_1\,c_{\text{md}} + t_2\,c_{\text{od}}$ e $[\,\boldsymbol{T}(\infty)]_{ij} = t_3\,c_{\text{md}} + t_4\,c_{\text{od}}$ para todo $i \neq j$, e

resolvendo (3.60) para c_{md} e c_{od} obtemos

$$c_{\text{md}} = \left(\frac{((2r_{\text{md}}-\eta t_1)r_{\text{od}}-(2r_{\text{od}}-\eta t_3)r_{\text{md}})(\eta t_2-2(M-1)r_{\text{od}})}{(\eta t_2-2(M-1)r_{\text{od}})(2r_{\text{od}}-\eta t_3)-(\eta t_4-2(r_{\text{md}}+(M-2)r_{\text{od}}))(2r_{\text{md}}-\eta t_1)} + r_{\text{md}}\right)\left(\frac{\eta J_{\text{min}}}{2r_{\text{md}}-\eta t_1}\right) \tag{3.64}$$

e

$$c_{\text{od}} = \frac{\eta J_{\text{min}}((2r_{\text{md}}-\eta t_1)r_{\text{od}}-(2r_{\text{od}}-\eta t_3)r_{\text{md}})}{(\eta t_2-2(M-1)r_{\text{od}})(2r_{\text{od}}-\eta t_3)-(\eta t_4-2(r_{\text{md}}+(M-2)r_{\text{od}}))(2r_{\text{md}}-\eta t_1)} \tag{3.65}$$

com r_{md} and r_{od} definido em (3.6).

É possível verificar que usando (3.63)–(3.65) no lado esquerdo de (3.60) obtemos o referente lado direito. Também, já sabemos que a solução é única já que $(\boldsymbol{I}-\boldsymbol{G})$ pode ser invertida. Então, retornando a (3.61), obtemos a seguinte resultado desejado

$$\begin{aligned} J_{ms}(\infty) &= J_{\text{min}} + J_{ex}(\infty) \\ &= J_{\text{min}} + M(r_{\text{md}}c_{\text{md}} + (M-1)r_{\text{od}}c_{\text{od}}). \end{aligned} \tag{3.66}$$

No próximo capítulo apresentaremos a análise estatística para os *kernels* gaussiano e polinomial para complementar a análise estocástica do algoritmo adaptativo KLMS.

3.5 Conclusões

Neste capítulo estudamos o comportamento estocástico do algoritmo adaptativo KLMS para entrada gaussiana e não-linearidades que preservem a estacionaridade do sinal de entrada. Este estudo resultou em um modelo

analítico que prevê o comportamento do algoritmo em função dos parâmetros de projeto, estatísticas do sinal de entrada e parâmetros do *kernel*. Em particular, permite estudar as contribuições conjuntas dos parâmetros do *kernel* e do passo para o desempenho do algoritmo em ambos períodos de adaptação dos coeficientes (regime transitório e regime permanente).

Apresentamos um estudo da convergência do algoritmo KLMS quando este opera com *kernels* de valores positivos, de uma maneira diferente da apresentada em (LIU; PRINCIPE; HAYKIN, 2010). Do estudo da convergência derivamos um conjunto de expressões analíticas as quais proveem condições suficientes para a estabilidade. Infelizmente, as expressões derivadas são complexas advindas do estudo de um problema não-linear. Há uma necessidade de calcular os cinco momentos de quarta ordem de funções não-lineares de variáveis aleatórias gaussianas e isto pode parecer complicado. Todavia, eles são simples de serem programados.

O modelo derivado requer a determinação de momentos de quarta ordem de *kernels* dos vetores de entrada. No caso geral, esses momentos devem ser estimados numericamente. No próximo capítulo determinamos os momentos para sinais de entrada gaussianos e para dois dos *kernels* mais empregado na prática; o *kernel* gaussiano e o *kernel* polinomial de ordem dois. Nesses casos obtemos então modelos completamente analíticos.

No Capítulo 5 propomos as diretrizes para o projeto do algoritmo KLMS com *kernels* gaussiano e polinomial gerada a partir dos modelos analíticos derivados.

4 ANÁLISE ESTATÍSTICA DOS KERNELS GAUSSIANO E POLINOMIAL

4.1 Introdução

No Capítulo 3 determinamos as equações de análise do comportamento estocástico do KLMS, assumindo um dicionário estável com comprimento M. Com o objetivo de concluir essa análise para posteriormente aplicá-la a exemplos práticos, vamos deduzir aqui as expressões dos momentos de segunda e quarta ordem para os *kernels* gaussianos e polinomial.

Em todas as deduções apresentadas nas próximas seções consideramos $\boldsymbol{u}(n)$ vetores $(q \times 1)$ gaussianos, de média nula, i.i.d tal que $E\{\boldsymbol{u}(n-i)\,\boldsymbol{u}^{\top}(n-j)\} = \mathbf{0}$ para $i \neq j$. As componentes do vetor $\boldsymbol{u}(n)$ podem ser correlacionadas. Portanto, a matriz $\boldsymbol{R}_{\boldsymbol{uu}}$ pode não ser uma matriz diagonal.

4.2 Análise estatística do kernel gaussiano

Considere o *kernel* gaussiano (SCHÖLKOPF; SMOLA, 2002) dado pela expressão:

$$\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i)) = \exp\left(\frac{-\|\boldsymbol{u}(n) - \boldsymbol{u}(\omega_i)\|^2}{2\xi^2}\right) \tag{4.1}$$

em que ξ é o parâmetro do *kernel*.

O estudo das propriedades estatísticas do *kernel* gaussiano iniciará com o cálculo das componentes da matriz de autocorrelação. Posteriormente, apresentaremos o cálculo dos momentos de quarta ordem, complementando a análise estatística necessária para aplicação no modelo do KLMS.

4.2.1 Momentos de segunda ordem

Dado $\boldsymbol{u}(n)$ com as propriedades estatísticas descritas anteriormente, temos que a matriz de correlação do *kernel* é:

$$[\boldsymbol{R}_{\kappa\kappa}]_{ij} = \begin{cases} E\{\kappa^2(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i))\}, & i = j \\ E\{\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i))\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_j))\}, & i \neq j \end{cases} \tag{4.2}$$

com $1 \leq i, j \leq M$ e $\kappa(., \boldsymbol{u}(\omega_i)) \in \mathcal{D}$.

Vamos introduzir a seguinte notação:

$$\begin{aligned} &\|\boldsymbol{u}(n) - \boldsymbol{u}(\omega_i)\|^2 = \boldsymbol{y}_2^\top \, \boldsymbol{Q}_2 \, \boldsymbol{y}_2 \\ &\|\boldsymbol{u}(n) - \boldsymbol{u}(\omega_i)\|^2 + \|\boldsymbol{u}(n) - \boldsymbol{u}(\omega_j)\|^2 = \boldsymbol{y}_3^\top \, \boldsymbol{Q}_3 \, \boldsymbol{y}_3, \quad i \neq j \end{aligned} \tag{4.3}$$

em que $\|\cdot\|$ é a norma ℓ_2 (ou Euclidiana),

$$\begin{aligned} \boldsymbol{y}_2 &= \left(\boldsymbol{u}^\top(n) \; \boldsymbol{u}^\top(\omega_i)\right)^\top, \\ \boldsymbol{y}_3 &= \left(\boldsymbol{u}^\top(n) \; \boldsymbol{u}^\top(\omega_i) \; \boldsymbol{u}^\top(\omega_j)\right)^\top, \end{aligned} \tag{4.4}$$

$$\boldsymbol{Q}_2 = \begin{pmatrix} \boldsymbol{I} & -\boldsymbol{I} \\ -\boldsymbol{I} & \boldsymbol{I} \end{pmatrix} \quad \text{e} \quad \boldsymbol{Q}_3 = \begin{pmatrix} 2\boldsymbol{I} & -\boldsymbol{I} & -\boldsymbol{I} \\ -\boldsymbol{I} & \boldsymbol{I} & \boldsymbol{O} \\ -\boldsymbol{I} & \boldsymbol{O} & \boldsymbol{I} \end{pmatrix} \tag{4.5}$$

com $\boldsymbol{I}$ sendo a matriz identidade de ordem $(q \times q)$ e $\boldsymbol{O}$ é a matriz nula de ordem $(q \times q)$. Usando os resultados apresentados por Omura (OMURA; KAILATH, 1965, p. 100), sabemos que a função geradora de momentos $\Phi_z(.)$ de uma função quadrática da forma $z = \boldsymbol{y}^\top \boldsymbol{Q} \, \boldsymbol{y}$, em que $\boldsymbol{y}$ é um vetor com distribuição gaussiana e de média nula com matriz covariância $\boldsymbol{R}_y$, é dada por:

$$\Phi_z(s) = E\{e^{sz}\} = \det\{\boldsymbol{I} - 2\,s\,\boldsymbol{Q}\,\boldsymbol{R}_y\}^{-1/2}. \tag{4.6}$$

Fazendo $s = -1/(2\xi^2)$ em (4.6), encontramos que a (i, j)-ésima componente

de $\boldsymbol{R}_{\kappa\kappa}$ é dada por

$$[\boldsymbol{R}_{\kappa\kappa}]_{ij} = \begin{cases} r_{\text{md}} = \det\left\{\boldsymbol{I}_2 + 2\,\boldsymbol{Q}_2\,\boldsymbol{R}_2/\xi^2\right\}^{-1/2}, & i = j \\ r_{\text{od}} = \det\left\{\boldsymbol{I}_3 + \boldsymbol{Q}_3\,\boldsymbol{R}_3/\xi^2\right\}^{-1/2}, & i \neq j \end{cases} \tag{4.7}$$

com $1 \leq i, j \leq M$. Em (4.7), $\boldsymbol{R}_\ell$ é a matriz $(\ell q \times \ell q)$ de correlação vetor $\boldsymbol{y}_\ell$, $\boldsymbol{I}_\ell$ é a matriz identidade de ordem $(\ell q \times \ell q)$ e $\det\{\cdot\}$ denota o determinante da matriz. Finalmente, note que $\boldsymbol{R}_\ell$ é uma matriz bloco-diagonal com $\boldsymbol{R}_{\boldsymbol{uu}}$ ao longo de sua diagonal.

4.2.2 Momentos de quarta ordem

Vimos que o cálculo de (3.24) depende de i, j, ℓ e p, e que isto define cinco diferentes momentos de quarta ordem μ_k, $k = 1, \ldots, 5$. Para calcular esses momentos, no caso do *kernel* gaussiano, vamos usar (OMURA; KAILATH, 1965), com um procedimento análogo ao utilizado para calcular as componentes de $\boldsymbol{R}_{\kappa\kappa}$.

Para $\mu_1 = E\{\kappa^4_{\omega_i}(n)\}$ com $1 \leq i \leq M$.

Usando $\boldsymbol{y}_2$ definido em (4.4), a sua matriz de autocorrelação $\boldsymbol{R}_2$, $\boldsymbol{Q}_2$ dado por (4.5) e a matriz identidade $\boldsymbol{I}_2$, obtemos que:

$$\mu_1 = \left[\det\{\boldsymbol{I}_2 + 4\,\boldsymbol{Q}_2\,\boldsymbol{R}_2/\xi^2\}\right]^{-1/2} \tag{4.8}$$

Para $\mu_2 = E\{\kappa^3_{\omega_i}(n)\,\kappa_{\omega_\ell}(n)\}$ com $1 \leq i, \ell \leq M$ e $i \neq \ell$.

Usando $\boldsymbol{y}_3$ definido em (4.4) e sua matriz de autocorrelação $\boldsymbol{R}_3$. Defina

$$\boldsymbol{Q}_{3'} = \begin{pmatrix} 4\boldsymbol{I} & -3\boldsymbol{I} & -\boldsymbol{I} \\ -3\boldsymbol{I} & 3\boldsymbol{I} & \boldsymbol{O} \\ -\boldsymbol{I} & \boldsymbol{O} & \boldsymbol{I} \end{pmatrix}. \tag{4.9}$$

Assim, usando a matriz identidade $\boldsymbol{I}_3$, obtemos:

$$\mu_2 = [\det\{\boldsymbol{I}_3 + \boldsymbol{Q}_{3'}\,\boldsymbol{R}_3/\xi^2\}]^{-1/2} \tag{4.10}$$

Para $\mu_3 = E\{\kappa^2_{\omega_i}(n)\kappa^2_{\omega_\ell}(n)\}$ com $1 \leq i, \ell \leq M$ e $i \neq \ell$.

Usando $\boldsymbol{y}_3$ definido em (4.4), sua matriz de autocorrelação $\boldsymbol{R}_3$, a matriz $\boldsymbol{Q}_3$ definida em (4.5) e a matriz identidade $\boldsymbol{I}_3$, obtemos:

$$\mu_3 = [\det\{\boldsymbol{I}_3 + 2\,\boldsymbol{Q}_3\,\boldsymbol{R}_3/\xi^2\}]^{-1/2} \tag{4.11}$$

Para $\mu_4 = E\{\kappa^2_{\omega_i}(n)\kappa_{\omega_\ell}(n)\kappa_{\omega_p}(n)\}$ com $1 \leq i, \ell, p \leq M$ e $i \neq p \neq \ell$.

Denote $\boldsymbol{y}_4 = \left(\boldsymbol{u}^\top(n)\,\boldsymbol{u}^\top(\omega_i)\,\boldsymbol{u}^\top(\omega_\ell)\,\boldsymbol{u}^\top(\omega_p)\right)^\top$ e defina:

$$\boldsymbol{Q}_4 = \begin{pmatrix} 4\boldsymbol{I} & -2\boldsymbol{I} & -\boldsymbol{I} & -\boldsymbol{I} \\ -2\boldsymbol{I} & 2\boldsymbol{I} & \boldsymbol{O} & \boldsymbol{O} \\ -\boldsymbol{I} & \boldsymbol{O} & \boldsymbol{I} & \boldsymbol{O} \\ -\boldsymbol{I} & \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{I} \end{pmatrix}. \tag{4.12}$$

Então, obtemos:

$$\mu_4 = [\det\{\boldsymbol{I}_4 + \boldsymbol{Q}_4\,\boldsymbol{R}_4/\xi^2\}]^{-1/2} \tag{4.13}$$

Para $\mu_5 = E\{\kappa_{\omega_i}(n)\kappa_{\omega_\ell}(n)\kappa_{\omega_p}(n)\kappa_{\omega_j}(n)\}$ com $1 \leq i, \ell, p, j \leq M$ e $i \neq j \neq p \neq l$.

Denote $\boldsymbol{y}_5 = \left(\boldsymbol{u}^\top(n)\;\boldsymbol{u}^\top(\omega_i)\;\boldsymbol{u}^\top(\omega_j)\;\boldsymbol{u}^\top(\omega_\ell)\;\boldsymbol{u}^\top(\omega_p)\right)^\top$. Logo:

$$\mu_5 = [\det\{\boldsymbol{I}_5 + \boldsymbol{Q}_5\,\boldsymbol{R}_5/\xi^2\}]^{-1/2} \tag{4.14}$$

em que

$$\boldsymbol{Q}_5 = \begin{pmatrix} 4\boldsymbol{I} & -\boldsymbol{I} & -\boldsymbol{I} & -\boldsymbol{I} & -\boldsymbol{I} \\ -\boldsymbol{I} & \boldsymbol{I} & \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} \\ -\boldsymbol{I} & \boldsymbol{O} & \boldsymbol{I} & \boldsymbol{O} & \boldsymbol{O} \\ -\boldsymbol{I} & \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{I} & \boldsymbol{O} \\ -\boldsymbol{I} & \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{I} \end{pmatrix}. \tag{4.15}$$

Isto finaliza o cálculo das componentes da matriz $\boldsymbol{T}(n)$ para o caso do *kernel* gaussiano.

4.2.3 Relação entre os momentos de quarta ordem do kernel gaussiano

No capítulo anterior usamos a desigualdade de Hölder para estabelecer uma relação entre μ_1 e μ_3. Nesta subseção, veremos que é possível estender esta relação para os outros momentos de quarta ordem, como sendo:

$$\mu_5 \overset{(d)}{\leq} \mu_4 \overset{(c)}{\leq} \mu_3 \overset{(b)}{\leq} \mu_2 \overset{(a)}{\leq} \mu_1. \tag{4.16}$$

A desigualdade (a) pode ser obtida usando a Desigualdade de Hölder, Equação (3.34), com $X = \kappa^3_{\omega_i}(n)$, $Y = \kappa_{\omega_\ell}(n)$, $p = \frac{4}{3}$, e $q = 4$. Seguindo a ordem, para provar a desigualdade (b), inicialmente devemos observar que a desigualdade de Hölder produz [1]

$$E\{X^2Y^2\} \leq \sqrt{E\{|X^3Y|\}\,E\{|XY^3|\}}. \tag{4.17}$$

A desigualdade (b) é obtida diretamente de (4.17) para $X = \kappa_{\omega_i}(n)$ e $Y = \kappa_{\omega_\ell}(n)$.

A desigualdade (c) pode ser provada usando (3.34) com $p = q = 2$, $X = \kappa_{\omega_i}(n)\kappa_{\omega_\ell}(n)$ e $Y = \kappa_{\omega_i}(n)\kappa_{\omega_p}(n)$,

$$\begin{aligned}\mu_4 = E\{\kappa^2_{\omega_i}(n)\kappa_{\omega_\ell}(n)\kappa_{\omega_p}(n)\} &\leq E\{\kappa^2_{\omega_i}(n)\kappa^2_{\omega_\ell}(n)\}^{1/2} E\{\kappa^2_{\omega_i}(n)\kappa^2_{\omega_p}(n)\}^{1/2} \\ &= E\{\kappa^2_{\omega_i}(n)\kappa^2_{\omega_\ell}(n)\} = \mu_3\end{aligned}$$

onde a igualdade é obtida diretamente da estacionaridade de $\kappa_\omega(n)$. Agora, para $p = q = 2$, $X = \kappa_{\omega_i}(n)[\kappa_{\omega_p}(n)\kappa_{\omega_\ell}(n)]^{1/2}$ e $Y = \kappa_{\omega_j}(n)[\kappa_{\omega_p}(n)\kappa_{\omega_\ell}(n)]^{1/2}$, (3.34) obtemos

$$\begin{aligned}\mu_5 &= E\{\kappa_{\omega_i}(n)\kappa_{\omega_\ell}(n)\kappa_{\omega_p}(n)\kappa_{\omega_j}(n)\} \\ &\leq E\{\kappa^2_{\omega_i}(n)\kappa_{\omega_\ell}(n)\kappa_{\omega_p}(n)\}^{1/2} E\{\kappa^2_{\omega_j}(n)\kappa_{\omega_\ell}(n)\kappa_{\omega_p}(n)\}^{1/2} \\ &= E\{\kappa^2_{\omega_i}(n)\kappa_{\omega_\ell}(n)\kappa_{\omega_p}(n)\} = \mu_4\end{aligned}$$

[1] Substituindo X por $X^{3/2}Y^{1/2}$ e Y por $X^{1/2}Y^{3/2}$ em (3.34) com $p = q = 2$.

em que a igualdade é proveniente da estacionaridade de $\kappa_\omega(n)$. Esta última relação prova a desigualdade (d) e completa a prova de (4.16).

Estas desigualdades podem ser úteis na determinação do passo máximo de convergência, reduzindo as restrições no caso do KLMS baseado no *kernel* gaussiano[2].

4.3 Análise estatística do kernel polinomial

Nesta seção, abordaremos as propriedade estatísticas do *kernel* polinomial. Para isso, consideraremos $\boldsymbol{u}(n) \in \mathbb{R}^q$ com as propriedades estatísticas descritas no início desse capítulo e o *kernel* polinomial dado pela expressão (SCHÖLKOPF; SMOLA, 2002):

$$\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i)) = [a + \boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_i)]^\beta \tag{4.18}$$

com $a \in \mathbb{R}^+$ e $\beta \in \mathbb{N}^*$, sendo os parâmetros ajustáveis do *kernel* polinomial.

Inicialmente, apresentaremos aqui o cálculo do valor esperado, momento de primeira ordem, e posteriormente apresentaremos a expressão para o cálculo dos momentos de ordem superior como generalização do momento de primeira ordem.

4.3.1 Momento de primeira ordem

O valor esperado do *kernel* polinomial pode ser obtido fazendo,

$$\begin{aligned} E\{\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i))\} &= E\{[a + \boldsymbol{u}(n)\boldsymbol{u}(\omega_i)]^\beta\} \\ &= \sum_{k=0}^{\beta} \binom{\beta}{\beta-k} a^k E\left\{(\boldsymbol{u}^T(n)\boldsymbol{u}(\omega_i))^{\beta-k}\right\} \end{aligned} \tag{4.19}$$

[2]Note que essas desigualdades seguiram do fato de podermos eliminar os módulos da desigualdade de Hölder, já o que o *kernel* gaussiano é sempre positivo. Sendo assim, essas relações não podem ser obtidas para o *kernel* polinomial, já que a eliminação dos módulos não é sempre possível.

em que $\binom{\beta}{\beta-k} = \frac{\beta!}{k!(\beta-k)!}$. Note que o valor esperado de (4.19) depende da potência de $(\beta - k)$ da seguinte maneira:

$$E\left\{\left(\boldsymbol{u}^{\top}(n)\boldsymbol{u}(\omega_q)\right)^{\beta-k}\right\} = \begin{cases} 0, & \text{se } (\beta-k) \text{ é ímpar} \\ E\left\{\left(\sum_{j=1}^{q}[\boldsymbol{u}(n)]_j[\boldsymbol{u}(\omega_i)]_j\right)^{\beta-k}\right\} & \text{se } (\beta-k) \text{ é par.} \end{cases} \tag{4.20}$$

A igualdade a zero pode ser verificada da seguinte maneira:

Vamos definir $S(\theta) = E\left\{\left(\sum_{j=1}^{q}[\boldsymbol{u}(n)]_j[\boldsymbol{u}(\omega_i)]_j\right)^{\theta}\right\}$, assim

$$\begin{aligned} S(\theta) &= \sum_{k_1+k_2+\cdots+k_q=\theta} \binom{\theta}{k_1,k_2,\ldots,k_q} E\Big\{([\boldsymbol{u}(n)]_1[\boldsymbol{u}_k(\omega_i)]_1)^{k_1}([\boldsymbol{u}(n)]_2[\boldsymbol{u}_k(\omega_i)]_2)^{k_2} \\ &\quad \times \ldots([\boldsymbol{u}(n)]_q[\boldsymbol{u}_k(\omega_i)]_q)^{k_q}\Big\} \\ &= \sum_{k_1+k_2+\cdots+k_q=\theta} \binom{\theta}{k_1,k_2,\ldots,k_q} E\Big\{[\boldsymbol{u}(n)]_1^{k_1}\ldots[\boldsymbol{u}(n)]_q^{k_q}\Big\} \\ &\quad \times E\Big\{[\boldsymbol{u}(\omega_i)]_1^{k_1}\ldots[\boldsymbol{u}(\omega_i)]_q^{k_q}\Big\} \\ &= \sum_{k_1+k_2+\cdots+k_q=\theta} \binom{\theta}{k_1,k_2,\ldots,k_q} \left(E\Big\{[\boldsymbol{u}(n)]_1^{k_1}\ldots[\boldsymbol{u}(n)]_q^{k_q}\Big\}\right)^2 \end{aligned} \tag{4.21}$$

em que $\binom{\theta}{k_1,k_2,\ldots,k_q} = \frac{\theta!}{k_1!k_2!\ldots k_q!}$.

Note que, quando θ for um número ímpar, os valores esperados de (4.21) correspondem a momentos de ordem ímpar de variáveis aleatórias gaussianas. Portanto, podemos afirmar que dados $\boldsymbol{u}(n)$ e $\boldsymbol{u}(\omega_i)$ vetores de variáveis aleatórias gaussianas i.i.d, o valor esperado $E\left\{\left([\boldsymbol{u}(n)]^{\top}[\boldsymbol{u}(\omega_i)]\right)^{\theta}\right\}$ é nulo sempre que θ for ímpar.

Isto conclui o cálculo do valor esperado do *kernel* polinomial. Na próxima subseção, com um procedimento análogo, obtemos uma expressão para o cálculo dos momentos de ordem superior.

4.3.2 Momentos de ordens superiores

Nesta subseção vamos apresentar uma equação geral que pode ser utilizada para calcular um momento de qualquer ordem do *kernel* polinomial. No Apêndice G apresentamos um exemplo em que aplicamos esse procedimento para obter os momentos de segunda e quarta ordens para um *kernel* polinomial do segundo grau.

Vamos definir a expressão geral para o cálculo do momento como,

$$E\left\{\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_{i_1}))^{\theta_1}\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_{i_2})^{\theta_2}\ldots\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_{i_m})^{\theta_m}\right\} \tag{4.22}$$

em que $\theta_\ell \in \mathbb{N}$ e $i_j \neq i_k$ para todo $j \neq k$. Note que a soma $\sum_{\ell=1}^{m}\theta_\ell$ definirá a ordem do momento que será calculado.

Agora, fazendo $\theta'_\ell = \beta\theta_\ell$ e condicionando o valor esperado em $\boldsymbol{u}(n)$, obtemos:

$$\begin{aligned} E\left\{\prod_{\ell=1}^{m}\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_{i_\ell}))^{\theta_\ell}\right\} &= E\left\{\prod_{\ell=1}^{m}[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{i_\ell})]^{\theta'_\ell}\right\} \\ &= E\left\{\prod_{\ell=1}^{m}E\left\{[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{i_\ell})]^{\theta'_\ell}|\boldsymbol{u}(n)\right\}\right\} \end{aligned} \tag{4.23}$$

Note que cada parcela de valor esperado condicionado pode ser reescrita como

$$\begin{aligned} &E\left\{[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{i_\ell})]^{\theta'_\ell}|\boldsymbol{u}(n)\right\} = \\ &\qquad\sum_{k^{(\ell)}=0}^{\theta'_\ell}\mathsf{C}_{\theta'_\ell-k^{(\ell)}}^{\theta'_\ell}a^{k^{(\ell)}}E\left\{[\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{i_\ell})]^{\theta'_\ell-k^{(\ell)}}|\boldsymbol{u}(n)\right\} \end{aligned} \tag{4.24}$$

em que (ℓ) representa um índice e não uma potência. Calculando o valor

esperado condicional, obtemos:

$$E\left\{[\boldsymbol{u}^{\top}(n)\boldsymbol{u}(\omega_{i_\ell})]^{\theta'_\ell-k^{(\ell)}}|\boldsymbol{u}(n)\right\} = \sum_{k_1^{(\ell)}+k_2^{(\ell)}+\cdots+k_q^{(\ell)}=\theta_\ell-k^{(\ell)}} \mathsf{C}^{\theta_\ell-k^{(\ell)}}_{k_1^{(\ell)},k_2^{(\ell)},\ldots,k_q^{(\ell)}} E\left\{\prod_{j=1}^{q}[\boldsymbol{u}(\omega_{i_\ell})]_j^{k_j^{(\ell)}}\right\}\prod_{p=1}^{q}[\boldsymbol{u}(n)]_p^{k_p^{(\ell)}}. \tag{4.25}$$

Assim, podemos escrever a expressão geral do momento de ordem superior (4.22) como sendo:

$$\begin{aligned} E\left\{\prod_{\ell=1}^{m}\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_{i_\ell}))^{\theta_\ell}\right\} = \\ \sum_{k^{(1)}=0}^{\theta'_1}\cdots\sum_{k^{(m)}=0}^{\theta'_m}\mathsf{C}^{\theta'_1}_{\theta'_1-k^{(1)}}\cdots\mathsf{C}^{\theta'_m}_{\theta'_m-k^{(m)}}\, a^{k^{(1)}+\ldots+k^{(m)}} \sum_{k_1^{(1)}+\cdots+k_q^{(1)}=\theta_1-k^{(1)}}\cdots \\ \times\sum_{k_1^{(m)}+\cdots+k_q^{(m)}=\theta_m-k^{(m)}}\mathsf{C}^{\theta_1-k^{(1)}}_{k_1^{(1)},\ldots,k_q^{(1)}}\cdots\mathsf{C}^{\theta_m-k^{(m)}}_{k_1^{(m)},\ldots,k_q^{(m)}} E\left\{\prod_{j=1}^{q}[\boldsymbol{u}(\omega_{i_1})]_j^{k_j^{(1)}}\right\} \\ \times E\left\{\prod_{j=1}^{q}[\boldsymbol{u}(\omega_{i_m})]_j^{k_j^{(m)}}\right\}E\left\{\prod_{p=1}^{q}[\boldsymbol{u}(n)]_p^{k_p^{(1)}+\cdots+k_p^{(m)}}\right\} \end{aligned} \tag{4.26}$$

Esta expressão pode ser utilizada para calcular o momento de qualquer ordem, basta variar o valor de θ_ℓ. Considerando que $\boldsymbol{u}(n)$ pode ter componentes correlacionadas e que as componentes estão elevadas a determinadas potências, o cálculo do valor esperado de (4.26) pode parecer algo complicado de ser obtido. Porém, como $\boldsymbol{u}(n)$ possui distribuição gaussiana e, portanto, α-estável[3] existe uma maneira simples de calcular estes valores esperados, que

[3] Quando qualquer combinação linear finita de variáveis aleatórias de mesma distribuição gera uma variável aleatória de mesma distribuição das variáveis aleatórias que a gerou dizemos que essas variáveis aleatórias tem distribuição α-estável ou de Levy. Este tipo de distribuição invariante a sistemas lineares são muito úteis em processamento de sinais (MANOLAKIS; INGLE;

é feito na sequência.

Momentos cruzados de variáveis aleatórias gaussianas

Seja, então, $\boldsymbol{u} = [u_1, u_2, \ldots, u_q]^\top$ um vetor de variáveis aleatórias gaussianas (ou α-estável) com $E\{\boldsymbol{u}\} = \mathbf{0}$ e autocorrelação $\boldsymbol{R_{uu}}$ como já descrito no início deste capítulo. Como as componentes de $\boldsymbol{u}$ podem não ser descorrelacionadas, vamos reescrever cada componente u_i de $\boldsymbol{u}$ como combinação de variáveis aleatórias gaussianas descorrelacionadas e_k, isto é,

$$u_i = \sum_{k=1}^{i} b_k^{(i)} e_k \tag{4.27}$$

em que $E\{e_k\} = 0$ para todo k e

$$E\{e_\ell e_p\} = \begin{cases} 0, \text{ se } \ell \neq p \\ 1, \text{ se } \ell = p \end{cases} \tag{4.28}$$

Desta maneira, só precisamos calcular cada $b_k^{(i)}$, para obter os valores esperados de (4.26), quando $\boldsymbol{u}(n)$ tiver distribuição gaussiana. Isto pode ser feito sistematicamente usando os momentos de segunda ordem, as componentes de $\boldsymbol{R_{uu}}$, da seguinte maneira:

1. $[\boldsymbol{R_{uu}}]_{11} = E\left\{u_1^2\right\} = \left(b_1^{(1)}\right)^2 \Rightarrow b_1^{(1)} = \sqrt{[\boldsymbol{R_{uu}}]_{11}}$

2. $[\boldsymbol{R_{uu}}]_{12} = E\{u_1 u_2\} \Rightarrow [\boldsymbol{R_{uu}}]_{12} = b_1^{(1)} b_1^{(2)} \Rightarrow b_1^{(2)} = \frac{[\boldsymbol{R_{uu}}]_{12}}{\sqrt{[\boldsymbol{R_{uu}}]_{11}}}$

   $[\boldsymbol{R_{uu}}]_{22} = E\left\{u_2^2\right\} \Rightarrow [\boldsymbol{R_{uu}}]_{22} = \left(b_1^{(2)}\right)^2 + \left(b_2^{(2)}\right)^2 \Rightarrow b_2^{(2)} = \sqrt{[\boldsymbol{R_{uu}}]_{11} - \left(b_1^{(2)}\right)^2}$

3. $[\boldsymbol{R_{uu}}]_{13} = E\{u_1 u_3\} \Rightarrow [\boldsymbol{R_{uu}}]_{13} = b_1^{(1)} b_1^{(3)} \Rightarrow b_1^{(3)} = \frac{[\boldsymbol{R_{uu}}]_{13}}{\sqrt{[\boldsymbol{R_{uu}}]_{11}}}$

   $[\boldsymbol{R_{uu}}]_{23} = E\{u_2 u_3\} \Rightarrow [\boldsymbol{R_{uu}}]_{23} = b_1^{(2)} b_1^{(3)} + b_2^{(2)} b_2^{(3)} \Rightarrow b_2^{(3)} = \frac{[\boldsymbol{R_{uu}}]_{23} - b_1^{(2)} b_1^{(3)}}{b_2^{(2)}}$

KOGON, 2005).

$$[\boldsymbol{R}_{\boldsymbol{uu}}]_{33} = E\left\{u_3^2\right\} \Rightarrow b_3^{(3)} = \sqrt{[\boldsymbol{R}_{\boldsymbol{uu}}]_{33} - \left(b_2^{(3)}\right)^2 + \left(b_1^{(3)}\right)^2}$$

Assim, usando as componentes de $\boldsymbol{R}_{\boldsymbol{uu}}$ procedemos de maneira análoga até que todos os $b_k^{(i)}$ estejam devidamente calculados.

No Apêndice G usamos esta expressão para calcular os momentos de segunda e quarta ordem para um *kernel* polinomial do segundo grau, os quais são importantes para gerar alguns dos resultados apresentados no próximo capítulo.

4.4 Análise de convergência do KLMS para o caso geral

No capítulo anterior deduzimos as condições de convergência para o KLMS quando esse opera com *kernels* positivos. Essa classe de *kernels* inclui alguns dos *kernels* mais utilizados em processamento de sinais, mas não pode ser considerada o caso geral.

4.4.1 Estudo do sinal dos momentos de segunda e quarta ordem do kernel

Iniciamos o complemento da análise de estabilidade, caso em que o *kernel* pode não ser positivo, com o estudo do sinal dos momentos de segunda e quarta ordem de $\kappa_\omega(n)$.

Definido $\boldsymbol{R}_{\kappa\kappa}$ como a matriz de autocorrelação de $\kappa_\omega(n)$, já sabemos que a componente r_{md} é sempre positiva. Desde que o *kernel* utilizado não seja um *kernel* de valores positivos (Ex.: *kernel* gaussiano), verificar o sinal de r_{od} pode não parecer tão trivial. Por este motivo, vamos partir da definição da componente r_{od} que é:

$$E\{\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i))\,\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_j))\}, \text{ com } i \neq j$$

Note que r_{od} pode ser reescrita como:

$$r_{\text{od}} = E\{E\{\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i))|\boldsymbol{u}(n)\}\,E\{\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_j))|\boldsymbol{u}(n)\}\}, \qquad (4.29)$$

em que $E\{\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i))|\boldsymbol{u}(n)\}$ representa o valor esperado condicionado a variável aleatória $\boldsymbol{u}(n)$. Como $\boldsymbol{u}(n)$ é estacionário, então

$$E\{\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i))|\boldsymbol{u}(n)\} = E\{\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_j))|\boldsymbol{u}(n)\}, \ \ \forall 1 \leq i, j \leq M.$$

Assim, concluímos que,

$$r_{\text{od}} = E\{E\{\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_i))|\boldsymbol{u}(n)\}^2\} \geq 0 \ \forall 1 \leq i \leq M. \tag{4.30}$$

Esta relação é válida para qualquer *kernel*, pois usamos apenas a definição do momento de segunda ordem r_{od}, sem considerar um *kernel* específico.

Vamos agora verificar o sinal dos momentos de quarta ordem. Para μ_1 temos:

$$\mu_1 = E\{\kappa^4_{\omega_i}(n)\} \geq 0, \ \ \forall 1 \leq i \leq M. \tag{4.31}$$

Novamente, a desigualdade segue diretamente da definição. Portanto, essa relação também é válida para qualquer função *kernel*.

Na sequência, para μ_2 temos:

$$\mu_2 = E\{\kappa^3_{\omega_i}(n)\,\kappa_{\omega_\ell}(n)\}. \tag{4.32}$$

Note que para *kernel* com valores positivos $\mu_2 \geq 0$, mas isto nem sempre ocorre. Neste caso um procedimento que pode ser adotado é avaliar μ_2 em casos específicos, isto é, analisar a expressão para um *kernel* específico. Como neste trabalho optamos pelo uso do *kernel* gaussiano e *kernel* polinomial, vamos avaliar μ_2 apenas para o caso em que o *kernel* usado é o polinomial, uma vez que é trivial mostrar que no caso do *kernel* gaussiano $\mu_2 \geq 0$. Considere,

então, o momento de quarta ordem μ_2 do *kernel* polinomial dado por:

$$\begin{aligned}
\mu_2 &= E\{[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_i)]^{3\beta}[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_\ell)]^{\beta}\} \\
&= E\{E\{[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_i)]^{3\beta}|\boldsymbol{u}(n)\}\, E\{[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_\ell)]^{\beta}|\boldsymbol{u}(n)\}\} \\
&= E\Bigg\{\sum_{k=0}^{3\beta} \mathsf{C}_{3\beta-k}^{3\beta} a^{3\beta-k} E\{[\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_i)]^{k}|\boldsymbol{u}(n)\} \\
&\times \sum_{k'=0}^{\beta} \mathsf{C}_{\beta-k'}^{\beta} a^{\beta-k'} E\{[\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_\ell)]^{k'}|\boldsymbol{u}(n)\}\Bigg\} \\
&= \sum_{k=0}^{3\beta}\sum_{k'=0}^{\beta} \mathsf{C}_{3\beta-k}^{3\beta} \mathsf{C}_{\beta-k'}^{\beta} a^{(4\beta-k'-k)} E\{E\{[\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_i)]^{k}|\boldsymbol{u}(n)\} \\
&\times E\{[\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_\ell)]^{k'}|\boldsymbol{u}(n)\}\}.
\end{aligned} \tag{4.33}$$

Note que, por um lado, se k e k' representam potências pares os valores esperados $E\{[\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_i)]^{k}|\boldsymbol{u}(n)\}$ e $E\{[\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_\ell)]^{k'}|\boldsymbol{u}(n)\}$ são ambos não negativos. Por outro lado, se k e k' são valores ímpares, $E\{[\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_i)]^{k}|\boldsymbol{u}(n)\}$ e $E\{[\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_\ell)]^{k'}|\boldsymbol{u}(n)\}$ são nulos. Portanto, para o caso em que o *kernel* utilizado é o polinomial temos a relação:

$$\mu_2 = E\{\kappa_{\omega_i}^3(n)\kappa_{\omega_\ell}(n)\} \geq 0, \;\; \forall 1 \leq i \leq M. \tag{4.34}$$

Seguindo a ordem, é fácil ver que μ_3 é positivo, pois

$$\mu_3 = E\{\kappa_{\omega_i}^2(n)\kappa_{\omega_\ell}^2(n)\} \geq 0, \;\; \forall 1 \leq i,\ell \leq M. \tag{4.35}$$

Para verificar o sinal de μ_4, podemos fazer:

$$\begin{aligned}
\mu_4 &= E\{E\{\kappa_{\omega_i}^2(n)|\boldsymbol{u}(n)\}\, E\{\kappa_{\omega_\ell}(n)|\boldsymbol{u}(n)\}\, E\{\kappa_{\omega_p}(n)|\boldsymbol{u}(n)\}\} \\
&= E\{E\{\kappa_{\omega_i}^2(n)|\boldsymbol{u}(n)\}\, E\{\kappa_{\omega_\ell}(n)|\boldsymbol{u}(n)\}^2\} \geq 0, \;\; \forall 1 \leq i,\ell \leq M.
\end{aligned} \tag{4.36}$$

A segunda igualdade segue da estacionaridade de $\boldsymbol{u}(n)$.

Para concluir o estudo do sinal, considere μ_5, que é:

$$\begin{aligned}\mu_5 &= E\{E\{\kappa_{\omega_i}(n)|\boldsymbol{u}(n)\}\,E\{\kappa_{\omega_\ell}(n)|\boldsymbol{u}(n)\}\,E\{\kappa_{\omega_p}(n)|\boldsymbol{u}(n)\}\,E\{\kappa_{\omega_j}(n)|\boldsymbol{u}(n)\}\}\\ &= E\{E\{\kappa_{\omega_i}(n)|\boldsymbol{u}(n)\}^4\} \geq 0, \quad \forall 1 \leq i,\ell,p,j \leq M.\end{aligned} \tag{4.37}$$

Do presente estudo, podemos concluir que a análise da estabilidade apresentada na Subseção 3.4.1 do Capítulo 3 pode ser estendida para o *kernel* polinomial independentemente do grau utilizado. Na próxima subseção, apresentaremos um procedimento que substitui um dos conjuntos de restrições obtidos a partir do fato de $\mu_2 \geq 0$. Esta nova restrição só deve ser utilizada na análise da estabilidade do algoritmo KLMS quando $\mu_2 < 0$.

4.4.2 Condições para convergência quando $\mu_2 < 0$

O conjunto de restrições que apresentamos nesta subseção deverá substituir aquelas apresentadas em (3.49) da Subseção 3.4.1. Este procedimento deve ser adotado sempre que $\mu_2 < 0$. Vamos retomar as desigualdades dadas pelas Expressões (3.45a) e (3.45b), que são:

$$2(M-1)(\eta\mu_2 - r_{\text{od}}) < 2r_{\text{md}} - \eta\mu_1 - (M-1)\eta\mu_3 - (M-1)(M-2)\eta\mu_4 \tag{4.38a}$$

$$2(M-1)(\eta\mu_2 - r_{\text{od}}) > -2r_{\text{md}} + \eta\mu_1 + (M-1)\eta\mu_3 + (M-1)(M-2)\eta\mu_4. \tag{4.38b}$$

Essas inequações podem ser reescritas como:

$$\eta(\mu_1 + 2(M-1)\mu_2 + (M-1)\mu_3 + (M-1)(M-2)\mu_4) < 2r_{\text{md}} + 2(M-1)r_{\text{od}} \tag{4.39a}$$

$$-\eta(\mu_1 - 2(M-1)\mu_2 + (M-1)\mu_3 + (M-1)(M-2)\mu_4) > -2r_{\text{md}} + 2(M-1)r_{\text{od}}. \tag{4.39b}$$

Considerando,

$$\theta_0 = \mu_1 + 2(M-1)\mu_2 + (M-1)\mu_3 + (M-1)(M-2)\mu_4, \tag{4.40}$$

podemos resolver (4.39a) usando $\eta_1 = \frac{2r_{\text{md}} + 2(M-1)r_{\text{od}}}{\theta_0}$. Note que, como o numerador é sempre positivo (pois $r_{\text{od}} > 0$), é necessário avaliar apenas o sinal do denominador. Assim, o conjunto de restrições para (4.39a) é:

$$\begin{cases} 0 \leq \eta < \eta_1, & \text{quando } \theta_0 \geq 0 \\ 0 \leq \eta, & \text{quando } \theta_0 < 0 \end{cases} \tag{4.41}$$

Em contrapartida, a Equação (4.39b) gera uma restrição na forma $\theta_2\eta < \theta_1$, em que

$$\theta_1 = 2r_{\text{md}} - 2(M-1)r_{\text{od}} \tag{4.42}$$

e

$$\theta_2 = \mu_1 - 2(M-1)\mu_2 + (M-1)\mu_3 + (M-1)(M-2)\mu_4. \tag{4.43}$$

Note que para $\mu_2 < 0$, temos que $\theta_2 > 0$, o que produz um novo conjunto de restrições obtidas a partir de (4.39b):

$$\begin{cases} 0 \leq \eta < \eta_2, & \text{quando } \theta_1 \geq 0 \\ 0 \leq \eta, & \text{quando } \theta_1 < 0 \end{cases} \tag{4.44}$$

Fazendo a interseção entre as Restrições (4.41) e (4.44), obtemos as restrições apresentadas na Tabela 4.1. Assim, a combinação das Restrições (4.39a) e

Tabela 4.1: Restrições em η obtidas a partir de (4.41) e (4.44)

Caso	Sinal de θ_0 (4.40)	Sinal de θ_1 (3.47)	Restrição	Observação
(i)	+	+	$0 \leq \eta < \min\{\eta_1, \eta_2\}$	
(ii)	+	−	sem solução	Não existe $\eta > 0$
(iii)	−	+	$0 < \eta < \eta_2$	
(iv)	−	−	sem solução	Não existe $\eta > 0$

(4.39b) é:

$$\begin{cases} \eta < \min\{\eta_1, \eta_2\}, & \text{para o caso } (i) \text{ na Tabela 4.1} \\ \eta < \eta_2, & \text{para o caso } (iii) \text{ na Tabela 4.1.} \end{cases} \tag{4.45}$$

O conjunto de restrições em (4.45) substitui àquelas apresentadas em (3.49) considerando $\mu_2 \geq 0$. O restante da análise segue de maneira análoga ao apresentado na Subseção 3.4.1, pois as restrições (3.56) não são influenciadas pelo sinal de μ_2. Da mesma maneira com que foi feito na Subseção 3.4.1, se não for possível obter η que satisfaça ao conjunto de restrições encontradas, devemos recorrer a uma solução numérica para obter η_{max}.

4.5 Conclusões

Neste capítulo foram apresentadas as expressões dos momentos estatísticos dos *kernels* gaussiano e polinomial. O conhecimento destes momentos é de fundamental importância para conclusão da análise do algoritmo adaptativo KLMS, pois toda a teoria obtida no Capítulo 3 está baseada no conhecimento destas estatísticas.

O que justifica a presente análise estatística é a transformação não-linear gerada pelo *kernel* no sinal de entrada $\boldsymbol{u}(n)$, que é $\kappa_\omega(n)$. Na análise convencional do LMS considerando que $\boldsymbol{u}(n)$ é i.i.d gaussiano, todos os momentos eram facilmente obtidos pelo teorema da Fatoração de Momentos de

variáveis aleatórias gaussianas (MILLER, 1963), o que não ocorre na análise do KLMS. Portanto, para a presente análise, devemos calcular os momentos de segunda e quarta ordem especificamente para cada *kernel* que será utilizado no KLMS. Mais ainda, no caso do *kernel* polinomial, devemos fazê-lo para cada grau, pois uma modificação do grau do polinômio modifica também a transformação gerada pelo *kernel*.

Apesar de apresentar expressões analíticas que permitam obter os momentos de segunda e quarta ordem para um polinômio de qualquer grau, fizemos no Apêndice G os cálculos para obter esses momentos considerando um *kernel* polinomial do segundo grau, pois, este *kernel* será avaliado no próximo capítulo.

5 DIRETRIZES DE PROJETO, SIMULAÇÕES E RESULTADOS

5.1 Introdução

No Capítulo 3, vimos que a superfície de desempenho é alterada pelo *kernel* e seus parâmetros (3.8). Isso ocorre porque os parâmetros do *kernel* alteram as estatísticas do sinal de entrada. Vimos também a necessidade de estudar o comportamento do MSE mínimo pelo fato deste estar diretamente relacionado ao cálculo do regime permanente do MSE (3.66) e do MSE em excesso (3.62). Na análise convencional do LMS, o MSE mínimo coincidia com a potência do ruído adicionado à saída do sistema. No caso do algoritmo KLMS isto não ocorre, porque o Teorema da Representação de Riez não garante que esta representação, de um funcional não-linear no RKHS, seja exata (VERT; TSUDA; SCHÖLKOPF, 2004).

No Apêndice F apresentamos alguns resultados da avaliação do MSE mínimo de uma função não-linear partindo do conhecimento prévio do sinal não-linear a ser estimado. Nesse caso, é possível fazer os cálculos analíticos das estatísticas (autocorrelação do sinal desejado e a correlação cruzada entre a entrada e a saída do sistema). É claro que em aplicações práticas, a função que caracteriza a não-linearidade do sistema não é conhecida. Porém, usando pré-observações (realizadas antes de inciarmos a adaptação dos coeficientes) podemos estimar as estatísticas necessárias e obter o MSE mínimo. Usamos esse procedimento para avaliar o comportamento do algoritmo adaptativo KLMS em três exemplos que apresentaremos neste capítulo.

Neste capítulo, organizamos as equações obtidas a partir da análise estocástica do KLMS, Capítulos 3 e 4, em uma sequência de passos (diretrizes) para elaboração de um projeto de filtro usando o algoritmo adaptativo KLMS. Nosso objetivo final é optar pelo uso de um determinado *kernel* com seu respectivo parâmetro, considerando o conhecimento prévio da qualidade

da estimativa e a demanda computacional[1] de uma dada aplicação. Além dos exemplos apresentados na Seção 5.3, outros exemplos de estimação não-linear usando a presente análise já foram apresentados à comunidade científica, (PARREIRA et al., 2011b) e (PARREIRA et al., 2011a). Os resultados promissores obtidos nessas publicações para o *kernel* gaussiano motivou a extensão dos resultados para o *kernel* polinomial.

5.2 Diretrizes de projeto do KLMS usando kernel gaussiano e polinomial

Nesta seção usamos as equações de análise do algoritmo KLMS para estabelecer as diretrizes de projeto. Sem limitar a generalidade, o processo de esparsificação baseado na função coerência (2.13) é considerado a seguir para projetar os dicionários. No entanto, fica implícito que qualquer técnica para selecionar funções *kernel* $\kappa(\cdot, \boldsymbol{u}(\omega_j))$ pode ser utilizada.

Suponha que um objetivo de projeto seja obter um MSE inferior a um valor específico $J_{\max}$. O seguinte procedimento pode ser aplicado.

1. Defina um limiar para coerência, ε_0, e o conjunto de parâmetros do *kernel* o qual será testado[2] . Vamos denotar por $\mathcal{P}$ o conjunto de parâmetros do *kernel* que será avaliado, com $\mathcal{P}_{GK}$ sendo um conjunto de parâmetros ξ do *kernel* gaussiano e $\mathcal{P}_{PK}$ um conjunto de parâmetros (a, β) no caso do *kernel* polinomial[3]. Para os exemplos que apresentaremos a seguir, ε_0 e o conjunto de parâmetros $\mathcal{P}$ foram escolhidos para produzirem um dicionário de dimensão M razoável.

   O valor M é determinado, para cada elemento de $\mathcal{P}$ associado a um dado ε_0, por simulação usando um sinal de entrada $\boldsymbol{u}(n)$ até o seu comprimento estabilizar. A simulação é repetida algumas vezes, então um

[1] Quantidade de iterações necessárias para convergência.

[2] É importante deixar claro que M e ε_0 são parâmetros diretamente proporcionais. Assim aumentando o valor de ε_0 aumenta também os valores de M.

[3] Nos exemplos apresentados foi usado $\beta = 2$.

valor de M é associado a todo par $(\mathcal{P}_i, \varepsilon_0)$, em que $\mathcal{P}_i$ é um elemento de $\mathcal{P}$. O valor de M é determinado pela média dos comprimentos dos dicionários estáveis sobre todas as realizações.

2. Usando a entrada do sistema $\boldsymbol{u}(n)$, obtenha a saída desejada $d(n)$ e estime $E\{d^2(n)\}$ e $\boldsymbol{p}_{\kappa d}$ em várias realizações. Em um cenário prático $d(n)$ deve ser medido na saída do sistema desconhecido.

3. Pela Equação (3.8), estime o mínimo MSE, $J_{\min}$ para cada elemento do conjunto de parâmetros $(\mathcal{P}_i, M)$. Se nenhum dos parâmetros testado gerar $J_{\min} < J_{\max}$, retorne ao passo 1 e defina novos parâmetros para teste.

4. Determine $\eta_{\max}$ usando as regras (3.49) e (3.56) para cada valor de $\mathcal{P}_i$ e escolha o maior valor de η tal que $\eta < \eta_{\max}$ e $J_{ms}(\infty) < J_{\max}$ (note que $J_{ms}(\infty)$ é calculado usando (3.66)). Uma sugestão para busca de η é usar um algoritmo de bipartição considerando $\eta_0 = 0,5\eta_{\max}$.

5. Dado M, $\boldsymbol{R}_{\kappa\kappa}$, $\boldsymbol{p}_{\kappa d}$, $J_{\min}$ e η determine $\boldsymbol{G}$ usando (3.40) e então $\boldsymbol{c}_{\boldsymbol{v}}(n)$ a partir de (3.41).

6. Determine $\boldsymbol{C}_{\boldsymbol{v}}(\infty)$ a partir das Equações (3.64)–(3.65), e $J_{ms}(\infty)$ a partir da expressão (3.66).

7. Escolha ϵ no critério (3.59) e encontre n_ϵ a partir das simulações usando os parâmetros determinados nos passos 1 a 6.

Repita os passos 1 a 7 para cada *kernel* (e seus parâmetros) que será avaliado e escolha aquele com o menor valor de n_ϵ.

A Figura 5.1 apresenta a sequência de passos descrita anteriormente.

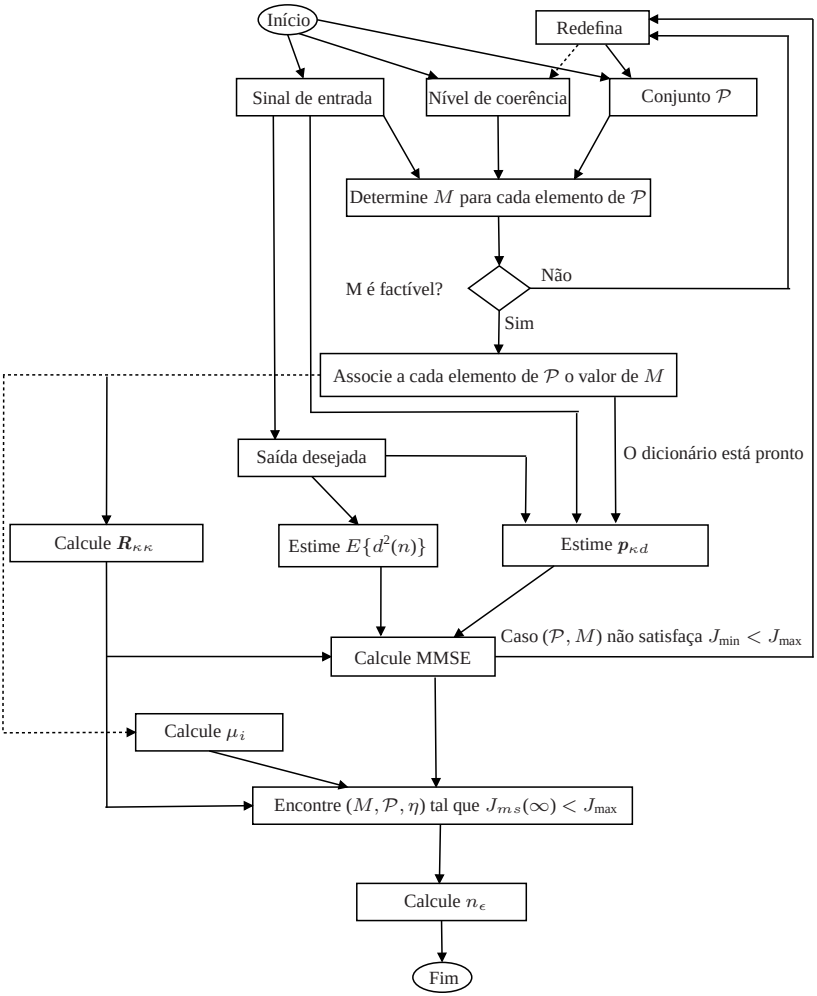


Figura 5.1: Diagrama em blocos das diretrizes para um projeto de filtro usando o algoritmo KLMS para um dado kernel.

5.3 Simulações e resultados

Esta seção apresenta três exemplos para ilustrar o procedimento de projeto proposto. Usamos esses resultados também para verificar a correspondência do modelo analítico, proposto nos Capítulos 3 e 4, com a simulação de Monte Carlo. Nesses exemplos usamos o *kernel* gaussiano e o *kernel* polinomial do segundo grau para estabelecer as comparações de desempenho. As condições de simulação descritas no Exemplo 1 são comuns a todos os exemplos. Nos demais exemplos descreveremos apenas as modificações feitas.

5.3.1 Exemplo 1

Considere o problema de estimação do seguinte sistema não-linear, modificado de (NARENDRA; PARTHASARATHY, 1990; MANDIC, 2004):

$$\begin{cases} y(n) = \dfrac{y(n-1)}{1+y^2(n-1)} + u^3(n-1) \\ d(n) = y(n) + z(n) \end{cases} \tag{5.1}$$

em que a saída $y(n)$ foi corrompida por um ruído gaussiano branco de média nula $z(n)$ com variância $\sigma_z^2 = 10^{-4}$. A entrada $u(n)$ é uma sequência também gaussiana de média nula i.i.d com desvio padrão $\sigma_u = 0,15$.

A metodologia proposta foi testada com um MSE máximo $J_{\max} = -22,35$ dB, um nível de coerência $\varepsilon_0^{GK} = 10^{-5}$ e um conjunto de parâmetros $\xi \in \mathcal{P}_{GK} = \{0,0075; 0,0125; 0,015; 0,0225\}$ para o *kernel* gaussiano e um nível de coerência $\varepsilon_0^{PK} = 0,7$ e um conjunto de parâmetros $\beta = 2$ e $a \in \mathcal{P}_{PK} = \{0,1; 0,2; 0,3; 0,5\}$ para o *kernel* polinomial. Os valores de ξ e a foram escolhidos por se mostrarem mais adequados para esta aplicação.

Para cada valor de parâmetro do *kernel*, em ambos os *kernels* utilizados, 500 dicionários com dimensão M_i, $i = 1,\ldots,500$, foram determinados usando 500 iterações do sinal de entrada. Cada M_i foi determinado como o mínimo comprimento de dicionário necessário para atingir um nível de coe-

rência[4] ε_0. O valor M foi determinado como a média de todos os M_i, o inteiro mais próximo, para todo conjunto de parâmetros do *kernel* testado. O mínimo MSE, $J_{\min}$, foi calculado usando (3.8) para cada par (ξ, M) para o *kernel* gaussiano e (a, M) para o *kernel* polinomial do segundo grau, pares formados pelo parâmetros do *kernel* e o respectivo comprimento do dicionário associado. Os valores dos momentos de segunda ordem $\boldsymbol{p}_{\kappa d}$ e $E\{d^2(n)\}$ usados foram estimados pela média de 500 realizações (Monte Carlo) em todos os casos avaliados.

Antes de encontrar um valor de η, tal que $J_{ms}(\infty) < J_{\max}$, devemos definir $\eta_{\max}$ para cada elemento de $\mathcal{P}_{GK}$ e de $\mathcal{P}_{PK}$. Para isto temos 2 possibilidades: 1) usando a análise dos discos de Gerschgorin para obtermos o valor de $\eta_{\max}$ denotado como $\eta_{\max}^{GD}$, 2) calcular e testar os autovalores de $\boldsymbol{G}$ gerando um valor de $\eta_{\max}$ denotado por $\eta_{\max}^{EG}$. As Tabelas 5.1 e 5.2 mostram que, como esperado, a condição imposta pelos discos de Gerschgorin é mais restritiva que a imposta pelos autovalores de $\boldsymbol{G}$. No entanto, note que escolhendo η a partir de $\eta_{\max} = \eta_{\max}^{GD}$ é mais simples e geralmente produz bons resultados de projeto. A Tabela 5.3 apresenta os resultados obtidos para os

Tabela 5.1: Limites de estabilidade do Exemplo 1 usando KLMS gaussiano

ξ	M	$\eta_{\max}^{EG}$	$\eta_{\max}^{GD}$
0,0075	17	1,70	0,29
0,0125	11	1,67	0,16
0,015	9	1,70	0,33
0,0225	6	1,73	0,63

valores escolhidos de ξ e a Tabela 5.4 apresenta os resultados para os valores escolhidos de a. O passo η foi escolhido de tal maneira que o sistema seja assintoticamente estável ($\eta \leq \eta_{\max}$) e $J_{ms}(\infty) < -22,35$ dB.

Os valores de $J_{ms}(\infty)$ e $J_{ex}(\infty)$ foram determinados a partir de (3.66)

[4] ε_0^{GK} e ε_0^{PK}, respectivamente, para o *kernel* gaussiano e polinomial.

Tabela 5.2: Limites de estabilidade do Exemplo 1 usando KLMS polinomial

a	M	η_{max}^{EG}	η_{max}^{GD}
0,1	3	481,87	368,98
0,2	2	443,23	435,06
0,3	2	109,02	108,39
0,5	2	15,33	15,31

e n_ϵ foi obtido a partir de (3.59) para $\epsilon_{GK} = 0,5 \times 10^{-4}$ para o caso do *kernel* gaussiano e $\epsilon_{PK} = 2,5 \times 10^{-4}$ para o caso do *kernel* polinomial. Note que $J_{\text{min}} < -22,35$ dB em todos os casos. Por um lado, é evidente que $(M; \xi; \eta) = (9; 0,015; 0,0815)$ é uma boa escolha para um projeto usando o *kernel* gaussiano, já que satisfaz as condições de projeto com apenas $n_{\epsilon_{GK}} =$ 1141 iterações. De maneira análoga, $(M; a; \eta) = (2; 0,2; 0,8072)$ é uma boa escolha para um projeto usando o *kernel* polinomial, porém são necessárias $n_{\epsilon_{PK}} = 11286$ iterações . Assim, podemos escolher como a melhor opção para projeto o *kernel* gaussiano com $(M; \xi; \eta) = (9; 0,015; 0,0815)$ por satisfazer as condições estabelecidas em menos iterações.

Note que o valor de η escolhido foi inferior a $1/10$ de η_{max} para ambos os *kernels* gaussiano e polinomial. Isto é devido ao pequeno valor de J_{max} imposto pelo projeto. Esse mesmo comportamento acontece quando projetamos os algoritmo LMS convencional para atender especificações práticas em problemas de estimação linear.

Para cada simulação, a ordem M do dicionário permanece fixa. O dicionário foi inicializado em cada realização a partir da geração de um conjunto de M funções *kernel* da entrada, de tal maneira que este conjunto tenha um nível de coerência igual ou inferior ao estabelecido no projeto. Assim, o dicionário será diferente em cada realização. Durante cada realização, os elementos que compõem o dicionário são atualizados de tal maneira que na iteração n o elemento mais antigo é substituído por $\boldsymbol{u}(n-1)$.

As Figuras 5.2 e 5.3 ilustram a acurácia do modelo analítico para os quatro casos apresentados na Tabela 5.3. De maneira similar, as Figuras 5.4 e 5.5 representam a Tabela 5.4. As Figuras 5.2 e 5.4 mostram uma excelente concordância entre simulações de Monte Carlo, usando uma média sobre 500 realizações, e as previsões teóricas feitas usando (3.16) e (3.57). As Figuras 5.3 e 5.5 comparam as previsões do regime permanente (linhas pontilhadas) usando (3.58) com o resultado simulado. Novamente ocorre a concordância entre os valores simulados e os valores previstos pelo modelo de comportamento estocástico.

Tabela 5.3: Resumo dos resultados simulados para o Exemplo 1 usando o kernel gaussiano.

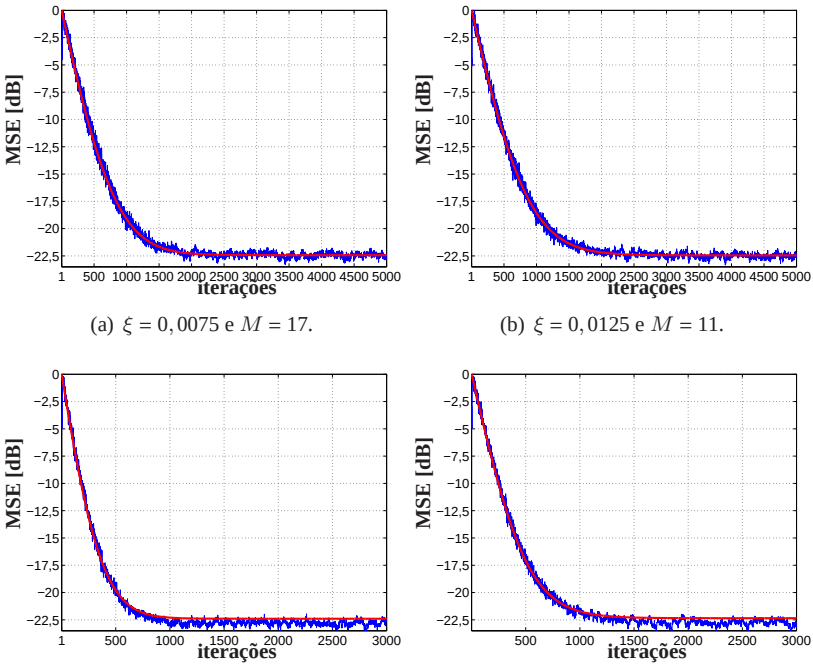
ξ	M	η	$J_{\min}$ [dB]	$J_{ms}(\infty)$ [dB]	$J_{ex}(\infty)$ [dB]	$n_{\epsilon_{GK}}$
0,0075	17	0,0736	-22,51	-22,43	-40,37	2421
0,0125	11	0,0399	-22,49	-22,45	-42,78	2668
0,015	9	0,0815	-22,48	-22,40	-39,64	1141
0,0225	6	0,0393	-22,39	-22,35	-42,83	1595

Tabela 5.4: Resumo dos resultados simulados para o Exemplo 1 usando o kernel polinomial.

a	M	η	$J_{\min}$ [dB]	$J_{ms}(\infty)$ [dB]	$J_{ex}(\infty)$ [dB]	$n_{\epsilon_{PK}}$
0,1	3	0,6846	-22,35	-22,35	-60,01	69245
0,2	2	0,8072	-22,37	-22,36	-50,62	11286
0,3	2	0,2011	-22,36	-22,35	-49,95	15271
0,5	2	0,0071	-22,36	-22,35	-55,78	87768

5.3.2 Exemplo 2

Como um segundo exemplo de projeto, considere o problema de identificação de sistemas dinâmicos não-linear, modificado de (VÖRÖS, 2003),



(a) $\xi = 0,0075$ e $M = 17$.

(b) $\xi = 0,0125$ e $M = 11$.

(c) $\xi = 0,015$ e $M = 9$.

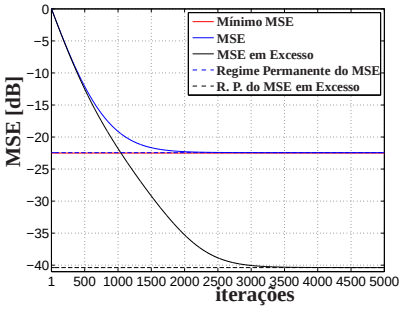
(d) $\xi = 0,0225$ e $M = 6$.

Figura 5.2: Modelo teórico e simulações (Monte Carlo) do KLMS gaussiano usando diferentes parâmetros para o Exemplo 1. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) e (3.57).

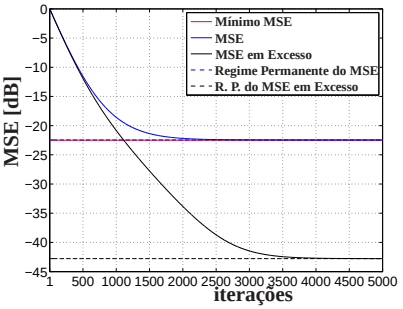
em que o sinal de entrada é uma sequência de vetores estatisticamente independentes,

$$\boldsymbol{u}(n) = [u_1(n) \;\; u_2(n)]^\top \tag{5.2}$$

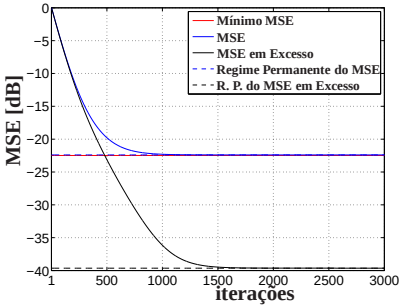
com componentes correlacionadas, tal que, $u_1(n) = 0,5u_2(n) + \eta_u(n)$. A segunda componente de $\boldsymbol{u}(n)$ em (5.2) é uma sequência gaussiana com variância $\sigma^2_{u_2} = 0,0156$ e $\eta_u(n)$ é um ruído gaussiano branco tal que $u_1(n)$ tem variância $\sigma^2_{u_1} = 0,0156$.



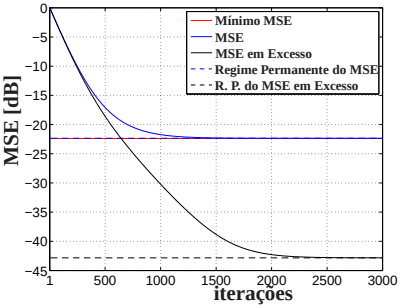
(a) $\xi = 0,0075$ e $M = 17$.



(b) $\xi = 0,0125$ e $M = 11$.



(c) $\xi = 0,015$ e $M = 9$.

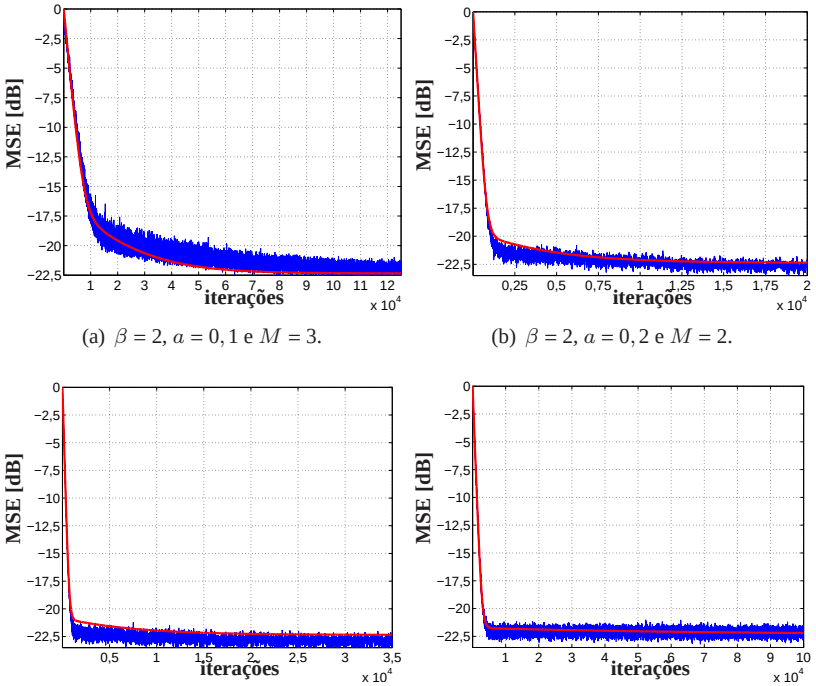


(d) $\xi = 0,0225$ e $M = 6$.

Figura 5.3: Resultados do KLMS gaussiano em regime permanente para o Exemplo 1. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente.

Considere um sistema linear com memória definido por:

$$y(n) = \boldsymbol{b}^{\top}\, \boldsymbol{u}(n) - 0,2\, y(n-1) + 0,35\, y(n-2) \tag{5.3}$$



(a) $\beta = 2$, $a = 0,1$ e $M = 3$.

(b) $\beta = 2$, $a = 0,2$ e $M = 2$.

(c) $\beta = 2$, $a = 0,3$ e $M = 2$.

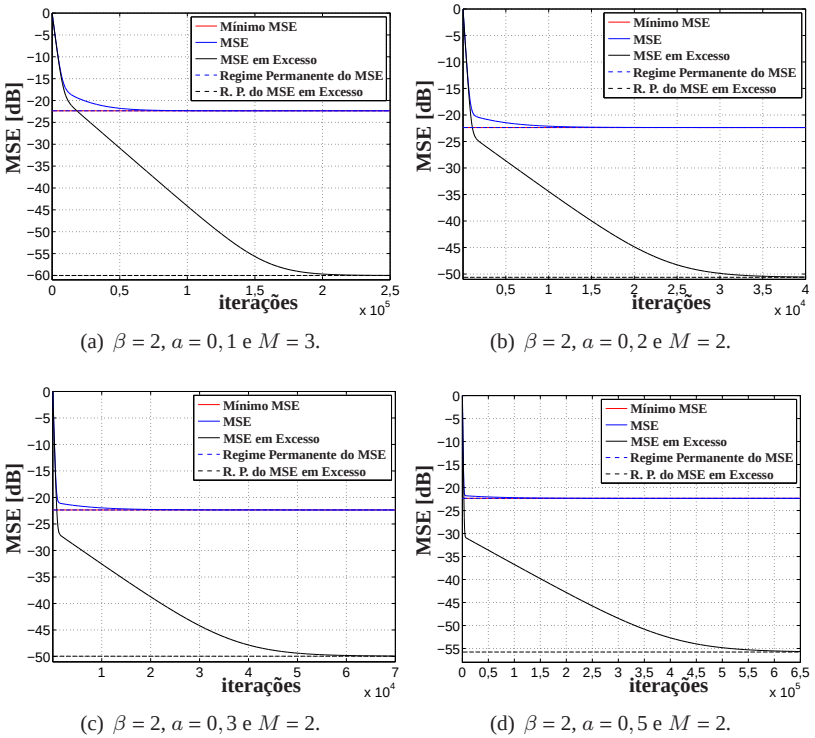
(d) $\beta = 2$, $a = 0,5$ e $M = 2$.

Figura 5.4: Modelo teórico e simulações (Monte Carlo) do KLMS polinomial usando diferentes parâmetros para o Exemplo 1. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) e (3.57).

em que $\boldsymbol{b} = [1 \;\; 0,5]^\top$ e uma função não-linear de Wiener

$$\varphi(y(n)) = \begin{cases} \dfrac{y(n)}{3[0,1 + 0,9\,y^2(n)]^{1/2}} & \text{para } \; y(n) \geq 0 \\ \dfrac{-y^2(n)[1 - \exp(0,7\,y(n))]}{3} & \text{para } \; y(n) < 0, \end{cases} \tag{5.4}$$

$$d(n) = \varphi(y(n)) + z(n) \tag{5.5}$$



(a) $\beta = 2$, $a = 0,1$ e $M = 3$.

(b) $\beta = 2$, $a = 0,2$ e $M = 2$.

(c) $\beta = 2$, $a = 0,3$ e $M = 2$.

(d) $\beta = 2$, $a = 0,5$ e $M = 2$.

Figura 5.5: Resultados do KLMS polinomial em regime permanente para o Exemplo 1. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente.

em que $d(n)$ é a saída desejada, corrompida por um ruído gaussiano branco de média nula $z(n)$ e variância $\sigma_z^2 = 10^{-6}$. A condição inicial considerada neste problema é $y(1) = 0$.

Neste exemplo, testamos as diretrizes de projeto considerando um máximo MSE $J_{\max} = -20,25$ dB. E os parâmetros dos respectivos KLMS gaussiano e KLMS polinomial usados foram: níveis de coerência $\varepsilon_0^{GK} =$

10^{-4} e $\varepsilon_0^{PK} = 0,85$, os parâmetros do *kernel* escolhidos para teste foram $\xi \in \{0,05; 0,065; 0,075; 0,125\}$ e $a \in \{0,5; 0,75; 1,25; 1,5\}$. As demais condições de simulação são similares ao primeiro exemplo.

As Tabelas 5.5 e 5.6 exibem os limites estabelecidos para o passo do KLMS, $\eta_{\max}$, obtidos a partir da análise dos discos de Gerschgorin e do estudo dos autovalores da matriz $\boldsymbol{G}$. Usamos a expressão (w.s.) (do inglês *without solution*) para indicar que a interseção das soluções provenientes de (3.49) e (3.56) é vazia. Nos casos em que isto ocorre, devemos usar os limites obtidos numericamente a partir dos autovalores da matriz $\boldsymbol{G}$, isto é, $\eta_{\max} = \eta_{\max}^{EG}$. As

Tabela 5.5: Limites de estabilidade do Exemplo 2 usando KLMS gaussiano

ξ	M	$\eta_{\max}^{EG}$	$\eta_{\max}^{GD}$
0,050	7	2,33	*w.s.*
0,065	4	2,50	0,68
0,075	3	2,60	1,92
0,125	2	2,34	2,32

Tabela 5.6: Limites de estabilidade do Exemplo 2 usando KLMS polinomial.

a	M	$\eta_{\max}^{EG}$	$\eta_{\max}^{GD}$
0,50	4	7,84	*w.s.*
0,75	3	2,08	*w.s.*
1,25	2	0,41	0,41
1,50	2	0,20	0,20

Tabelas 5.7 e 5.8 mostram os resultados obtidos para os parâmetros escolhidos, ξ (*kernel* gaussiano) e a (*kernel* polinomial). Para cada par (ξ, M) e (a, M), o passo do KLMS, η, foi escolhido a fim de garantir a estabilidade do algoritmo ($\eta < \eta_{\max}$) e obter $J_{ms}(\infty) < -20,25$ dB. Os valores de $J_{ms}(\infty)$ e $J_{ex}(\infty)$ foram determinados usando as Equação (3.66) e o número de iterações do KLMS gaussiano ($n_{\epsilon_{GK}}$) e do KLMS polinomial ($n_{\epsilon_{PK}}$) foram obtidos usando (3.59) para $\epsilon_{GK} = 10^{-3}$ e $\epsilon_{PK} = 2,5 \times 10^{-4}$.

Note que $J_{ms}(\infty) < -20,25$ dB em todos os casos testados. Assim, avaliando os parâmetros testados verificamos que o *kernel* polinomial com (M; a; η) = (2; 1,25; 0,102) é a melhor opção para projeto, por atingir o $J_{ms}(\infty)$ desejado após $n_{\epsilon_{PK}} = 7$ iterações.

As Figuras 5.6– 5.9 ilustram a exatidão do modelo analítico para os quatro casos apresentados na Tabela 5.7 e na Tabela 5.8. Novamente, o acordo entre a teoria e as simulações é excelente.

Tabela 5.7: Resumo dos resultados simulados para o Exemplo 2 usando o kernel gaussiano.

ξ	M	η	$J_{\min}$ [dB]	$J_{ms}(\infty)$ [dB]	$J_{ex}(\infty)$ [dB]	$n_{\epsilon_{GK}}$
0,050	7	0,0724	-20,32	-20,27	-39,73	1544
0,065	4	0,2394	-20,41	-20,25	-34,63	314
0,075	3	0,3398	-20,68	-20,45	-33,33	184
0,125	2	0,0728	-20,54	-20,47	-38,39	502

Tabela 5.8: Resumo dos resultados simulados para o Exemplo 2 usando o kernel polinomial.

a	M	η	$J_{\min}$ [dB]	$J_{ms}(\infty)$ [dB]	$J_{ex}(\infty)$ [dB]	$n_{\epsilon_{PK}}$
0,50	4	0,4902	-21,22	-20,94	-33,01	1142
0,75	3	0,1304	-21,32	-21,04	-33,10	47
1,25	2	0,102	-21,98	-20,74	-26,76	7
1,50	2	0,0123	-21,30	-21,02	-33,06	33

5.3.3 Exemplo 3

Para um terceiro exemplo, considere o problema de controle de vazão de fluido, modificado de (WANG; HSU, 2008; AL-DUWAISH; KARIM; CHANDRASEKAR, 1996), no qual o sinal de entrada é uma sequência ve-

tores estatisticamente independente

$$\boldsymbol{u}(n) = [u_1(n)\ \ u_2(n)]^\top \tag{5.6}$$

com componentes correlacionadas por um processo AR(1), $u_1(n) = 0,5\,u_2(n) + \eta_u(n)$, em que $u_2(n)$ é um ruído gaussiano branco com variância $\sigma^2_{u_2} = 0,0625$ e $\eta_u(n)$ é gaussiano branco tal que $u_1(n)$ tem variância $\sigma^2_{u_1} = 0,0625$. Considere o sistema linear com memória definido por

$$y(n) = \boldsymbol{b}^\top \boldsymbol{u}(n) + 1.4138\,y(n-1) - 0,6065\,y(n-2) \tag{5.7}$$

com $\boldsymbol{b} = [0,1044\ \ 0,0883]^\top$ e um sistema não-linear de Wiener

$$d(n) = \frac{0,3163\,y(n)}{\sqrt{0,10 + 0,90\,y^2(n)}} + z(n) \tag{5.8}$$

em que a saída desejada $d(n)$ foi corrompida por um ruído gaussiano branco $z(n)$ com variância $\sigma^2_z = 10^{-4}$.

A condição de projeto desejada é: $J_{ms}(\infty) < J_{\max} = -19,75$ dB (MSE máximo). Os parâmetros de projeto testados foram: $\varepsilon_0^{GK} = 10^{-1}$; $\varepsilon_0^{PK} = 0,6$; $\xi \in \{0,15;\ 0,20;\ 0,25;\ 0,30\}$ e $a \in \{0,25;\ 0,5;\ 1,25;\ 2\}$.

Tabela 5.9: Limites de estabilidade do Exemplo 3 usando KLMS gaussiano.

ξ	M	$\eta_{\max}^{EG}$	$\eta_{\max}^{GD}$
0,15	11	1,17	*w.s.*
0,20	7	1,19	*w.s.*
0,25	5	1,24	*w.s.*
0,30	3	1,60	*w.s.*

As Tabelas 5.9 e 5.10 mostram os valores de $\eta_{\max}$ obtidos pelo teste dos autovalores de $\boldsymbol{G}$ e pela análise dos discos de Gerschgorin. Note que as condições impostas pelos discos de Gerschgorin foram também mais restritivas para todos os casos em que conseguimos calcular $\eta_{\max}^{GD}$, como o esperado. Aqueles que não foram possíveis calcular representamos por *w.s.*.

Tabela 5.10: Limites de estabilidade do Exemplo 3 usando KLMS polinomial.

a	M	$\eta_{\max}^{EG}$	$\eta_{\max}^{GD}$
0,25	5	9,30	*w.s.*
0,5	3	6,71	*w.s.*
1,25	2	0,38	0,38
2,0	2	0,06	0,06

As Tabelas 5.11 e 5.12 apresentam, respectivamente, os resultados aos valores escolhidos de ξ e a. Para cada par de (ξ, M) (*kernel* gaussiano) e (a, M) (*kernel* polinomial), o passo η foi determinado para garantir a estabilidade (η menor que $\eta_{\max}$) e $J_{ms}(\infty) < -19,75$ dB. Os valores $J_{ms}(\infty)$ e $J_{ex}(\infty)$ foram determinados a partir de (3.66) e n_ϵ foi obtido a partir de (3.59) com $\epsilon_{GK} = 10^{-4}$ e $\epsilon_{PK} = 5 \times 10^{-4}$. Note que $J_{\min}(\xi) < -19,75$ dB em todos os casos. É evidente que $(M; \xi; \eta)$ = (7; 0,2; 0,149) é uma boa escolha para o *kernel* gaussiano, pois satisfaz a condição de projeto exigida após $n_{\epsilon_{GK}} = 382$ iterações, enquanto para o *kernel* polinomial $(M; \xi; \eta)$ = (5; 0,25; 0,4651) se mostra a melhor escolha com $n_{\epsilon_{PK}} = 1343$. Sobre as condições impostas neste exemplo o uso do *kernel* gaussiano com $(M; \xi; \eta)$ = (7; 0,2; 0,1494) se torna novamente a opção mais atrativa para projeto.

As Figuras 5.10 – 5.13 ilustram a precisão do modelo analítico para os casos apresentados nas Tabelas 5.11 e 5.12. Novamente, os resultados mostraram bastante promissores para este exemplo.

Tabela 5.11: Resumo dos resultados simulados para o Exemplo 3 usando o kernel gaussiano.

ξ	M	η	$J_{\min}$ [dB]	$J_{ms}(\infty)$ [dB]	$J_{ex}(\infty)$ [dB]	$n_{\epsilon_{GK}}$
0,15	11	0,1466	-20,56	-20,20	-31,24	517
0,20	7	0,1494	-20,41	-20,03	-30,81	382
0,25	5	0,0388	-20,07	-19,97	-36,68	1298
0,30	3	0,0125	-19,78	-19,75	-42,48	4052

Tabela 5.12: Resumo dos resultados simulados para o Exemplo 3 usando o kernel polinomial.

a	M	η	$J_{\min}$ [dB]	$J_{ms}(\infty)$ [dB]	$J_{ex}(\infty)$ [dB]	$n_{\epsilon_{PK}}$
0,25	5	0,4651	-19,80	-19,75	-39,81	1343
0,50	3	0,0839	-19,89	-19,85	-39,92	1609
1,25	2	0,0012	-19,77	-19,76	-44,95	6829
2,00	2	$4{,}75 \times 10^{-5}$	-19,76	-19,75	-50,88	5031

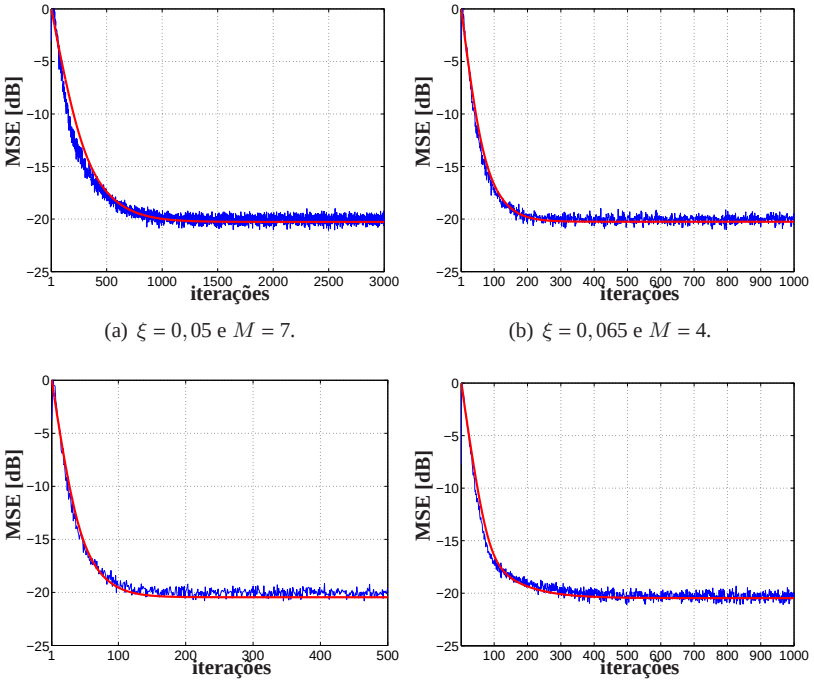
5.4 Conclusões

Neste capítulo apresentamos alguns exemplos de aplicação das diretrizes de projeto. Essas diretrizes foram sugeridas a partir dos resultados obtidos da análise do comportamento estocástico do algoritmo adaptativo KLMS. O projeto proposto pode ser dividido em duas partes. Na primeira parte o dicionário é construído a partir do nível de coerência e dos parâmetros do *kernel* que serão testados. Os dicionários são construídos para diversas realizações de $\boldsymbol{u}(n)$ de maneira que se tornem estáveis em M_i (com i variando de 1 à 500). Em seguida definimos M como sendo a média sobre estas realizações. Desta maneira podemos afirmar que M é dependente das estatísticas da entrada, $\boldsymbol{u}(n)$. Após esse processo associamos a cada parâmetro do *kernel* um referido valor de M. Este par é usado para calcular analiticamente $\boldsymbol{R}_{\kappa\kappa}$ e estimar $\boldsymbol{p}_{\kappa d}$ com a finalidade de obter $J_{\min}$. Quando o parâmetro do *kernel* associado a uma dimensão M gerar um $J_{\min}$ tal que $J_{\min} < J_{\max}$ seguiremos para a segunda parte. Na segunda parte são avaliados limites de estabilidade e o tempo de convergência para cada um dos parâmetros do *kernel* (que já está associado a determinado valor de M) e escolhemos aquele cujo $J_{ms}(\infty) < J_{\max}$ com a menor quantidade de iterações.

Os resultados encontrados mostraram-se bastante promissores. Apesar de terem sido utilizadas apenas duas funções *kernel* distintas o estudo

pode ser estendido para outros *kernels*. Nos três exemplos apresentados foi possível optar pelo uso de um determinado *kernel* de maneira analítica e não de maneira empírica. Optamos pela utilização do *kernel* gaussiano (Exemplo 1 e 3 respectivamente apresentados na Subseções 5.3.1 e 5.3.3) e do *kernel* polinomial (Exemplo 2 apresentado na Subseção 5.3.2), e quais parâmetros eram mais favoráveis para aplicação avaliando a velocidade de convergência, uma vez que todos os parâmetros exibidos atingiram o desempenho exigido ($J_{ms}(\infty) < J_{\max}$).

Um outro ponto que podemos ressaltar dos resultados encontrados nos exemplos, é a análise da estabilidade. Em alguns casos não foi possível calcular analiticamente o $\eta_{\max}$ e foi necessário recorrer a testes numéricos envolvendo os autovalores de $\boldsymbol{G}$. Apesar de parecer complicado, obter $\eta_{\max}$ avaliando os discos de Gerschgörin é facilmente implementável. E por ser uma condição mais restritiva, em alguns casos se torna uma opção mais interessante para iniciarmos uma busca por η tal que η satisfaça a condição de projeto $J_{ms}(\infty) < J_{\max}$.



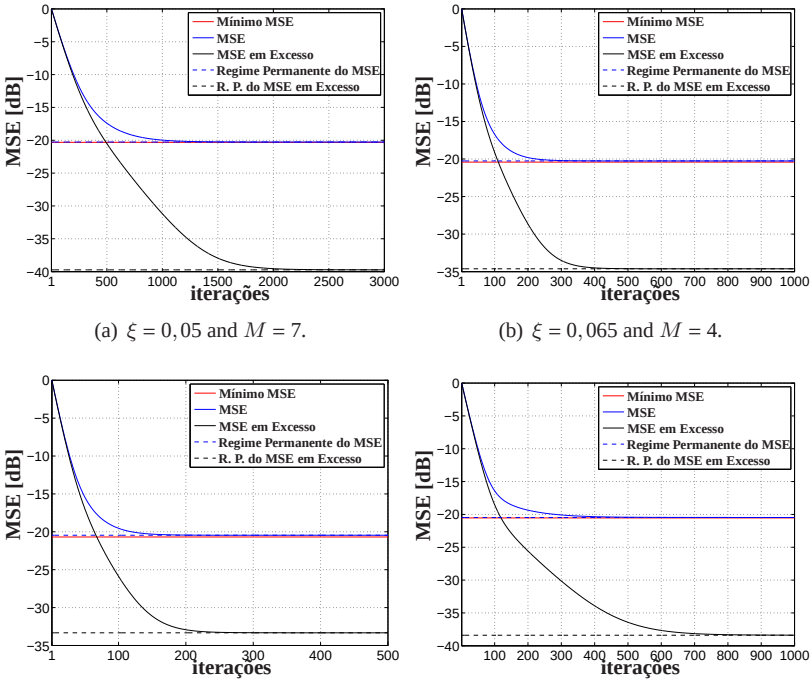
(a) $\xi = 0,05$ e $M = 7$.

(b) $\xi = 0,065$ e $M = 4$.

(c) $\xi = 0,075$ e $M = 3$.

(d) $\xi = 0,125$ e $M = 2$.

Figura 5.6: Modelo teórico e simulações (Monte Carlo) do KLMS gaussiano usando diferentes parâmetros para o Exemplo 2. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) e (3.57).



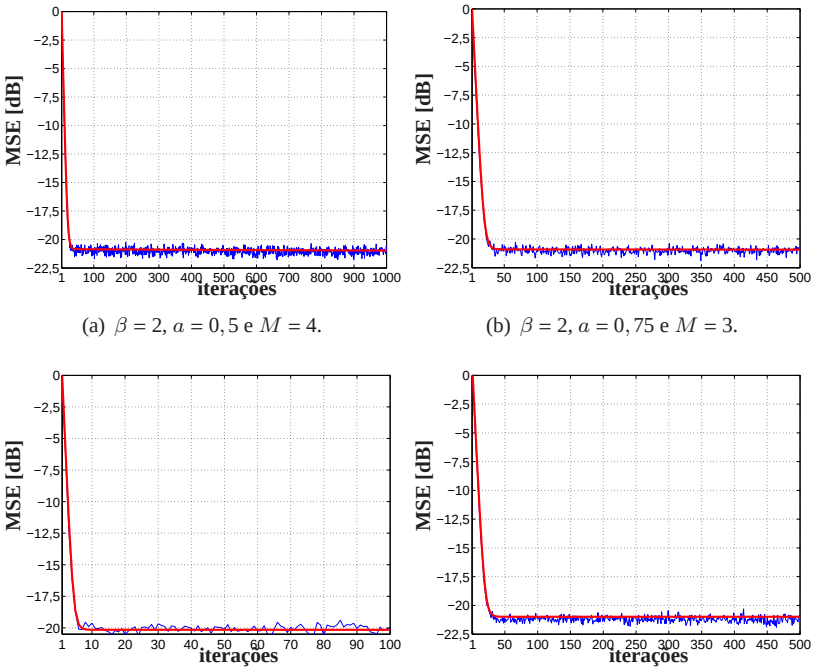
(a) $\xi = 0,05$ and $M = 7$.

(b) $\xi = 0,065$ and $M = 4$.

(c) $\xi = 0,075$ and $M = 3$.

(d) $\xi = 0,125$ and $M = 2$.

Figura 5.7: Resultados do KLMS gaussiano em regime permanente para o Exemplo 2. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente.



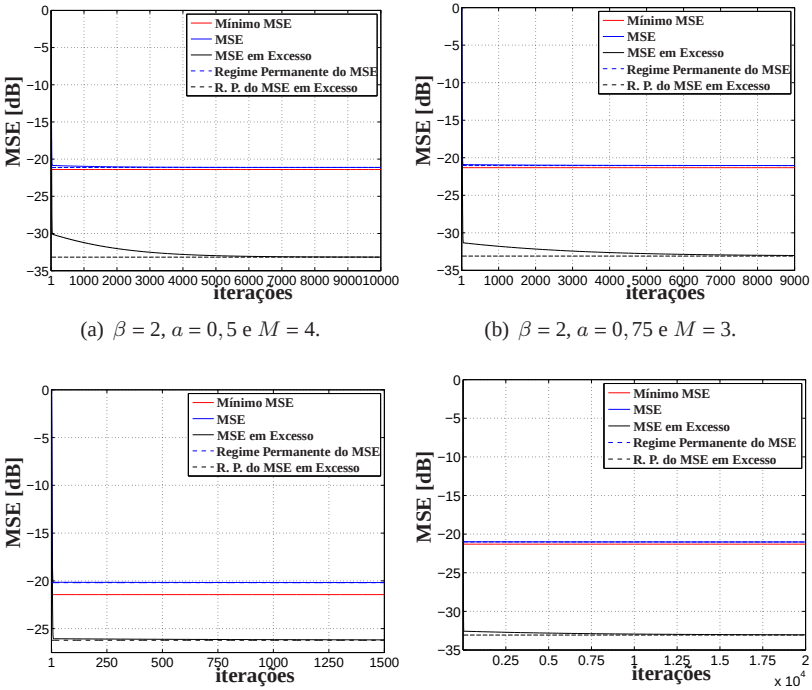
(a) $\beta = 2$, $a = 0,5$ e $M = 4$.

(b) $\beta = 2$, $a = 0,75$ e $M = 3$.

(c) $\beta = 2$, $a = 1,25$ e $M = 2$.

(d) $\beta = 2$, $a = 1,5$ e $M = 2$.

Figura 5.8: Modelo teórico e simulações (Monte Carlo) do KLMS polinomial usando diferentes parâmetros para o Exemplo 2. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) and (3.57).



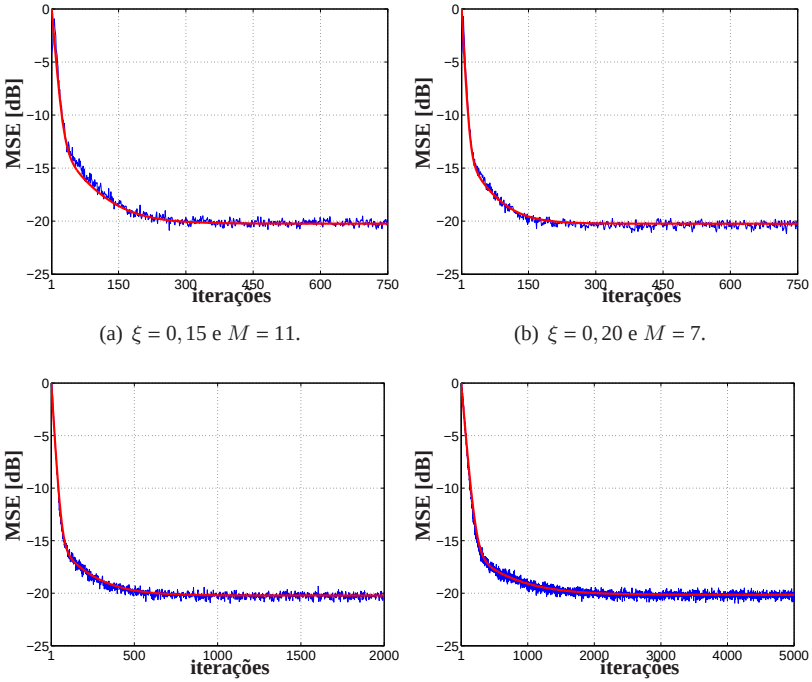
(a) $\beta = 2$, $a = 0,5$ e $M = 4$.

(b) $\beta = 2$, $a = 0,75$ e $M = 3$.

(c) $\beta = 2$, $a = 1,25$ e $M = 2$.

(d) $\beta = 2$, $a = 1,5$ e $M = 2$.

Figura 5.9: Resultados do KLMS polinomial em regime permanente para o Exemplo 2. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente.



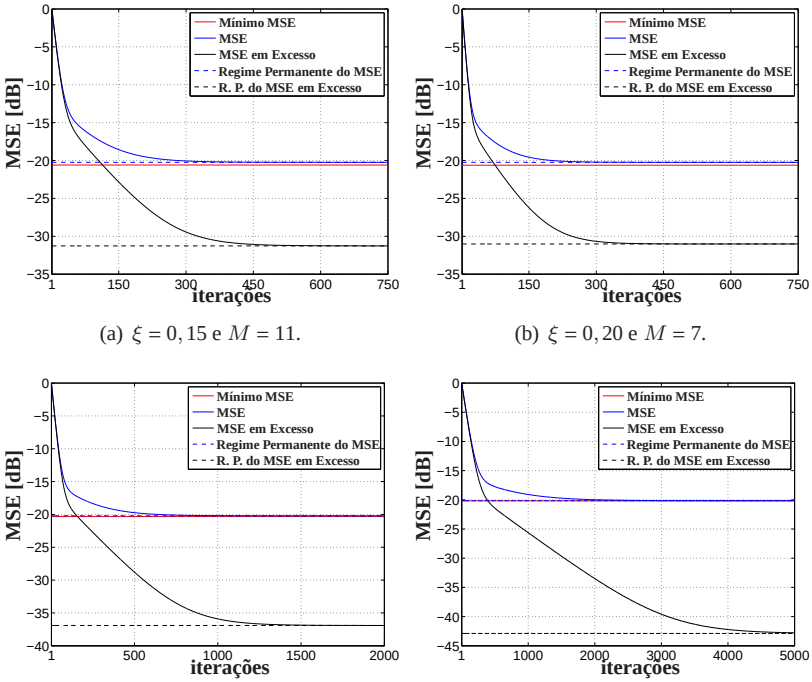
(a) $\xi = 0,15$ e $M = 11$.

(b) $\xi = 0,20$ e $M = 7$.

(c) $\xi = 0,25$ e $M = 5$.

(d) $\xi = 0,30$ e $M = 3$.

Figura 5.10: Modelo teórico e simulações (Monte Carlo) do KLMS gaussiano usando diferentes parâmetros para o Exemplo 3. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) e (3.57).



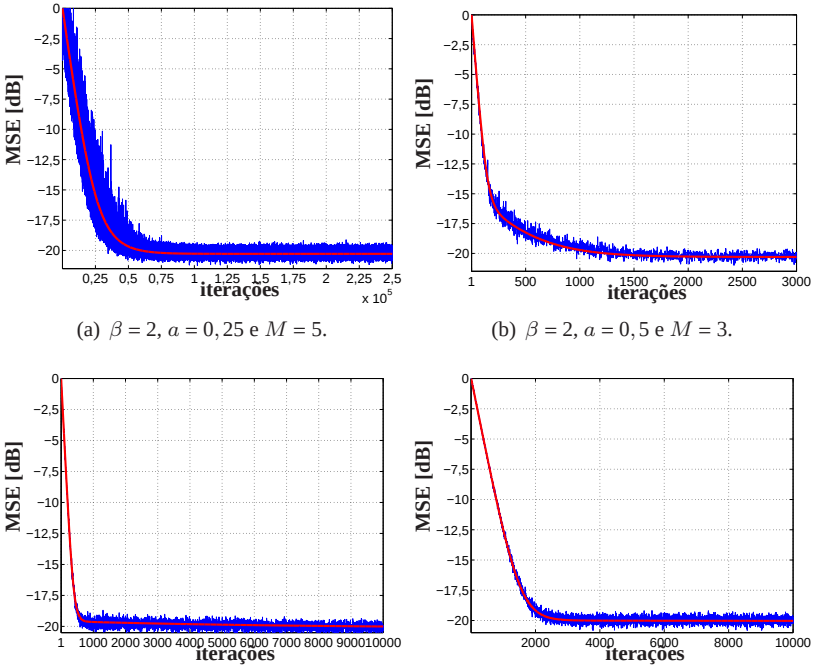
(a) $\xi = 0,15$ e $M = 11$.

(b) $\xi = 0,20$ e $M = 7$.

(c) $\xi = 0,25$ e $M = 5$.

(d) $\xi = 0,30$ e $M = 3$.

Figura 5.11: Resultados do KLMS gaussiano em regime permanente para o Exemplo 3. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente.



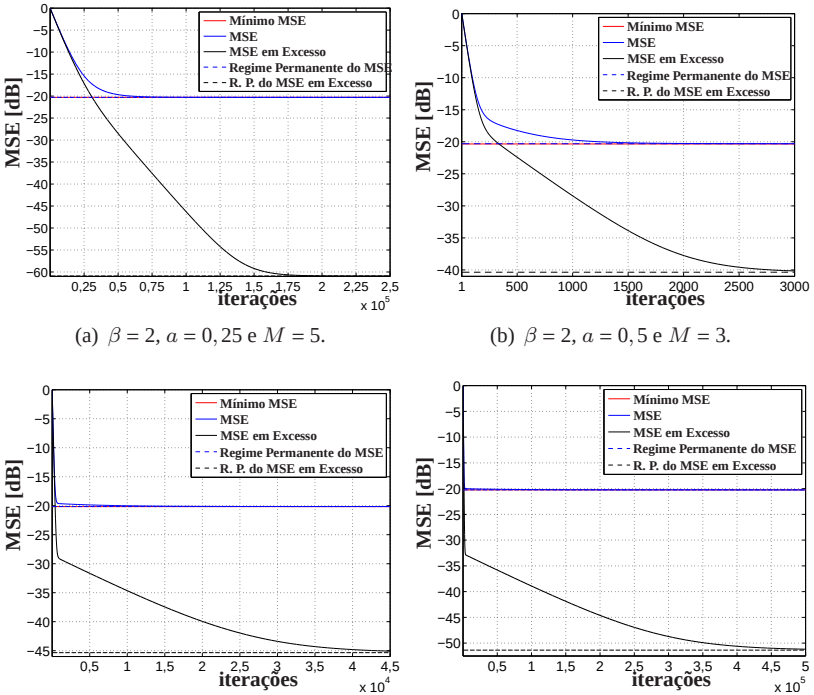
(a) $\beta = 2$, $a = 0,25$ e $M = 5$.

(b) $\beta = 2$, $a = 0,5$ e $M = 3$.

(c) $\beta = 2$, $a = 1,25$ e $M = 2$.

(d) $\beta = 2$, $a = 2,0$ e $M = 2$.

Figura 5.12: Modelo teórico e simulações (Monte Carlo) do KLMS polinomial usando diferentes parâmetros para o Exemplo 3. A curva em azul representa a média sobre 500 realizações. A curva em vermelho representa o modelo teórico usando (3.16) e (3.57).



(a) $\beta = 2$, $a = 0,25$ e $M = 5$.

(b) $\beta = 2$, $a = 0,5$ e $M = 3$.

(c) $\beta = 2$, $a = 1,25$ e $M = 2$.

(d) $\beta = 2$, $a = 2$ e $M = 2$.

Figura 5.13: Resultados do KLMS polinomial em regime permanente para o Exemplo 3. Curva contínua em vermelho representa o MSE mínimo. Curva contínua em azul representa o MSE. Curva contínua em preto representa o MSE em excesso. As linhas pontilhadas representam a previsão do MSE (azul) e MSE em excesso (preto) em regime permanente.

6 CONCLUSÕES E PROPOSTAS DE CONTINUIDADE DA PESQUISA

6.1 Conclusões e contribuições

Os *kernels* são estruturas matemáticas que têm ganhado espaço no cenário de filtragem adaptativa. O Teorema da Representação, as propriedades dos *kernels* no espaço RKHS e o chamado "truque do *kernel*" dão o suporte para que um sinal não-linear qualquer possa ser estimado por uma combinação linear de *kernels*. Para estimar os coeficientes dessa combinação linear de funções *kernel*, muitos autores recorrem aos algoritmos adaptativos. Porém, em aplicações práticas, torna-se inviável empregar um modelo no RKHS cujo o número de coeficientes cresce com o número de observações realizadas. O uso do modelo de ordem finita vem da dificuldade de analisar o comportamento estocástico do algoritmo adaptativo no espaço das características.

Uma das principais dificuldades existentes no projeto de filtros adaptativos não-lineares usando *kernel* é a escolha do *kernel* mais adequado à aplicação, bem como o projeto otimizado dos parâmetros do *kernel* escolhido. Nosso trabalho contribuiu para novas discussões sobre filtragem adaptativa envolvendo algoritmos baseados em *kernel*. Estudamos o comportamento estocástico do algoritmo adaptativo KLMS para entradas gaussianas estacionárias e não-linearidades que preservam a estacionaridade da entrada. Apresentamos as equações que modelam o comportamento médio e médio quadrático dos coeficientes do algoritmo adaptativo KLMS. Para derivarmos as expressões estocásticas que modelam os momentos de segunda ordem, usamos dicionários fixos e finitos obtidos por um processo de esparsificação e algumas hipóteses simplificadoras como MIA.

O estudo resultou em um modelo analítico que prediz o comportamento do algoritmo como uma função dos parâmetros de projeto (passo de

adaptação, função *kernel* e seus parâmetros). Em particular, o novo modelo esclarece a contribuição conjunta dos parâmetros do *kernel* e do passo para a performance do algoritmo em ambos os períodos de adaptação dos coeficientes (regime transitório e regime permanente). Dado que a função *kernel* realiza uma transformação não-linear do espaço de entrada, foi preciso um estudo mais detalhado do comportamento dos momentos de quarta ordem do *kernel*. As expressões para esses momentos foram derivadas para o caso em que o KLMS opera como o *kernel* Gaussiano e *kernel* Polinomial.

Estudamos também a convergência do algoritmo e derivamos expressões analíticas as quais fornecem condições suficientes para estabilidade. Isso foi feito utilizando a projeção lexicográfica da equação recursiva da matriz de autocorrelação do vetor de erro nos coeficientes. Por se tratar de uma condição suficiente, nem sempre foi possível aplicá-la. Nos casos em que não foi aplicada essa condição, sugerimos a análise numérica dos autovalores da matriz transição de estados.

Considerando as equações de análise obtidas neste trabalho, propomos diretrizes para projeto do um algoritmo KLMS usando *kernels* gaussiano e polinomial de segunda ordem. Essas diretrizes permitem uma escolha dos parâmetros do filtro adaptatitivo "kernelizado“ baseando-se no desempenho (qualidade da estimação e tempo de adaptação dos coeficientes), obtido pelo uso de um determinado *kernel* e seus respectivos parâmetros. Vimos que a estabilidade do algoritmo também é influenciada pela escolha do *kernel* e de seus parâmetros. Essa influência é levada em consideração na escolha do passo.

Aplicamos as diretrizes estabelecidas a diferentes problemas de identificação de sistemas. Sem perda de generalidade consideramos dicionários obtidos a partir da função coerência. Os promissores resultados ilustraram a acurácia do modelo teórico e sua utilidade para propostas de projeto.

6.2 Propostas de continuidade da pesquisa

- **Análise para outros kernels:** A partir da propriedade reprodutiva dos *kernels*, podemos explorar a análise apresentada neste trabalho para outros *kernels*. É claro que em alguns casos determinar os momentos de quarta ordem pode ser feito de maneira mais geral, diferentemente do que foi feito para *kernel* Polinomial, em que uma alteração no grau do polinômio exige que os cálculos desses momentos sejam refeitos.

- **Combinação entre funções kernel:** Uma maneira de otimizar o desempenho do algoritmo KLMS na estimação de sistemas não-lineares poderia ser combinando mais de uma função *kernel* associado a determinados pesos, por exemplo $\gamma_1\kappa_1(.,.) + \gamma_2\kappa_2(.,.)$ em que γ_1, γ_2 são pesos e $\kappa_1(.,.), \kappa_2(.,.)$ são funções *kernel* que podem ser diferentes ou apenas possuir parâmentros diferentes. Vimos neste trabalho que os *kernels* realizam uma transformação não-linear no espaço de entrada. Essa transformação resulta na alteração da superfície de desempenho. Dessa maneira não é possível saber de antemão se isto realmente é algo vantajoso. E se for, precisamos determinar esse espaço resultante dessa transformação não-linear.

- **Análise do KLMS para o passo variável:** No LMS convencional o passo de adaptação é um parâmetro que estabelece um compromisso entre velocidade de convergência e qualidade de estimação em regime permanente. Como vimos aqui, usando um parâmetro fixo do *kernel*, o KLMS pode se comportar como o LMS tradicional. Porém, ao buscarmos um ganho no desempenho do algoritmo, não podemos simplesmente fixar esse parâmetro sem a preocupação com os ganhos em regime permanente e tempo de adaptação que são alterados por esse parâmetro. Por isso, o ajuste deve ser feito em conjunto. Assim, devemos investigar a influência do passo variável e dos parâmetros no

desempenho do algoritmo.

- **Investigação considerando sistemas variantes no tempo:** Uma linha de investigação que surge quando fazemos o tipo de análise que apresentamos aqui é a extensão dos resultados para sistemas variantes no tempo. Neste caso, inicialmente precisamos definir o que seria seria variável no tempo. Poderiam ser alguns dos parâmetros do algoritmo ou a não-linearidade, por exemplo. Em ambos os exemplos, uma investigação mais detalhada é necessária para sabermos como o sistema adaptativo seria adaptado a tempo de acompanhar a variação do sistema.

REFERÊNCIAS BIBLIOGRÁFICAS

AIZERMAN, M. A.; BRAVERMAN, E. M.; ROZONOER, L. I. The method of potential functions for the problem of restoring the characteristic of a function converter from randomly observed points. *Automation and Remote Control*, v. 25, n. 12, p. 1546–1556, 1964.

AL-DUWAISH, H.; KARIM, M. N.; CHANDRASEKAR, V. Use of multilayer feedfoorward neural networks in identification and control of wiener model. In: *Proc. Control Theory Appl.* [S.l.: s.n.], 1996. v. 143, n. 3.

ARONSZAJN, N. Theory of reproducing kernels. *Trans. Amer. Math. Soc.*, v. 68, 1950.

BOSER, B.; GUYON, I. M.; VAPNIK, V. A training algorithm for optimal margin classifiers. In: HAUSSLER, D. (Ed.). *Fifth Annual ACM Workshop on Computacional Learning Theory*. [S.l.]: ACM Press, Pittsburg, PA, 1992. p. 144–152.

BOUBOULIS, P.; THEODORIDIS, S. Extension of wirtinger's calculus in reproducing kernel hilbert spaces and the complex kernel lms. *IEEE Trans. Signal Process.*, v. 59, n. 3, p. 964 – 978, March 2011.

CHEN, S. S.; DONOHO, D. L.; SAUNDERS, M. A. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, v. 20, n. 1, p. 33–61, 1999.

DAVIS, G.; MALLAT, S.; ZHANG, Z. Adaptive time-frequency decompositions. *SPIE Journal of Optical Engineering*, v. 33, n. 7, p. 2183–2191, July 1994.

DODD, T. J.; KADIRKAMANATHAN, V.; HARRISON, R. F. Function estimation in hilbert space using sequential projections. In: *IFAC Conference on Intelligent Control Systems and Signal Processing (ICONS 2003)*. [S.l.: s.n.], 2003. p. 113–118.

DODD, T. J.; MITCHINSON, B.; HARRISON, R. F. Sparce stochastic gradient descent learning in kernel mdels. In: *Proc. of the 2nd International Conference on Compuatational Intelligence, Robotics and Autonomous Systems*. [S.l.: s.n.], 2003.

DUTTWEILER, D. L.; KAILATH, T. An RKHS approach to detection and estimation theory: Some parameter estimation problems (Part V). *IEEE Trans. Inf. Theory*, v. 19, n. 1, p. 29–37, 1973.

ENGEL, Y.; MANNOR, S.; MEIR, R. Kernel recursive least squares. *IEEE Trans. Signal Process.*, v. 52, n. 8, p. 2275–2285, 2004.

GIANNAKINS, G. B.; SERPEDIN, E. T. A bibliography on nonlinear system identification. *IEEE Trans. on Signal Processing*, v. 81, p. 553–580, 2001.

GIROLAMI, M. Mercer kernel-based clustering in feature space. *IEEE Transactions on Neural Networks*, v. 13, p. 780–784, 2002.

GOLUB, G. H.; LOAN, C. F. V. *Matrix Computations*. [S.l.]: The Johns Hopkins University Press, 1996.

GRIMMETT, G.; STIRZAKER, D. *Probability and random processes*. [S.l.]: Oxford University Press, 2001.

HAUSSLER, R. *Convolution kernels on discrete structures*. [S.l.], 1999.

HAYKIN, S. *Adaptive Filter Theory*. 2nd. ed. New Jersey: Prentice-Hall, 1991.

HAYKIN, S. *Neural Networks, A Comprehensive Foundation*. [S.l.]: Macmillan College Publishing Company, Inc., 1994.

HAYKIN, S. *Adaptive Filter Theory, 4th ed.* [S.l.]: Prentice Hall, New-Jersey, 2002.

HERBRICH, R. *Learning Kernel Classifiers - Theory and Algorithms*. [S.l.]: The MIT Press, Cambridge, MA, 2002.

HONEINE, P.; RICHARD, C.; BERMUDEZ, J. C. M. On-line nonlinear sparse approximation of functions. In: *Proc. IEEE ISIT'07*. Nice, France: [s.n.], 2007. p. 956–960.

KIMELDORF, G.; WAHBA, G. Some results on Tchebycheffian spline functions. *J. Math. Anal. Appl.*, v. 33, p. 82–95, 1971.

KREYSZIG, E. *Introductory Functional Analysis with Applications*. [S.l.]: John Wiley, New York, 1989.

KUSS, M.; GRAEPEL, T. *The geometry of kernel canonical correlation analysis*. [S.l.], 2003.

LIMA, E. L. *Espaços Métricos*. [S.l.]: Instituto de Matemática Pura e Aplicada, Rio de Janeiro, BR, 1977.

LIU, W.; POKHAREL, P. P.; PRINCIPE, J. C. The kernel least-mean-squares algorithm. *IEEE Trans. Signal Process.*, v. 56, n. 2, p. 543–554, February 2008.

LIU, W.; PRINCIPE, J. C.; HAYKIN, S. *Kernel Adaptive Filtering*. [S.l.]: Jonh Wiley & Sons, 2010.

LUENBERGER, D. G. *Introduction Dynamic Systems: Theory, Models & Applications*. [S.l.]: Jonh Wiley & Sons, Inc., Stanford, CA, 1979.

MALLAT, S. *A wavelet tour of signal processing: The sparse way, 3rd. ed.* [S.l.]: Academic Press, 2008.

MALLAT, S.; ZHANG, Z. Matching pursuit with time-frenquency dictionaries. *IEEE Transactions on Signal Processing*, v. 41, p. 3397–3415, 1993.

MANDIC, D. P. A generalized normalized gradient descent algorithm. *IEEE Signal Processing Letters*, v. 2, p. 115–118, February 2004.

MANOLAKIS, D.; INGLE, V. K.; KOGON, S. M. *Statistical and Adptive Signal Processing: Spectral Estimation, Signal Modeling, Adaptive Filtering and Array Processing*. [S.l.]: Artech House Publishers, 2005.

MATHEWS, V. J.; SICURANZA, G. L. *Polynomial Signal Processing*. [S.l.]: Jonh Wiley & Sons,Inc, New York, 2000.

MILLER, K. S. *Multidimensional Gaussian Distributions*. [S.l.]: Jonh Wiley & Sons,Inc, New York, 1963.

MINKOFF, J. Comment: On the unnecessary assumption of statistical independence between reference signal and filter weights in feedforward adaptive systems. *IEEE Trans. Signal Process.*, v. 49, n. 5, p. 1109, May 2001.

NARENDRA, K. S.; PARTHASARATHY, K. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networs*, v. 1, n. 1, p. 3–27, March 1990.

OMURA, J.; KAILATH, T. *Some Useful Probability Distributions*. [S.l.], 1965.

PAPOULIS, A. *Probability Random Variables, and Stochastic Processes, 3rd ed*. [S.l.]: McGraw-Hill, 1991.

PARREIRA, W. et al. Steady-state behavior and design of the gaussian klms algorithm. In: *19th European Signal Processing Conference (EUSIPCO-2011), Barcelone, Espagne, 29/08/2011-02/09/2011*. http://www.eurasip.org/: EURASIP, 2011. p. 121–125.

PARREIRA, W. D. et al. Stochastic behavior analysis of the gaussian kernel least mean square algorithm. In: *Proc. IEEE ICASSP'11*. Prague, Czech Republic: [s.n.], 2011.

PARZEN, E. Extraction and delection problems and reproduncing kernel hilbert spaces. *Journal of the Society for Industrial and Applied Mathematics. Series A, On Control.*, v. 1, p. 35–62, 1962.

POKHAREL, P. P.; LIU, W.; PRINCIPE, J. C. Kernel LMS. In: *International Conference on Acoustics, Speech and Signal Processing - ICASSP*. [S.l.: s.n.], 2007. v. 3, p. III–1421–III–1424.

PÉREZ-CRUZ, F.; BOUSQUET, O. Kernel methods and their potential use in signal processing. *IEEE SIGNAL PROCESSING MAGAZINE*, v. 21, p. 57–65, 2004.

RICHARD, C.; BERMUDEZ, J. C. M.; HONEINE, P. Online prediction of time series data with kernels. *IEEE Trans. Signal Process.*, v. 57, n. 3, p. 1058 –1067, March 2009. ISSN 1053-587X.

SAYED, A. H. *Fundamentals of adaptive filtering*. Hoboken, NJ: John Wiley & Sons, 2003.

SCHETZEN, M. *The Volterra and Wiener theory of the nonlinear systems*. New York, NY: Wiley, 1980.

SCHÖLKOPF, B. *Support Vector Learning*. [S.l.]: Munich, Oldenboug Verlag, 1997.

SCHÖLKOPF, B.; SMOLA, A. J. *Learning with Kernels - Support Vector Machines, Regularization, Optimization, and Beyond*. [S.l.]: MIT-Press, London, 2002.

SCHÖLKOPF, B.; SMOLA, A. J.; MÜLLER, K. R. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, v. 10, p. 1299–1319, 1998.

SLAVAKIS, K.; THEODORIDIS, S. Sliding window generalized kernel affine projection algorithm using projection mappings. *EURASIP Journal on Advances in Signal Processing*, v. 2008, p. ID 735351, 2008.

SMOLA, A. J.; SCHÖLKOPF, B. *A tutorial on support vector regression*. [S.l.], 1998.

TROPP, J. A. Greed is good: Algorithmis results for sparce approximation. *IEEE Trans. on signal processing*, v. 52, n. 8, p. 2275–2285, August 2004.

VERT, J. V.; TSUDA, K.; SCHÖLKOPF, B. *Kernel Methods in Computational Biology*. [S.l.]: MIT Press, 2004.

VÖRÖS, J. Modeling and identification of wiener systems with two-segment nonlinearities. *IEEE Transactions on Control Systems Thecnology*, v. 11, n. 2, p. 253 – 257, March 2003.

WANG, J.-S.; HSU, Y.-L. Dynamic nonlinear system identification using a Wiener-type recurrent network with okid algorithm. *Journal of Information Science and Engineering*, v. 24, p. 891–905, 2008.

WATKINS, C. Dynamic alignment kernels. In: SMOLA, A. J. et al. (Ed.). *Advances in Large Margin Classifiers*. [S.l.]: MIT Press, Cambridge, MA, 2000.

WESTON, J. et al. Kernel dependency estimation. In: *16th Annual Conference on Neural Information Processing Systems (NIPS 2002), Cambridge, MA, USA*. [S.l.]: MIT Press, 2002. p. 873–880.

ANEXO A – REPRESENTAÇÃO DE DADOS USANDO KERNEL

Seja um conjunto $X = \{x_1, x_2, ..., x_n\}$ de n objetos que devem ser analisados, em que cada objeto $x_i \in X$ pode ser um conjunto de moléculas, um conjunto de imagens, sinais recebidos por uma antena, funções ou simplesmente elementos de um espaço vetorial qualquer. Para projetarmos um método de análise de dados a primeira questão que surge é: Como representar o conjunto de dados X para posterior processamento?

Uma vasta quantidade de métodos para análise de dados, fora os métodos baseados em *kernels*, tem uma resposta natural para esta pergunta: primeiramente definimos um representante para cada objeto e então representamos o conjunto pelo seu representante. Ou seja, definimos uma representação $\phi(x_i) \in \mathcal{X}$ para cada objeto $x_i \in X$, onde a representação pode estar (pertencer) por exemplo o espaço vetorial real, $\mathcal{X} = \mathbb{R}^q$. Assim o conjunto X de dados é então representado pelo conjunto das representações individuais de seus elementos, isto é, $\mathcal{X} = \{\phi(x_1),\ \phi(x_2),\ ...,\ \phi(x_n)\}$. Posteriormente, basta definir um algoritmo para processar os dados.

Os métodos baseados em *kernel* têm uma resposta um pouco diferente para a questão de representação de dados. Os dados não necessitam de uma representação individual, eles serão representados por um conjunto de pares ordenados de comparações. Isto significa que, em vez de usarmos uma aplicação $\phi : X \to \mathcal{X}$ para representar cada objeto $x_i \in X$ com $\phi(x_i) \in \mathcal{X}$, uma "função comparação" definida com imagem nos reais $\kappa : X \times X \to \mathbb{R}$ é usada, e o conjunto X de dados é representado por uma matriz[1] $n \times n$ em que cada componente é um par de medidas $K_{i,j} = \kappa(x_i, x_j)$. Vejamos um exemplo para compreendermos melhor esta forma de representação.

Exemplo A.1 (Representação de dados)*: Considere o conjunto X de se-*

[1]Todos os métodos baseados em *kernel* são projetados para processar tais matrizes.

quências numéricas, representado na Figura A.1. Uma forma simples de

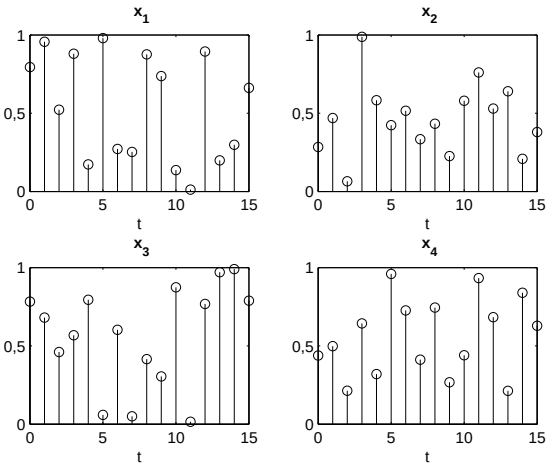


Figura A.1: Conjunto X de sinais

representar tais sinais seria utilizando a magnitude atingida em cada ponto como uma função do tempo, ou seja,

$$\begin{aligned}
\phi(x_1) &= [0,79; 0,96; 0,52; 0,88; 0,17; 0,98; 0,27; 0,25; 0,88; 0,74; 0,14; \\
&\quad 0,01; 0,89; 0,20; 0,30; 0,66]^\top, \\
\phi(x_2) &= [0,28; 0,47; 0,06; 0,99; 0,58; 0,42; 0,52; 0,33; 0,43; 0,23; 0,58; \\
&\quad 0,76; 0,53; 0,64; 0,21; 0,38]^\top, \\
\phi(x_3) &= [0,78; 0,68; 0,46; 0,57; 0,79; 0,06; 0,60; 0,05; 0,42; 0,31; 0,87; \\
&\quad 0,02; 0,77; 0,97; 0,99; 0,79]^\top, \\
\phi(x_4) &= [0,44; 0,50; 0,21; 0,64; 0,32; 0,96; 0,73; 0,41; 0,74; 0,27; 0,44; \\
&\quad 0,93; 0,68; 0,21; 0,83; 0,63]^\top
\end{aligned}$$

e

$$\mathcal{X} = \{\phi(x_1), \phi(x_2), \phi(x_3), \phi(x_4)\}.$$

Utilizando uma função kernel, podemos representar esses sinais usando a

seguinte matriz:

$$K = \begin{bmatrix} 6,42 & 3,87 & 4,79 & 5,04 \\ 3,87 & 4,21 & 4,25 & 4,37 \\ 4,79 & 4,25 & 6,78 & 4,76 \\ 5,04 & 4,37 & 4,76 & 5,93 \end{bmatrix}.$$

Note que K é uma matriz simétrica e os valores das sequências geradas estão no domínio dos reais. Em muitos casos são utilizadas matrizes simétricas com elementos dados pela seguinte equação:

$$K_{ij} = \kappa(x_i, x_j) = \frac{\langle \phi(x_i), \phi(x_j) \rangle}{\sqrt{\langle \phi(x_i), \phi(x_i) \rangle \langle \phi(x_j), \phi(x_j) \rangle}}.$$

Assim a matriz K pode ser reescrita como sendo:

$$K = \begin{bmatrix} 1,0000 & 0,7429 & 0,7280 & 0,8179 \\ 0,7429 & 1,0000 & 0,7968 & 0,8755 \\ 0,7280 & 0,7968 & 1,0000 & 0,7532 \\ 0,8179 & 0,8755 & 0,7532 & 1,0000 \end{bmatrix}.$$

É importante fazermos algumas considerações:

- A representação dos objetos não depende da sua natureza, eles podem ser imagens, moléculas, sequências ou objetos de qualquer outra natureza que a matriz que os representará será sempre quadrada e de valores reais[2].

- A dimensão da matriz usada para representar um conjunto de dados de n objetos é sempre $n \times n$, a ordem da matriz também independe da natureza ou complexidade dos objetos. Como vimos no Ex. A.1, cada padrão x_i foi representado por uma sequência de 16 valores, mas se fossem utilizados 32 ou 64 valores para representar a mesma quantidade

[2]Esta matriz é chamada matriz *Gram* que definiremos formalmente posteriormente.

de 4 padrões, a nova matriz de representação teria a mesma dimensão da matriz anterior, ou seja 4×4. Isso se torna um atrativo computacional, por exemplo, quando formos processar um conjunto pequeno de objetos muito complexos.

- Existem muitos casos em que a comparação dos objetos é uma tarefa mais fácil do que encontrar uma representação explícita para cada objeto que um dado algoritmo processará. Como exemplos temos alguns algoritmos de análise de dados tais como regressão quadrática ou rede neural, que requerem uma representação explícita para cada objeto x_i como um vetor $\phi(x_i) \in \mathbb{R}^q$.

Vimos no Exemplo A.1, uma descrição intuitiva de *kernel* como uma medida de similaridade entre os elementos apresentados. No Apêndice B, apresentamos uma definição formal para o *kernel*.

ANEXO B – KERNEL DEFINIDO POSITIVO E PROPRIEDADES

A maioria dos métodos baseados em *kernel* apenas processa matrizes quadradas, as quais são simétricas e definidas positivas. Iniciamos esta seção com algumas definições básicas para compreendermos o conceito de *kernel* definido positivo.

Definição B.1 (Matriz Gram)*: Dada uma função* $\kappa : X \times X \to \mathbb{R}$*(ou* $\mathbb{C}$*) e um conjunto de padrões* $x_1,\ x_2,\ ...,\ x_n \in X$*, a matriz* $[\boldsymbol{K}]_{n\times n}$ *com elementos*

$$[\boldsymbol{K}]_{ij} = \kappa(x_i, x_j) \tag{B.1}$$

é denominada **matriz Gram** *ou* **matriz de kernels** *de* κ *com relação aos elementos* $x_1,\ x_2,...,\ x_n$*.*

Definição B.2 (Matriz definida positiva)*: Uma matriz complexa* $K_{n\times n}$ *é dita definida positiva, quando,*

$$\sum_{j=1}^{n}\sum_{i=1}^{n} c_i\bar{c_j}[\boldsymbol{K}]_{ij} \geq 0 \tag{B.2}$$

onde $c_i, c_j \in \mathbb{R}$*(ou* $\mathbb{C}$*) e* $\bar{c_j}$ *denota o complexo conjugado*[1] *de* c_j*.*

Note que a matriz simétrica é definida positiva se e somente se todos os seus autovalores são positivos[2].

Definição B.3 (Kernel definido positivo)*: Uma função* $\kappa : X \times X \to \mathbb{R}$ *é chamada* **kernel definido positivo** *se para todo* $n > 0$ *e todo* $x_1,\ x_2,\ ...,\ x_n \in X$ *a matriz Gram* $\boldsymbol{K}$ *gerada a partir do par de medidas* $[\boldsymbol{K}]_{ij} = \kappa(x_i, x_j)$ *é definida positiva (Definição B.2).*

[1] Note que no caso de c_i e c_j reais isso não tem efeito.
[2] Se forem não-negativos a matriz será semi-definida positiva.

Em todo o nosso trabalho para simplificar, um *kernel definido positivo* está sempre sendo referido apenas como *kernel*.

B.1 Origem do termo kernel

Segundo Schölkopf em (SCHÖLKOPF; SMOLA, 2002), o termo *kernel* vem de um primeiro uso para um tipo de função em um corpo de operadores integrais tais como os estudados por Hilbert. Uma função κ é definida a partir do operador T_κ dado por:

$$(T_\kappa f)(x) = \int_X \kappa(x, x') f(x') dx' \tag{B.3}$$

e é denominado *kernel* de T_κ.

Na literatura, diferentes termos são utilizados para denominar os *kernels* definidos positivos, tais como *kernel* reprodutivo, *kernel* de Mercer, *kernel* admissível, *kernel* de suporte vetorial, *kernel* definido não negativo e função de covariância (SCHÖLKOPF; SMOLA, 2002).

Uma propriedade interessante do kernel definido positivo é: como se está livre para escolher pontos nos quais o *kernel* é avaliado, temos que, para toda escolha de pontos, o *kernel definido positivo* sempre induz a uma matriz *Gram* definida positiva.

Os *kernels* podem ser considerados como produto interno generalizado. Certamente, todo produto interno é um *kernel*, mas a linearidade nos argumentos, que é uma propriedade padrão de produto interno, não se aplica sobre os *kernels* gerais. Entretanto, uma outra propriedade de produtos internos, a desigualdade de Cauchy-Schwarz (LIMA, 1977), tem uma generalização para *kernels* (SCHÖLKOPF; SMOLA, 2002):

Proposição B.1 (Desigualdade de Cauchy-Schwarz para kernels)*: Se κ é*

um kernel definido positivo, dados $x, x' \in X$, então:

$$|\kappa(x, x')|^2 \leq \kappa(x, x).\kappa(x', x'). \tag{B.4}$$

A demonstração da Proposição B.1 pode ser vista em (SCHÖLKOPF; SMOLA, 2002, pag.31).

Nas próximas seções veremos outras propriedades dos *kernels*.

B.2 Kernel como produto interno

Seja ϕ um mapeamento no espaço das funções definidas de X em $\mathbb{R}$. Este espaço de funções será denotado por $\mathbb{R}^X := \{f : X \to \mathbb{R}\}$. Isto é:

$$\begin{aligned} \phi : \quad X \quad &\longrightarrow \mathbb{R}^X \\ x \quad &\longmapsto \kappa(., x) \end{aligned} \tag{B.5}$$

em que $\phi(x)$ denota a função que assumirá o valor $\kappa(x', x)$ para um dado $x' \in X$ fixo, isto é, $\phi(x)(.) = \kappa(., x)$. Desta maneira, cada padrão de teste x será transformado em uma função com domínio em X e imagem nos $\mathbb{R}$. Consequentemente, temos que um padrão pode ser representado pela medida de sua similaridade com todos os demais pontos do domínio X. Esta parece uma representação mais rica do que a representação convencional que usa o representante da classe para representar os elementos. Seguem então as etapas para construção do espaço das características associado a ϕ:

1.Transformar a imagem de $\phi(.)$ em um espaço vetorial;

2.definir um produto interno, isto é, uma forma bilinear estritamente definida positiva neste espaço vetorial e

3.mostrar que o produto interno satisfaz $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$.

Antes de definirmos um produto interno para as imagens de todos os padrões x pela $\phi(.)$, devemos definir o espaço vetorial[3]. Isso pode ser feito tomando as combinações lineares dos elementos de $\mathbb{R}^X$. Sendo assim, considere:

$$f(.) = \sum_{i=1}^{n} \alpha_i \kappa(., x_i) \tag{B.6}$$

em que $n \in \mathbb{N}$, $\alpha_i \in \mathbb{R}$ e $x_1, x_2, ..., x_n \in X$, quaisquer. Agora, é necessário definir o produto interno entre f e uma outra função qualquer do espaço,

$$g(.) = \sum_{j=1}^{n'} \beta_j \kappa(., x'_j) \tag{B.7}$$

em que, $n' \in \mathbb{N}$, $\beta_j \in \mathbb{R}$ e $x'_1, x'_2, ..., x'_{n'} \in X$. Note que para elementos $f,\ g \in \mathbb{R}^X$ e $\alpha_i,\ \beta_j \in \mathbb{R}$ o espaço $\mathbb{R}^X$ é fechado para soma e para o produto por escalar $\alpha \in \mathbb{R}$. Logo, é fácil ver que o espaço gerado pelas combinações lineares de elementos de $\mathbb{R}^X$ é um espaço vetorial. Agora podemos definir produto no espaço das combinações lineares de $\mathbb{R}^X$ como:

$$\langle f, g \rangle := \sum_{i=1}^{n} \sum_{j=1}^{n'} \alpha_i \beta_j \kappa(x_i, x'_j). \tag{B.8}$$

[3]Neste momento é importante deixar claro que o espaço das características é um espaço vetorial, mas não significa que $\phi(.)$ seja um vetor no sentido comum de elementos do um $\mathbb{R}^p$, mas sim no sentido mais amplo, como elemento de um espaço vetorial como apresentado em (KREYSZIG, 1989).

Esta expressão contém explicitamente os coeficientes da expansão, a qual não necessita ser única. Isto pode ser visto fazendo:

$$\begin{aligned}\langle f,g\rangle &= \sum_{j=1}^{n'}\beta_j\sum_{i=1}^{n}\alpha_i\kappa(x_i,x'_j) = \\ &= \sum_{j=1}^{n'}\beta_j\sum_{i=1}^{n}\alpha_i\kappa(x'_j,x_i) = \\ &= \sum_{j=1}^{n'}\beta_j f(x'_j). \end{aligned} \tag{B.9}$$

Assim, pela Equação (B.9), podemos dizer que a operação definida pela Equação (B.8) resulta em elementos do próprio espaço vetorial das funções $\mathbb{R}^X$, já que pode ser reescrita como combinação linear de elementos deste espaço. Vamos verificar então se a Equação (B.8) pode ser definida com produto interno no espaço $\mathbb{R}^X$.

Note que as igualdades na Equação (B.9) seguiram respectivamente da simetria do *kernel*, e da definição apresentada na Equação (B.6). Sendo assim, a Equação (B.9) não depende de uma expansão particular de f, pois similarmente podemos escrever:

$$\langle f,g\rangle = \sum_{i=1}^{n}\alpha_i g(x_i). \tag{B.10}$$

Considere agora $h \in \mathbb{R}^X$ dada por:

$$h(.) = \sum_{p=1}^{n''}\gamma_p\kappa(.,x''_p). \tag{B.11}$$

De maneira análoga à Equação (B.9), temos que:

$$\langle h,g\rangle = \sum_{j=1}^{n'}\beta_j h(x'_j). \tag{B.12}$$

Assim, somando as Equações (B.9) e (B.12):

$$\begin{aligned}\langle h,g\rangle + \langle f,g\rangle &= \sum_{j=1}^{n'} \beta_j h(x'_j) + \sum_{j=1}^{n'} \beta_j f(x'_j) \\ &= \sum_{j=1}^{n'} \beta_j \Big(h(x'_j) + f(x'_j)\Big) \\ &= \sum_{j=1}^{n'} \beta_j \Big((h+f)(x'_j))\Big) \\ &= \langle h+f,g\rangle . \end{aligned} \tag{B.13}$$

Então a operação, $\langle .,. \rangle$ definida pela Equação (B.8) satisfaz a primeira propriedade de produto interno (LIMA, 1977). Ainda utilizando a forma dada pela Equação (B.9) para $f_1 = \delta f$ com $\delta \in \mathbb{R}$ (ou $\mathbb{C}$) segue que:

$$\begin{aligned}\langle \delta f,g\rangle &= \langle f_1,g\rangle = \sum_{j=1}^{n'} \beta_j f_1(x'_j) \\ &= \sum_{j=1}^{n'} \beta_j \delta f(x'_j) = \delta \sum_{j=1}^{n'} \beta_j f(x'_j) \\ &= \delta \langle f,g\rangle . \end{aligned} \tag{B.14}$$

Desta forma está satisfeito o segundo axioma de produto interno (LIMA, 1977). Diretamente das Equações (B.9) e (B.10) segue o terceiro axioma (LIMA, 1977), que é:

$$\langle f,g\rangle = \langle g,f\rangle . \tag{B.15}$$

Pelo fato das funções aqui definidas terem suas respectivas imagens no corpo dos reais, $\mathbb{R}$, temos que o produto interno aqui definido é simétrico e bilinear. Assim, resta verificar apenas o quarto axioma de produto interno

(LIMA, 1977). Para isso devemos provar que:

$$\langle f, f \rangle \geq 0, \quad \forall f \;\; e \;\; \langle f, f \rangle = 0 \Leftrightarrow f \equiv 0 \tag{B.16}$$

Para verificar (B.16), substituímos g na Equação (B.8) por f dado pela Equação (B.6). Assim,

$$\langle f, f \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \kappa(x_i, x_j). \tag{B.17}$$

Note que a Equação (B.17) define um *kernel* de acordo com a Definição B.3 utilizando a matriz definida positiva. Desta forma, desde que o *kernel* seja definido positivo temos que:

$$\langle f, f \rangle \geq 0, \quad \forall f. \tag{B.18}$$

Para mostrar o último item dos axiomas que qualificarão a operação, $\langle ., . \rangle$, definida pela Equação (B.8) como o produto interno de um espaço de funções, serão necessárias algumas propriedades de ϕ que serão deduzidas a seguir:

1.Seguindo a definição dada pela Equação B.6,

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha_i \; k(x, x_i) = \\ &= \sum_{j=1}^{1} \sum_{i=1}^{n} 1 \; \alpha_i \; \kappa(x, x_i) = \\ &= \langle \kappa(., x), f \rangle, \end{aligned} \tag{B.19}$$

então, κ pode ser visto como um representante da avaliação de f em x.

2.Em particular, para uma $f(.) = \kappa(., x')$,

$$\langle \kappa(., x), \kappa(., x') \rangle = \sum_{i=1}^{1} \sum_{j=1}^{1} 1 \; 1 \; \kappa(x, x') = \kappa(x, x'). \tag{B.20}$$

3.Agora vamos provar uma generalização da desigualdade de Cauchy-Schwarz no espaço das funções $\mathbb{R}^X$.

Seja $\lambda \in \mathbb{R}$, assim usando (B.18) temos:

$$\langle \lambda f + g, \lambda f + g \rangle \geq 0.$$

Usando as propriedades dadas por (B.13),(B.14) e (B.15) obtemos:

$$\lambda^2 \langle f, f \rangle + 2\lambda \langle f, g \rangle + \langle g, g \rangle \geq 0. \tag{B.21}$$

Para manter a desigualdade da equação quadrática (B.21) o discriminante deve ser não-positivo, então:

$$\langle f, g \rangle^2 - \langle f, f \rangle \langle g, g \rangle \leq 0 \Leftrightarrow \langle f, g \rangle^2 \leq \langle f, f \rangle \langle g, g \rangle . \tag{B.22}$$

Devido as propriedades apresentadas em (B.19) e (B.20), os *kernels* definidos positivos são também chamados de *kernels* reprodutivos. A Equação (B.22) é uma generalização da Desigualdade de Cauchy-Schwarz no espaço vetorial $\mathbb{R}^X$. Agora segue das Equações (B.19) e (B.22) que:

$$|f(x)|^2 = |\langle \kappa(.,x), f \rangle|^2 \leq \kappa(x,x) \langle f, f \rangle . \tag{B.23}$$

Assim, se $\langle f, f \rangle = 0$, utilizando a Equação (B.23), segue que:

$$0 \leq |f(x)|^2 \leq \kappa(x,x). \langle f, f \rangle = 0, \ \forall x \in X \Leftrightarrow f = 0.$$

A recíproca é trivial.

Esta é a última propriedade necessária para caracterizar a operação $\langle .,. \rangle$ definida em (B.8) como produto interno (ver (LIMA, 1977)). É importante saber que esta formalização é possível para o caso de *kernels* complexos. Neste caso, os *kernels* dariam origem a um espaço de produto interno complexo.

Como consequência do que foi mostrado até o momento, podemos concluir que o *kernel* pode ser visto como um produto interno entre elementos do espaço das características gerado por (B.5). Desta forma, a característica reprodutiva dos *kernels*, vista na Equação (B.20), pode ser reescrita como:

$$\langle \phi(x), \phi(x') \rangle = \kappa(x, x') \tag{B.24}$$

Assim, a construção de um espaço com produto interno, definido pela Equação (B.24), é uma possibilidade imediata dos espaços característicos associados aos *kernels*. Tal espaço será denotado por $\mathcal{H}$.

Vamos agora enunciar o seguinte teorema:

Teorema B.1 (Kernel como produto interno)*: Para todo kernel κ definido sobre um espaço X, existe um espaço de Hilbert $\mathcal{H}$ e uma aplicação $\phi : X \rightarrow \mathcal{H}$ tal que:*

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle \qquad \forall\, x, x' \in X \tag{B.25}$$

onde $\langle ., . \rangle$ representa o produto interno entre os pontos do espaço de Hilbert $\mathcal{H}$.

A demonstração do Teorema B.1 pode ser vista em (SCHÖLKOPF; SMOLA, 2002). Até agora, iniciávamos com um *kernel* e construíamos um mapeamento das características. Considerando agora o sentido oposto, tendo um mapeamento ϕ em um espaço $\mathcal{X}$ com produto interno, podemos obter um *kernel* definido positivo? A resposta a esta pergunta é sim, tendo em vista que podemos obter um *kernel* definido positivo, definindo um *kernel* como $\kappa(x, x') := \langle \phi(x), \phi(x') \rangle$. Isto porque tomando escalares $c_i \in \mathbb{R}$, e elementos

$x_i \in X$, $i = 1, ..., n$, temos:

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \kappa(x_i, x_j) &= \sum_{i=1}^{n} c_i \sum_{j=1}^{n} c_j \left\langle \phi(x_i), \phi(x_j) \right\rangle \\
&= c_1 c_1 \left\langle \phi(x_1), \phi(x_1) \right\rangle + ... + c_1 c_n \left\langle \phi(x_1), \phi(x_n) \right\rangle + \dots \\
&\quad + c_n c_1 \left\langle \phi(x_n), \phi(x_1) \right\rangle + ... + c_n c_n \left\langle \phi(x_n), \phi(x_n) \right\rangle \\
&= \left\langle c_1\phi(x_1), c_1\phi(x_1) \right\rangle + ... + \left\langle c_1\phi(x_1), c_n\phi(x_n) \right\rangle + \dots \\
&\quad + \left\langle c_n\phi(x_n), c_1\phi(x_1) \right\rangle + ... + \left\langle c_n\phi(x_n), c_n\phi(x_n) \right\rangle \\
&= \left\langle \sum_{i=1}^{n} c_i\phi(x_i), \sum_{j=1}^{n} c_j\phi(x_j) \right\rangle \\
&= \left\| \sum_{i=1}^{n} c_i\phi(x_i) \right\|^2 \geq 0,
\end{aligned} \tag{B.26}$$

a desigualdade segue do primeiro axioma de Norma (LIMA, 1977).

Isto tem duas consequências. Primeiro, permite uma definição equivalente à de *kernel* definido positivo, definindo-o como uma função com a propriedade que garante a existência de um mapeamento ϕ no espaço de produto interno tal que a Equação(B.24) seja verdadeira. Segundo, permite a construção dos *kernels* a partir do mapeamento das características. A Equação (B.24) é a base para o "truque" do *kernel*.

B.2.1 O "truque" do kernel

Dado um algoritmo que seja formulado em termos de um *kernel* definido positivo, podemos construir um algoritmo alternativo trocando κ por um outro *kernel* definido positivo $\tilde{\kappa}$.

A aplicação mais conhecida do *"truque" do kernel* é quando $\kappa(.,.)$ é um produto interno no domínio da entrada. Neste caso o algoritmo original opera o produto interno entre $\{\phi(x_1), \phi(x_2), \dots, \phi(x_n)\}$ para algum $n \in \mathbb{N}$. O

algoritmo obtido pela troca de $\kappa(.,.)$ por $\tilde{\kappa}(.,.)$ será o mesmo, só que este passará a o operar vetores $\{\tilde{\phi}(x_1), \tilde{\phi}(x_2), \ldots, \tilde{\phi}(x_n)\}$ para algum $n \in \mathbb{N}$. Contudo, a aplicação do "truque" do *kernel* não se limita a esse caso, pois $\kappa(.,.)$ e $\tilde{\kappa}(.,.)$ pode ser ambos kernels não lineares.

B.3 Espaço Hilbertiano de kernel reprodutivo – RKHS

Na seção anterior, foi descrito como validar um espaço de funções como um espaço de características associado a um *kernel*, para isso verificávamos se esse espaço era um espaço vetorial dotado de produto interno.

Consideremos agora um espaço pré-Hilbert das funções definidas pela Equação (B.6), com produto interno definido pela Equação (B.8). Para transformá-lo em um espaço de Hilbert (sobre $\mathbb{R}$), devemos completá-lo com a norma corresponde a esse produto interno, $||f|| = \sqrt{\langle f, f \rangle}$, e adicionar os limites das séries que são convergentes nessa norma (KREYSZIG, 1989, Cap. 3). A partir das propriedades definidas pelas Equações (B.19) e (B.20), segundo (SCHÖLKOPF; SMOLA, 2002), este espaço é chamado de Espaço Hilbertiano de Kernel Reprodutivo (RKHS).

Em geral um RKHS pode ser definido como se segue:

Definição B.4 (Espaço Hilbertiano de kernel reprodutivo)*: Seja X um conjunto não-vazio (conjunto indexador) e $\mathcal{H}$ um espaço hilbertiano de funções $f : X \to \mathbb{R}$. Então $\mathcal{H}$ é chamado Espaço Hilbertiano de kernel reprodutivo dotado de norma induzida pelo produto interno, $||f|| = \sqrt{\langle f, f \rangle}$, se existe uma função $\kappa : X \times X \to \mathbb{R}$ com as seguintes propriedades.*

1.κ tem a propriedade de reprodução
$\langle \kappa(.,x), f \rangle = f(x)$ *para todo* $f \in \mathcal{H}$*;*
em particular,
$\langle \kappa(.,x), \kappa(.,x') \rangle = \kappa(x,x')$.

2.κ gera $\mathcal{H}$, isto é, $\mathcal{H} = \overline{span\,\{\kappa(x,.)|x \in X\}}$ onde a barra denota

o fecho do conjunto, de acordo com a definição apresentada em (KREYSZIG, 1989).

Em um nível mais abstrato, um RKHS pode ser definido como espaço hilbertiano das funções f em $\mathcal{H}$ tal que todos os funcionais $\mathcal{F}$ neste espaço (o mapeamento $\mathcal{F}_{x'} f \mapsto f(x')$, onde $x' \in X$), são contínuos e limitados (veja (KREYSZIG, 1989)). Pelo teorema da representação de Riez (KREYSZIG, 1989), sendo $\mathcal{F}$ limitado ele pode ser escrito como produto interno, em um espaço de Hilbert, $\mathcal{H}$. Assim para cada $x' \in X$:

$$\mathcal{F}_{x'}(f) = f(x') = \langle f, \kappa(.,x') \rangle_X . \tag{B.27}$$

Ainda, pelo teorema de Riesz, tem-se que $\kappa(.,x')$ é unicamente determinado pela norma de $\mathcal{F}$, ou seja, $||\kappa(.,x')|| = ||\mathcal{F}||$.

A respeito dos RKHS é possível verificar que eles são unicamente determinados por κ. Isto pode ser mostrado por contradição da seguinte forma. Assuma a existência de dois *kernels*, κ e κ', geradores de um mesmo RHKS de $\mathcal{H}$. Considerando a propriedade reprodutiva dos *kernels* dado pela Equação (B.19), com $\kappa'(x',.) \in \mathcal{H}$ temos:

$$\langle \kappa(x,.), \kappa'(x',.) \rangle_{\mathcal{H}} = \kappa(x,x')$$

por outro lado $\kappa' \in \mathcal{H}$ também gera o espaço, assim, pela propriedade de simetria do produto interno, tem-se:

$$\langle \kappa'(x',.), \kappa(x,.) \rangle_{\mathcal{H}} = \kappa'(x,x')$$

Como os *kernels* são também simétricos, o que podemos verificar pela análise das Equações (B.9) e (B.10), segue:

$$\kappa(x,x') = \kappa'(x,x')$$

isto é, os *kernels* são iguais.

ANEXO C – MATRIZ DE AUTOCORRELAÇÃO DO KERNEL É UMA MATRIZ DEFINIDA POSITIVA

Considere a matriz de autocorrelação $\boldsymbol{R}_{\kappa\kappa}$ de acordo com a definição (4.2), em que $\boldsymbol{u}(n)$ é i.i.d gaussiano. Para provar que $\boldsymbol{R}_{\kappa\kappa}$ é uma matriz definida positiva, vamos mostrar que todos os seus autovalores são positivos (GOLUB; LOAN, 1996).

Sejam $\lambda_i \in \mathbb{R}$ com $i = 1, \ldots, M$ os autovalores da matriz $\boldsymbol{R}_{\kappa\kappa}$. Tome agora os autovetores ν_i, com $i = 1, \ldots, M$, associados aos autovalores λ_i. Assim, temos:

$$\boldsymbol{R}_{\kappa\kappa}\nu_i = \lambda_i \nu_i, \quad i = 1, \ldots, M. \tag{C.1}$$

A partir das propriedades estruturais de $\boldsymbol{R}_{\kappa\kappa}$ (3.6), $[\boldsymbol{R}_{\kappa\kappa}]_{ii} = r_{\text{md}}$ e $[\boldsymbol{R}_{\kappa\kappa}]_{ij} = r_{\text{od}}$ com $i \neq j$, temos:

$$\boldsymbol{R}_{\kappa\kappa} = (r_{\text{md}} - r_{\text{od}})\boldsymbol{I} + r_{\text{od}}\,\mathbf{1}\,\mathbf{1}^{\top} \tag{C.2}$$

em que $\mathbf{1} = [1, 1, \ldots, 1]^{\top}_{M\times 1}$ e $\boldsymbol{I}$ é a matriz identidade de ordem $(M \times M)$.

Usando (C.2) podemos reescrever a Equação (C.1) como:

$$[(r_{\text{md}} - r_{\text{od}})\boldsymbol{I} + r_{\text{od}}\mathbf{1}\,\mathbf{1}^{\top}]\nu_i = \lambda_i \nu_1, \quad i = 1, \ldots, M. \tag{C.3}$$

Observe que $\nu_1 = \beta_1 \mathbf{1}$ com $\beta_1 \in \mathbb{R}^*$ é uma solução diferente da trivial ($\nu_i = \mathbf{0}$) para (C.1). Assim podemos obter o autovalor λ_1 ao qual ν_1 está associado, fazendo:

$$\begin{aligned}
&\beta_1(r_{\text{md}} - r_{\text{od}})\mathbf{1} + \beta_1 r_{\text{od}}\mathbf{1}\,\mathbf{1}^{\top}\,\mathbf{1} = \lambda_1\beta_1\,\mathbf{1} \\
&\beta_1(r_{\text{md}} - r_{\text{od}})\mathbf{1} + \beta_1 r_{\text{od}} M\,\mathbf{1} = \lambda_1\beta_1\,\mathbf{1} \\
&\lambda_1 = r_{\text{md}} + (M-1)r_{\text{od}}.
\end{aligned} \tag{C.4}$$

Note que λ_1 é positivo pois $r_{\text{od}} > 0$ (confira Equação (4.30)).

Os demais autovalores podem ser obtidos a partir dos ν_j, com $j \neq 1$,

autovetores ortogonais a ν_1. Assim, $\mathbf{1}^\top \nu_j = 0$ para todo $j \neq 1$, desta maneira:

$$(r_{\text{md}} - r_{\text{od}})\nu_j = \lambda_j \nu_j, \quad j = 2, \ldots, M \tag{C.5}$$

o que gera

$$\lambda_j = r_{\text{md}} - r_{\text{od}}, \quad j = 2, \ldots, M \tag{C.6}$$

o qual também é positivo já que $r_{\text{md}} > r_{\text{od}}$. Isto conclui a prova.

ANEXO D – ANÁLISE DO SINAL DAS COMPONENTES DA MATRIZ G

Nesta seção vamos analisar o sinal das componentes da matriz $\boldsymbol{G}$ definida pelas equações (3.38)-(3.39).

Por um lado, sabemos que as componentes $\eta^2\mu_3$, $\eta^2\mu_4$ e $\eta^2\mu_5$ são estritamente positivas. Por outro lado, as componentes $\eta^2\mu_2 - \eta r_{\text{od}}$ e $\frac{1}{2}(2\eta^2\mu_4 - \eta r_{\text{od}})$ podem ser tanto positivas quanto negativas dependendo do valor de η. A análise das componentes da diagonal de $\boldsymbol{G}$ necessita de uma atenção maior.

Inicialmente, considere a expressão $1 - 2\eta r_{\text{md}} + \eta^2\mu_1$. Esta expressão representa um polinômio de segundo grau com relação ao parâmetro η, o qual assume valor mínimo em $1 - \frac{r_{\text{md}}^2}{\mu_1}$. Usando (3.35) e (3.37), sabemos que $\mu_1 > r_{\text{md}}^2$. Isto implica que $1 - 2\eta r_{\text{md}} + \eta^2\mu_1 > 0$ para todo η.

Agora, vamos focar em $\frac{1}{2}(1 - 2\eta r_{\text{md}} + 2\eta^2\mu_3)$, o qual tem valor mínimo em $\frac{1}{2}(1 - \frac{r_{\text{md}}^2}{2\mu_3})$. Similarmente, sabemos que $\mu_3 > r_{\text{md}}^2$, o que significa que $\frac{1}{2}(1 - 2\eta r_{\text{md}} + 2\eta^2\mu_3) > 0$ para todo η.

Desta maneira podemos concluir que as entradas da matriz $\boldsymbol{G}$ são estritamente positivas, o que simplifica muito a análise das condições de estabilidade (3.42).

ANEXO E – ANÁLISE DA ESTABILIDADE NOS CASOS EM QUE $M = 1$ E $M = 2$

Vamos, inicialmente, derivar a condição para estabilidade do sistema (3.41) no caso $M = 1$. A matriz $\boldsymbol{G}$ reduz à componente para $1 - 2\eta r_{\text{md}} + \eta^2 \mu_1$. Isto implica diretamente que o sistema (3.41) é estável se

$$\eta < \frac{2\, r_{\text{md}}}{\mu_1}. \tag{E.1}$$

Considere, agora, o caso $M = 2$. A Expressão (3.42) define as seguintes desigualdades

$$(1 - 2\eta\, r_{\text{md}} + \eta^2\, \mu_1) + \eta^2\, \mu_3 + 2\,|\eta^2\, \mu_2 - \eta\, r_{\text{od}}| < 1, \tag{E.2a}$$

$$(1 - 2\,\eta\, r_{\text{md}} + 2\,\eta^2\, \mu_3) + 2\,|\eta^2\, \mu_2 - \eta\, r_{\text{od}}| < 1. \tag{E.2b}$$

Observe que o lado esquerdo de (E.2a) é maior que o lado esquerdo de (E.2b), para qualquer η, porque $\mu_1 \geq \mu_3$, como mostrado por (3.35). Procedendo da mesma maneira como fizemos para $M \geq 3$ na Seção 3.4.1, concluímos que o sistema (3.41) é estável para *kernels* positivos se as seguintes condições à seguir estiverem satisfeitas:

$$\begin{cases} \eta < \min\{\eta_1, \eta_2\}, & \text{para o caso } (i) \text{ na Tabela 3.1} \\ \eta < \eta_1, & \text{para os casos } (ii) \text{ e } (iv) \text{ na Tabela 3.1} \end{cases} \tag{E.3}$$

em que

$$\eta_1 = \frac{2\, r_{\text{md}} + 2\, r_{\text{od}}}{\mu_1 + 2\,\mu_2 + \mu_3} \qquad \eta_2 = \frac{\theta_1}{\theta_2} = \frac{2\, r_{\text{md}} - 2\, r_{\text{od}}}{\mu_1 - 2\,\mu_2 + \mu_3}. \tag{E.4}$$

Porém, quando o *kernel* não é de valores positivos e, ainda, μ_2 for

negativo, vimos na Seção 4.4 que a Restrição (3.49) é substituída por:

$$\begin{cases} \eta < \min\{\eta_1, \eta_2\}, & \text{para o caso } (i) \text{ na Tabela 4.1,} \\ \eta < \eta_2, & \text{para o caso } (iii) \text{ na Tabela 4.1.} \end{cases} \tag{E.5}$$

ANEXO F – EXEMPLO DE ESTIMAÇÃO DE SISTEMA COM ESTATÍSTICAS CONHECIDAS

Apresentamos este exemplo com o objetivo de ilustrar o procedimento de projeto. Utilizamos um sistema em que possamos obter as estatísticas, $E\{d^2(n)\}$ e $\boldsymbol{p}_{\kappa d}$, analiticamente.

F.1 Descrição do Sistema

O sinal de entrada utilizado é uma sequência de vetores estatisticamente independentes $\boldsymbol{u}(n) = [u_1(n) \;\; u_2(n)]^\top$ com as componentes correlacionadas de tal forma que $u_1(n) = 0,65\,u_2(n) + \eta_u(n)$, em que $\eta_u(n)$ é gaussiano tal que $\sigma^2_{u_1} = 1$. O sistema não-linear a ser estimado é (PARREIRA et al., 2011a):

$$\varphi(\boldsymbol{u}(n)) = \sum_{\ell=0}^{3} a^{\ell+1} \exp\left\{ \frac{-||\boldsymbol{u}(n-\ell) - \boldsymbol{b}_\ell||^2}{s_\ell^2} \right\} \tag{F.1}$$

com $a = 0,5$; $s_0 = 0,8063$; $s_1 = 0,9873$; $s_2 = 0,2756$; $s_3 = 0,7662$; $\boldsymbol{b}_0 = [-0,1454;\; -0,3862]^\top$, $\boldsymbol{b}_1 = [1,3162;\; -0,7965]^\top$, $\boldsymbol{b}_2 = [0,1354;\; 0,4178]^\top$ e $\boldsymbol{b}_3 = [0,8199;\; -0,8544]^\top$. A saída do sistema não-linear foi corrompida por um ruído i.i.d $z(n) \sim \mathcal{N}(0, \sigma_z^2)$, com $\sigma_z^2 = 10^{-6}$.

F.2 Cálculo das estatísticas da saída desejada – $d(n)$

Seja $d(n)$ a saída desejada para a entrada $\boldsymbol{u}(n)$ como descrito na Equação (F.1).

- A autocorrelação da saída desejada $d(n)$ pode ser calculada fazendo:

$$d^2(n) = \sum_{\ell=0}^{3} a^{2(\ell+1)} \exp\left\{\frac{-2||\boldsymbol{u}(n-\ell)-\boldsymbol{b}_\ell||^2}{s_\ell^2}\right\} + \sum_{\ell=0}^{3}\sum_{\substack{k=0\\k\neq\ell}}^{3} a^{(\ell+k+2)} \times \exp\left\{\frac{-||\boldsymbol{u}(n-\ell)-\boldsymbol{b}_\ell||^2}{s_\ell^2}\right\}\exp\left\{\frac{-||\boldsymbol{u}(n-k)-\boldsymbol{b}_k||^2}{s_k^2}\right\} + z(n)\sum_{\ell=0}^{3} a^{(\ell+1)} \exp\left\{\frac{-||\boldsymbol{u}(n-\ell)-\boldsymbol{b}_\ell||^2}{s_\ell^2}\right\} + z^2(n). \tag{F.2}$$

Tirando o valor esperado de (F.2) obtemos:

$$E\{d^2(n)\} = \sum_{\ell=0}^{3} a^{2(\ell+1)} E\left\{\exp\left[\frac{-2||\boldsymbol{u}(n-\ell)-\boldsymbol{b}_\ell||^2}{s_\ell^2}\right]\right\} + \sum_{\ell=0}^{3}\sum_{\substack{k=0\\k\neq\ell}}^{3} a^{(\ell+k+2)} \times E\left\{\exp\left[\frac{-||\boldsymbol{u}(n-\ell)-\boldsymbol{b}_\ell||^2}{s_\ell^2}\right]\exp\left[\frac{-||\boldsymbol{u}(n-k)-\boldsymbol{b}_k||^2}{s_k^2}\right]\right\} + E\{z(n)\}\sum_{\ell=0}^{3} a^{(\ell+1)} E\left\{\exp\left[\frac{-||\boldsymbol{u}(n-\ell)-\boldsymbol{b}_\ell||^2}{s_\ell^2}\right]\right\} + E\left\{z^2(n)\right\} \tag{F.3}$$

Podemos reescrever o primeiro valor esperado como,

$$E\left\{\exp\left[\frac{-2||\boldsymbol{u}(n-\ell)-\boldsymbol{b}_\ell||^2}{s_\ell^2}\right]\right\} = \exp\left[\frac{-2||\boldsymbol{b}_\ell||^2}{s_\ell^2}\right] E\left\{\exp\left[-\boldsymbol{u}^\top(n-\ell)\,\boldsymbol{W}\boldsymbol{u}(n-\ell) - \boldsymbol{r}^\top\boldsymbol{u}(n-\ell)\right]\right\} \tag{F.4}$$

em que $\boldsymbol{W} = \left(2/s_\ell^2\right)\boldsymbol{I}$ e $\boldsymbol{r} = \left(-4/s_\ell^2\right)\boldsymbol{b}_\ell$, sendo $\boldsymbol{I}$ igual a matriz identidade (2×2). Assim, temos que o primeiro valor esperado de (F.3) será

dado por (OMURA; KAILATH, 1965):

$$E\left\{\exp\left[\frac{-2||\boldsymbol{u}(n-\ell)-\boldsymbol{b}_\ell||^2}{s_\ell^2}\right]\right\} = \det(\boldsymbol{I}+2\,\boldsymbol{W}\boldsymbol{R}_{\boldsymbol{uu}})^{-1/2} \times \exp\left[\frac{-2||\boldsymbol{b}_\ell||^2}{s_\ell^2}+(1/2)\boldsymbol{r}^\top\boldsymbol{R}_{\boldsymbol{uu}}(\boldsymbol{I}+2\,\boldsymbol{W}\boldsymbol{R}_{\boldsymbol{uu}})^{-1}\boldsymbol{r}\right] \tag{F.5}$$

Para calcularmos o segundo valor esperado, vamos levar em consideração que $\boldsymbol{u}(n-\ell)$ e $\boldsymbol{u}(n-k)$ são i.i.d. e têm distribuição gaussianas. Portanto, é suficiente realizar os seguintes cálculos:

$$E\left\{\exp\left[\frac{-||\boldsymbol{u}(n-\ell)-\boldsymbol{b}_\ell||^2}{s_\ell^2}\right]\right\} = \exp\left[\frac{-||\boldsymbol{b}_\ell||^2}{s_\ell^2}\right]E\left\{\exp\left[-\boldsymbol{u}^\top(n-\ell)\,\boldsymbol{W}'\boldsymbol{u}(n-\ell)-\boldsymbol{r}'^\top\boldsymbol{u}(n-\ell)\right]\right\} \tag{F.6}$$

em que $\boldsymbol{W}' = \left(1/s_\ell^2\right)\boldsymbol{I}$ e $\boldsymbol{r}' = \left(-2/s_\ell^2\right)\boldsymbol{b}_\ell$. Por (OMURA; KAILATH, 1965), obtemos:

$$E\left\{\exp\left[\frac{-||\boldsymbol{u}(n-\ell)-\boldsymbol{b}_\ell||^2}{s_\ell^2}\right]\right\} = \det\left(\boldsymbol{I}+2\,\boldsymbol{W}'\boldsymbol{R}_{\boldsymbol{uu}}\right)^{-1/2} \times \exp\left[\frac{-||\boldsymbol{b}_\ell||^2}{s_\ell^2}+(1/2)\boldsymbol{r}'^\top\boldsymbol{R}_{\boldsymbol{uu}}\left(\boldsymbol{I}+2\,\boldsymbol{W}'\boldsymbol{R}_{\boldsymbol{uu}}\right)^{-1}\boldsymbol{r}'\right] \tag{F.7}$$

Usando os resultados apresentados em (F.5) e (F.7) podemos obter a autocorrelação de $d^2(n)$.

- A correlação cruzada entre $\kappa_\omega(n)$ e $d(n)$ pode ser obtida fazendo (PAPOULIS, 1991):

$$\boldsymbol{p}_{\kappa d} = E[\kappa_{\omega}(n)\, d(n)] = \begin{bmatrix} E[\kappa(\boldsymbol{u}(\omega_1), \boldsymbol{u}(n))\, d(n)] \\ E[\kappa(\boldsymbol{u}(\omega_2), \boldsymbol{u}(n))\, d(n)] \\ \vdots \\ E[\kappa(\boldsymbol{u}(\omega_M), \boldsymbol{u}(n))\, d(n)] \end{bmatrix} \tag{F.8}$$

Para i-ésima linha do vetor $\boldsymbol{p}_{\kappa d}$,

$$\begin{aligned}
[\boldsymbol{p}_{\kappa d}]_i &= \sum_{i=0}^{L} a_i E\left\{\kappa(\boldsymbol{u}(\omega_i), \boldsymbol{u}(n)) \exp\left[\frac{-||\boldsymbol{u}(n-i) - \boldsymbol{b}_i||^2}{s_i^2}\right]\right\} \\
&= \sum_{i=0}^{L} a_i \exp\left(-\frac{||\boldsymbol{b}_i||^2}{s_i^2}\right) \\
&\quad \times E\left\{\kappa(\boldsymbol{u}(\omega_i), \boldsymbol{u}(n)) \exp\left[\frac{-||\boldsymbol{u}(n-i)||^2 - 2\boldsymbol{b}_i^{\top}\boldsymbol{u}(n-i)}{s_i^2}\right]\right\}
\end{aligned}$$

Assim, se $\boldsymbol{u}(\omega_i) = \boldsymbol{u}(n-i) = \boldsymbol{x}$ e $\boldsymbol{u}(n) = \boldsymbol{x}'$ ou se $\boldsymbol{u}(n) = \boldsymbol{u}(n-i) = \boldsymbol{x}$ e $\boldsymbol{u}(\omega_i) = \boldsymbol{x}'$, temos,

$$\begin{aligned}
&E\left\{\kappa(\boldsymbol{x}, \boldsymbol{x}') \exp\left[\frac{-||\boldsymbol{x}||^2 - 2\boldsymbol{b}_i^{\top}\boldsymbol{x}}{s_i^2}\right]\right\} = \\
&\quad = E\left\{\exp\left[-\frac{||\boldsymbol{x} - \boldsymbol{x}'||^2}{2\xi^2} - \frac{||\boldsymbol{x}||^2 - 2\boldsymbol{b}_i^{\top}\boldsymbol{x}}{s_i^2}\right]\right\} \\
&\quad = E\left\{\exp\left[-\frac{1}{2}\left(\delta_i||\boldsymbol{x}||^2 - 2\gamma\boldsymbol{x}^{\top}\boldsymbol{x}' + \gamma||\boldsymbol{x}'||^2 + \boldsymbol{r}_i^{\top}\boldsymbol{x}\right)\right]\right\}.
\end{aligned} \tag{F.9}$$

Agora, vamos definir, $\boldsymbol{y}' = [\boldsymbol{x}^{\top}\ \boldsymbol{x}'^{\top}]^{\top}$,

$$\boldsymbol{O}' = \begin{bmatrix} \delta_i \boldsymbol{I} & -\gamma \boldsymbol{I} \\ -\gamma \boldsymbol{I} & \gamma \boldsymbol{I} \end{bmatrix}$$

$\boldsymbol{r}_i' = [\boldsymbol{r}_i^{\top}\ \boldsymbol{0}^{\top}]^{\top}$ em que $\boldsymbol{0}$ é o $(q \times 1)$ zero vetor. Assim,

$$\delta_i||\boldsymbol{x}||^2 - 2\gamma\boldsymbol{x}^{\top}\boldsymbol{x}' + \gamma||\boldsymbol{x}'||^2 + \boldsymbol{r}_i^{\top}\boldsymbol{x} = \boldsymbol{y}'^{\top}\boldsymbol{O}'\boldsymbol{y}' + \boldsymbol{r}_i'^{\top}\boldsymbol{y}'$$

Com um procedimento análogo ao realizado para calcular os momentos de $\kappa_\omega(n)$ usaremos os resultados de (OMURA; KAILATH, 1965) para calcular o valor esperado (F.9). Considere a função característica de $\boldsymbol{y}'^{\top}\boldsymbol{O}'\boldsymbol{y}' + \boldsymbol{r}_i'^{\top}$ dada por:

$$\Phi_z(\beta) = E\{e^{j\beta z}\} = \det(\boldsymbol{I}_2 - j\beta\boldsymbol{O}'\boldsymbol{R}_2)^{-1/2}. \tag{F.10}$$

Isto implica que,

$$E\left\{(\boldsymbol{u}(\omega_i), \boldsymbol{u}(n))\exp\left[\frac{-||\boldsymbol{u}(n-i) - \boldsymbol{b}_i||^2}{s_i^2}\right]\right\} = \det\Big(\boldsymbol{I}_2 + \boldsymbol{O}'\boldsymbol{R}_2\Big)^{-1/2}\exp\left[\frac{1}{8}\boldsymbol{r}_i'^{\top}\boldsymbol{R}_2(\boldsymbol{I}_2 + \boldsymbol{O}'\boldsymbol{R}_2)^{-1}\boldsymbol{r}_i'\right]. \tag{F.11}$$

Por outro lado, se $\boldsymbol{u}(\omega_i) \neq \boldsymbol{u}(n-i)$ para todo i, podemos definir $\boldsymbol{u}(n) = \boldsymbol{x}$, $\boldsymbol{u}(\omega_i) = \boldsymbol{x}'$ e $\boldsymbol{u}(n-i) = \boldsymbol{x}''$, e então o valor esperado é:

$$\begin{aligned}
&E\left\{\kappa_\omega(n)\exp\left[\frac{-||\boldsymbol{u}(n-i) - \boldsymbol{b}_i||^2}{s_i^2}\right]\right\} E\left\{(\boldsymbol{x}, \boldsymbol{x}')\exp\left[\frac{-||\boldsymbol{x}''||^2 - 2\boldsymbol{b}_i^{\top}\boldsymbol{x}}{s_i^2}\right]\right\} \\
&= E\left\{\kappa_\omega(n)\right\} E\left\{\exp\left[\frac{-||\boldsymbol{u}(n-i) - \boldsymbol{b}_i||^2}{s_i^2}\right]\right\} \\
&= \det\left(\boldsymbol{I} + \frac{1}{\xi^2}\boldsymbol{O}_2\boldsymbol{R}_2\right)^{-1/2}\det\left(\boldsymbol{I} + \frac{2}{s_i^2}\boldsymbol{\Sigma}_0\right)^{-1/2}\exp\left[\frac{2}{s_i^4}\boldsymbol{b}_i^{\top}\left(\boldsymbol{I} + \frac{2}{s_i^2}\boldsymbol{\Sigma}_0\right)^{-1}\boldsymbol{b}_i\right].
\end{aligned} \tag{F.12}$$

o qual conclui o cálculo de $\boldsymbol{p}_{\kappa d}$ em (F.8).

F.3 Condições de projeto e Resultados

Vamos considerar que MSE máximo para esta estimação ($J_{\max}$) deverá ser de $-16{,}8$ dB. Após alguns testes, selecionamos um nível de coerência $\varepsilon_0 = 10^{-3}$ avaliando o comportamento médio do comprimento dos di-

cionários. Como vamos usar o KLMS gaussiano, devemos selecionar um conjunto de parâmetros para o *kernel* gaussiano. Vamos testar ξ variando seus valores a partir de $0,1$ à 50, com incrementos de $0,01$. Para cada valor de ξ, um comprimento médio de dicionário M foi associado. O valor médio M foi obtido como a média de 1000 realizações utilizando a Regra 2.13.

Como devemos encontrar valores de (ξ, η) que satisfaçam $J_{ms}(\infty) < J_{\text{max}}$, iniciamos essa busca selecionando apenas os parâmetros ξ de tal maneira que $J_{\text{min}} < J_{\text{max}}$[1].

Os valores de $J_{\text{min}}(\xi)$ foram obtidos a partir de (3.8), e dos resultados apresentados na subseção anterior. Os valores de $J_{\text{min}}(\xi)$ para os quais $J_{\text{min}} < J_{\text{max}} = -16,8\,dB$ podem ser visualizados na Figura F.1. Os dados nesta figura foram interpolados para facilitar a visualização. A Figura F.2 exibe os pares (M, ξ) para o mesmo grupo de ξ selecionado.

A Tabela F.1 apresenta os valores para projeto para três valores de ξ que julgamos significativos. A partir desses valores, $\xi = 1,33$ é visto como um boa escolha para projeto. A Figura. F.2 mostra que $M = 2$ para $\xi = 1,33$. Assim, para o par $(M;\xi) = (2; 1,33)$, obtemos $J_{\text{min}} = -16,89$ dB, o qual é menor que J_{max}. Escolhendo um valor $\eta = 0,0601$ o qual é menor que η_{max}, obtemos $J_{ex}(\infty) = -33,63$ dB e $J_{ms}(\infty) = -16,80$ dB, com um número de iterações necessárias para convergência dado por $n_{\epsilon_{GK}} = 317$ iterações. A escolha do conjunto $(M;\xi;\eta) = (2; 1,33; 0,0601)$ é considerado uma boa escolha por satisfazer o critério de $J_{ms}(\infty) < J_{\text{max}}$, com a menor quantidade de iterações dentre as demais opções.

É claro que poderíamos selecionar os parâmetros de outra maneira utilizando os resultados apresentados aqui. As Figuras F.3 e F.4 mostram, respectivamente, os níveis de MSE e do MSE em excesso para os quais foi possível associar um ξ e um η de tal maneira que $J_{ms}(\infty) < J_{\text{max}}$. A Figura F.5

[1] Isto porque o KLMS tem a superfície de desempenho alterada pelo escolha do parâmetro do *kernel*.

apresenta o número de iterações necessárias para atingir esses níveis de MSE. As curvas das Figuras F.3 – F.5 tiveram seus pontos interpolados para facilitar a visualização. A partir desses resultados, podemos perceber que o parâmetro do *kernel* influencia na qualidade e na velocidade da estimação de $d(n)$. Portanto, poderíamos fazer outras escolhas de ξ baseando-se nos interesses secundários de projeto.

Finalmente, a Figura F.6 mostra o nível de precisão do modelo analítico derivado (3.16), comparando este com o comportamento médio sob 500 realizações (Monte Carlo), usando como referência os casos apresentados na Tabela F.1. A Figura F.7 compara o regime permanente obtido por (3.16) com o teórico predito pela Equação (3.58). A correspondência é claramente excelente.

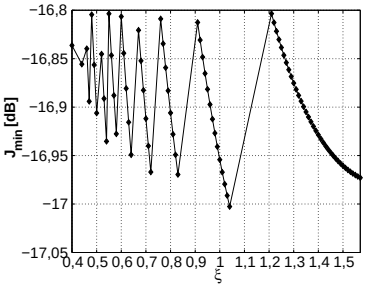


Figura F.1: Comportamento do MSE mínimo com $J_{\text{min}}(\xi) < J_{\text{max}}$.

98

Tabela F.1: Resumo dos resultados apresentados

ξ	M	η	J_{min} [dB]	$J_{ms}(\infty)$ [dB]	$J_{ex}(\infty)$ [dB]	$n_{\epsilon_{GK}}$
0,78	4	0,0298	-16,86	-16,82	-37,01	844
1,04	2	0,0618	-17,00	-16,90	-33,25	329
1,33	2	0,0601	-16,89	-16,80	-33,63	317

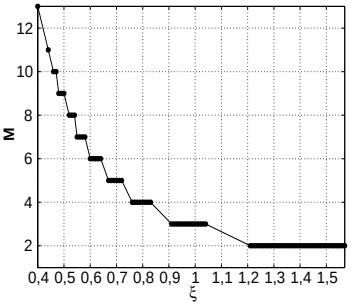


Figura F.2: Comportamento do comprimento médio do dicionário em que $J_{\text{min}}(\xi) < J_{\text{max}}$.

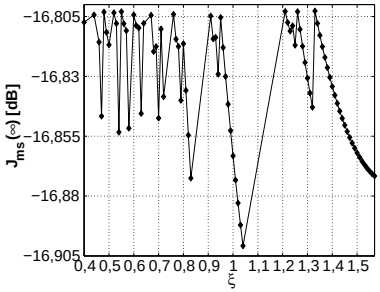


Figura F.3: Comportamento do regime permanente do MSE em que $J_{ms}(\infty) < J_{\text{max}}$.

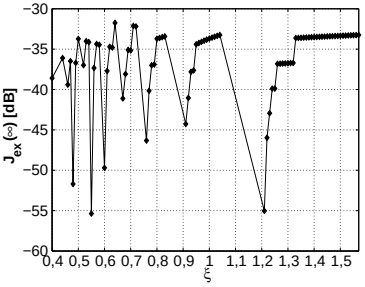


Figura F.4: Comportamento do regime permanente do MSE em excesso em que $J_{ms}(\infty) < J_{\text{max}}$.

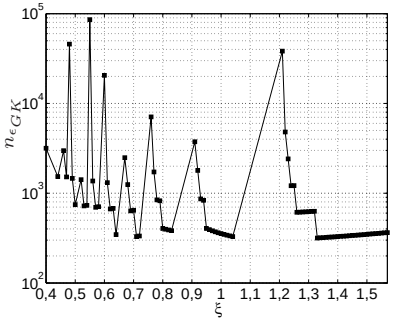


Figura F.5: Número de iterações necessárias para a convergência em que $J_{ms}(\infty) < J_{\max}$.

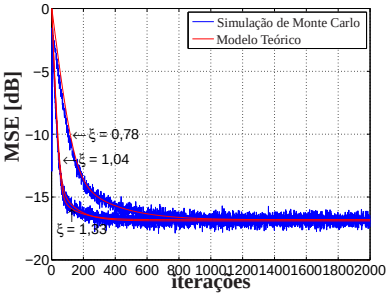
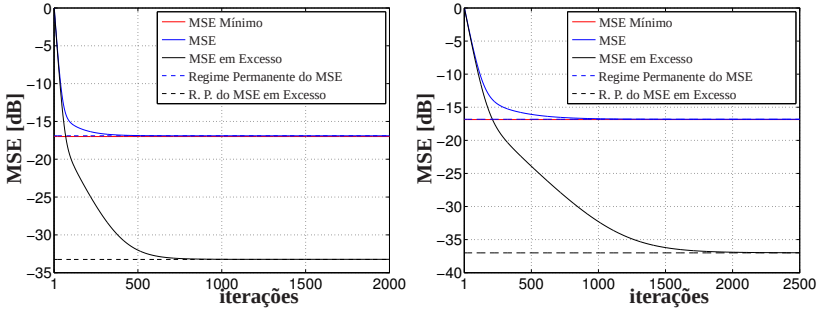
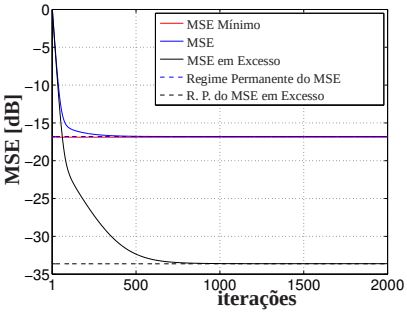


Figura F.6: Comportamento do KLMS para diferentes parâmetros do *kernel* gaussiano. A curva em vermelho representa o modelo teórico e a curva em azul representa as simulações de 500 realizações de Monte Carlo.



(a) Modelo teórico do algoritmo KLMS gaussiano com $\xi = 0,78$ e $M = 4$.

(b) Modelo teórico do algoritmo KLMS gaussiano com $\xi = 1,04$ e $M = 3$.



(c) Modelo teórico do algoritmo KLMS gaussiano com $\xi = 1,33$ e $M = 2$.

Figura F.7: Comportamento do modelo teórico do KLMS gaussiano. A curva contínua em vermelho representa o MSE mínimo, a curva contínua em azul representa MSE, a curva em preto representa o MSE em excesso e a curvas pontilhadas representa os seus respectivos regime permanente.

ANEXO G – ANÁLISE ESTATÍSTICA DO KERNEL POLINOMIAL DO SEGUNDO GRAU

No Capítulo 3, deduzimos as expressões para análise do comportamento estocástico do KLMS modificado para ordem finita operando em ambiente estacionário. Posteriormente, na Seção 4.3, fizemos um estudo dos momentos do *kernel* polinomial de ordem qualquer. Neste apêndice vamos particularizar esse estudo para os momentos de segunda e quarta ordem do *kernel* Polinomial do segundo grau. Assim, considere o *kernel* polinomial dado por:

$$\kappa(\boldsymbol{u}(n), \boldsymbol{u}(\omega_m)) = [a + \boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)]^2 \tag{G.1}$$

em que $a \geq 0$ e $\boldsymbol{u}(n) \in \mathbb{R}^2$ gaussianos, de média nula, i.i.d, tal que $E\{\boldsymbol{u}(n-i)\boldsymbol{u}(n-j) = 0\}$ para $i \neq j$. Para o cálculo desses momentos considere também que $E\{[\boldsymbol{u}(n)]_1^2\} = E\{[\boldsymbol{u}(n)]_2^2\} = \sigma_u^2$ em que $[\boldsymbol{u}(n)]_1$ denota a primeira componente do vetor $\boldsymbol{u}(n)$ e $[\boldsymbol{u}(n)]_2$ e denota a segunda componente do vetor $\boldsymbol{u}(n)$. As componentes do vetor $\boldsymbol{u}(n)$ podem ser correlacionadas com $E\{[\boldsymbol{u}(n)]_1[\boldsymbol{u}(n)]_2\} \neq 0$.

As expressões analíticas deduzidas aqui foram usadas para gerar o resultados apresentados no Capítulo 5.

G.1 Momento de segunda ordem

Nesta seção vamos apresentar as expressões analíticas das componentes da matriz de autocorrelação $\boldsymbol{R}_{\kappa\kappa}$, para o *kernel* polinomial do segundo grau.

G.1.1 Autocorrelação

Considerando $\boldsymbol{u}(n)$ como descrito anteriormente. A autocorrelação do *kernel* polinomial do segundo grau pode ser calculada fazendo:

$$
\begin{aligned}
E\left\{\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_m))^2\right\} =& E\left\{[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)]^4\right\} \\
=& \sum_{\ell=0}^{4} \mathsf{C}_{4-\ell}^{4}\, a^{\ell} \sum_{k_1+k_2=4-\ell} \mathsf{C}_{k_1,k_2}^{4-\ell}\, E\left\{[\boldsymbol{u}(\omega_m)]_1^{k_1}[\boldsymbol{u}(\omega_m)]_2^{k_2}\right\} \\
& \times E\left\{[\boldsymbol{u}(n)]_1^{k_1}[\boldsymbol{u}(n)]_2^{k_2}\right\} \\
=& \frac{4!}{4!0!}\, a^0 \left[2\,\frac{4!}{4!0!}\, s_{4,0}^2 + 2\,\frac{4!}{3!1!}\, s_{3,1}^2 + \frac{4!}{2!2!}\, s_{2,2}^2\right] \\
& + \frac{4!}{3!1!}\, a^1 \left[2\,\frac{3!}{3!0!}\, s_{3,0}^2 + 2\,\frac{3!}{2!1!}\, s_{2,1}^2\right] \\
& + \frac{4!}{2!2!}\, a^2 \left[2\,\frac{2!}{2!0!}\, s_{2,0}^2 + \frac{2!}{1!1!}\, s_{1,1}^2\right] \\
& + \frac{4!}{3!1!}\, a^3 \left[2\,\frac{1!}{1!0!}\, s_{1,0}^2\right] + \frac{4!}{0!4!}\, a^4
\end{aligned}
\tag{G.2}
$$

em que $1 \leq m \leq M$, $s_{\theta,\gamma} = E\{[\boldsymbol{u}(n)]_1^{\theta}[\boldsymbol{u}(n)]_2^{\gamma}\} = E\{[\boldsymbol{u}(n)]_1^{\gamma}[\boldsymbol{u}(n)]_2^{\theta}\} = E\left\{[\boldsymbol{u}(\omega_m)]_1^{\theta}[\boldsymbol{u}(\omega_m)]_2^{\gamma}\right\} = E\left\{[\boldsymbol{u}(\omega_m)]_1^{\gamma}[\boldsymbol{u}(\omega_m)]_2^{\theta}\right\}$.

Como vimos na Seção 4.3, os valores esperados de ordem ímpar, quando $\theta+\gamma$ é um número ímpar, são nulos. Assim, usando o teorema da fatoração dos momentos de variáveis aleatórias gaussianas (MILLER, 1963) obtemos os demais momentos representados por $s_{\theta,\gamma}$ necessários para obtermos (G.2), são eles:

$$
s_{4,0} = E\{[\boldsymbol{u}(n)]_1^4[\boldsymbol{u}(n)]_2^0\} = 3\,s_{2,0}^2 = 3\,\sigma_u^4\,, \tag{G.3}
$$

$$
s_{1,3} = E\{[\boldsymbol{u}(n)]_1^3[\boldsymbol{u}(n)]_2^1\} = 3\,s_{1,1}\,s_{0,2}^2 = 3\,\sigma_u^4\,s_{1,1}\,, \tag{G.4}
$$

e

$$s_{2,2} = E\{[\boldsymbol{u}(n)]_1^2[\boldsymbol{u}(n)]_2^2\} = \sigma_u^4 + 2\,s_{1,1}^2\,. \tag{G.5}$$

Logo, substituindo (G.3), (G.4) e (G.5) em (G.2), obtemos a autocorrelação do *kernel* polinomial do segundo grau:

$$\begin{aligned} E\left\{\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_m))^2\right\} =& 18\,\sigma_u^8 + 72\,\sigma_u^4\,s_{1,1}^2 + 6\,(\sigma_u^4 + 2\,s_{1,1})^2 \\ &+ 12\,a^2(\sigma_u^4 + s_{1,1}^2) + a^4. \end{aligned} \tag{G.6}$$

G.1.2 Correlação cruzada

A correlação cruzada para o *kernel* polinomial do segundo grau pode ser obtida fazendo:

$$\begin{aligned} &E\left\{\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_m))\,\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_{m'}))\right\} = \\ &= E\left\{[a + \boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)]^2[a + \boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{m'})]^2\right\} \\ &= \sum_{\ell=0}^{2}\sum_{p=0}^{2} \mathsf{C}_{2-\ell}^2\mathsf{C}_{2-p}^2 a^{\ell+p} \sum_{(k_1+k_2=2-\ell)} \mathsf{C}_{k_1,k_2}^{2-\ell} E\left\{[\boldsymbol{u}(\omega_m)]_1^{k_1}[\boldsymbol{u}(\omega_m)]_2^{k_2}\right\} \\ &\quad \times \sum_{(k_1'+k_2'=2-p)} \mathsf{C}_{k_1',k_2'}^{2-p} E\left\{[\boldsymbol{u}(\omega_{m'})]_1^{k_1'}[\boldsymbol{u}(\omega_{m'})]_2^{k_2'}\right\} E\left\{[\boldsymbol{u}(n)]_1^{k_1+k_1'}[\boldsymbol{u}(n)]_2^{k_2+k_2'}\right\} \\ &= a^0\Big[s_{0,2}^2 s_{0,4} + 2\,s_{0,2}s_{1,1}s_{1,3} + s_{0,2}^2 s_{2,2} + 2\,s_{1,1}s_{0,2}s_{1,3} + 4\,s_{1,1}^2 s_{2,2} \\ &\quad + 2\,s_{1,1}s_{0,2}s_{1,3} + s_{0,2}^2 s_{2,2} + 2\,s_{0,2}s_{1,1}s_{1,3} + s_{0,2}^2 s_{0,4}\Big] + 2\,a^2\Big[2s_{0,2}^2 + 2\,s_{1,1}^2\Big] + a^4. \end{aligned} \tag{G.7}$$

em que $1 \leq m, m' \leq M$ e $m \neq m'$. Aqui, omitimos os valores esperados de ordem ímpar pelo fato de serem nulos, como já visto na Seção 4.3.

Assim, substituindo (G.3), (G.4) e (G.5) em (G.7) obtemos a expressão para o cálculo da correlação cruzada do *kernel* polinomial do segundo grau:

$$E\{\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_m))\,\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_{m'}))\} = 6\sigma_u^8 + 24\sigma_u^4 s_{1,1}^2 + 2(\sigma_u^4 + 2s_{1,1}^2)^2 + 4a^2(\sigma_u^4 + s_{1,1}^2) + a^4. \tag{G.8}$$

G.2 Momentos de quarta ordem

No Capítulo 3, Subseção 3.3.4, vimos a necessidade do conhecimento dos momentos de quarta ordem para realizarmos o estudo do comportamento estocástico da matriz de correlação do vetor de erro nos coeficientes. Posteriormente, usamos essas expressões também para estudar a estabilidade do algoritmo KLMS com dimensão fixa e finita, Seções 3.4 e 4.3. Este apêndice tem a finalidade de exibir as expressões analíticas usadas para produzir os resultados apresentados na Seção 5, quando o *kernel* polinomial é do segundo grau e $\boldsymbol{u}(n) \in \mathbb{R}^2$.

G.2.1 Cálculo de μ_1

O momento de quarta ordem, dado por μ_1 para o *kernel* polinomial do segundo grau, pode ser obtido a partir de uma particularização da Equação (4.26), fazendo:

$$\begin{aligned}\mu_1 &= E\left\{\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_m))^4\right\} = E\left\{[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)]^8\right\}\\ &= a^8 + 14a^6 E\left\{\left(\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)\right)^2\right\} + 70a^4 E\left\{\left(\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)\right)^4\right\}\\ &\quad + 14a^2 E\left\{\left(\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)\right)^6\right\} + E\left\{\left(\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)\right)^8\right\}.\end{aligned} \tag{G.9}$$

em que $1 \leq m \leq M$. Os valores esperados de ordem ímpar foram omitidos por serem nulos, como visto na Seção 4.3.

Para efeito didático, vamos calcular cada valor esperado de (G.9) separadamente:

Para o primeiro valor esperado temos:

$$\begin{aligned} E\left\{\left(\boldsymbol{u}^{\top}(n)\boldsymbol{u}(\omega_m)\right)^2\right\} &= E\left\{E\left\{\left(\boldsymbol{u}^{\top}(n)\boldsymbol{u}(\omega_m)\right)^2\right\}\Big|\boldsymbol{u}(n)\right\} \\ &= s_{0,2}\,E\left\{[\boldsymbol{u}(n)]_1^2+[\boldsymbol{u}(n)]_2^2\right\}+2\,s_{1,1}\,E\left\{[\boldsymbol{u}(n)]_1[\boldsymbol{u}(n)]_2\right\} \\ &= 2\,s_{0,2}^2+2\,s_{1,1}^2 \\ &= 2\,\sigma_u^4+2\,s_{1,1}^2. \end{aligned} \tag{G.10}$$

Para o segundo valor esperado fazemos:

$$\begin{aligned} E\left\{\left(\boldsymbol{u}^{\top}(n)\boldsymbol{u}(\omega_m)\right)^4\right\} &= E\left\{E\left\{\left(\boldsymbol{u}^{\top}(n)\boldsymbol{u}(\omega_m)\right)^4\right\}\Big|\boldsymbol{u}(n)\right\} \\ &= 3\,s_{0,2}^2\,E\left\{[\boldsymbol{u}(n)]_1^4+[\boldsymbol{u}(n)]_2^4\right\} \\ &\quad +12\,s_{0,2}\,s_{1,1}E\left\{[\boldsymbol{u}(n)]_1^3[\boldsymbol{u}(n)]_2+[\boldsymbol{u}(n)]_1[\boldsymbol{u}(n)]_2^2\right\} \\ &\quad +6\,(s_{0,2}^2+s_{1,1}^2)E\left\{[\boldsymbol{u}(n)]_1^2[\boldsymbol{u}(n)]_2^2\right\} \\ &= 18\,s_{0,2}^4+72\,s_{0,2}^2\,s_{1,1}^2+6\left(s_{0,2}^2+2s_{1,1}^2\right)^2 \\ &= 18\,\sigma_u^8+72\,\sigma_u^4\,s_{1,1}^2+6\left(\sigma_u^4+2s_{1,1}^2\right)^2. \end{aligned} \tag{G.11}$$

Para o terceiro valor esperado:

$$\begin{aligned} E\left\{\left(\boldsymbol{u}^{\top}(n)\boldsymbol{u}(\omega_m)\right)^6\right\} &= E\left\{E\left\{\left(\boldsymbol{u}^{\top}(n)\boldsymbol{u}(\omega_m)\right)^6\right\}\Big|\boldsymbol{u}(n)\right\} \\ &= E\left\{\sum_{\ell=0}^{6}\mathsf{C}_{6-\ell}^{6}[\boldsymbol{u}(n)]_1^{6-\ell}[\boldsymbol{u}(n)]_2^{\ell}E\left\{[\boldsymbol{u}(\omega_m)]_1^{6-\ell}[\boldsymbol{u}(\omega_m)]_2^{\ell}\right\}\right\} \\ &= 30\,s_{0,2}^2\,s_{0,6}+180\,s_{0,2}^2\,s_{1,1}\,s_{1,5}+s_{2,4}(90\,s_{0,2}^3+360\,s_{0,2}\,s_{1,1}^2) \\ &\quad +60\,s_{1,1}(3\,s_{0,2}^2+2\,s_{1,1}^2)\,s_{3,3} \end{aligned} \tag{G.12}$$

em que $s_{0,6}$, $s_{1,5}$, $s_{2,4}$ e $s_{3,3}$ são momentos de sexta ordem de variáveis aleatórias gaussianas e podem ser calculadas da seguinte maneira:

$$s_{0,6} = E\{[\boldsymbol{u}(n)]_1^0[\boldsymbol{u}(n)]_2^6\} = 15\, s_{0,2}^3 = 15\, \sigma_u^6 . \tag{G.13}$$

Para calcular os demais momentos de sexta ordem de variáveis aleatórias gaussianas, podemos recorrer à seguinte técnica:

Fazemos $[\boldsymbol{u}(n)]_1 = \alpha_1 e_1$ e $[\boldsymbol{u}(n)]_2 = \beta_1 e_1 + \beta_2 e_2$, de tal maneira que,

$$E\{e_i e_j\} = \begin{cases} 1, & \text{se } i = j \\ 0, & \text{caso contrário.} \end{cases} \tag{G.14}$$

Agora, calculando os momentos de segunda ordem de $\boldsymbol{u}(n)$ podemos verificar que:

$$\alpha_1 = \sigma_u,\ \beta_1 = s_{1,1}/\sigma_u \text{ e } \beta_2 = (1/\sigma_u)\sqrt{\sigma_u^4 - s_{1,1}^2}.$$

Assim, os demais momentos de sexta ordem serão dados por:

$$s_{1,5} = E\left\{(\alpha_1 e_1)^5(\beta_1 e_1 + \beta_2 e_2)\right\} = 15\, s_{0,2}^2 s_{1,1} = 15\sigma_u^4 s_{1,1}\,, \tag{G.15}$$

$$s_{2,4} = 3s_{0,2}^3 + 12 s_{0,2}^2 s_{1,1}^2 = 3\sigma_{\boldsymbol{u}}^6 + 12\sigma_u^4 s_{1,1}^2 \tag{G.16}$$

e

$$s_{3,3} = 9\, s_{1,1}\, s_{0,2}^2 + 6\, s_{1,1}^3 = 9\sigma_u^4 s_{1,1} + 6\, s_{1,1}^3 . \tag{G.17}$$

Isto finaliza o cálculo do valor esperado $E\left\{\left(\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)\right)^6\right\}$.

Agora para o quarto valor esperado da Equação (G.9), fazemos:

$$\begin{aligned} E\left\{\left(\boldsymbol{u}^{\top}(n)\boldsymbol{u}(\omega_m)\right)^8\right\} &= E\left\{E\left\{\left(\boldsymbol{u}^{\top}(n)\boldsymbol{u}(\omega_m)\right)^8\right\}\middle|\boldsymbol{u}(n)\right\} \\ &= E\left\{\sum_{\ell=0}^{8} \mathrm{C}_{8-\ell}^{8}[\boldsymbol{u}(n)]_1^{8-\ell}[\boldsymbol{u}(n)]_2^{\ell} \right. \\ &\quad \left. \times E\left\{[\boldsymbol{u}(\omega_m)]_1^{8-\ell}[\boldsymbol{u}(\omega_m)]_2^{\ell}\right\}\right\} \\ &= s_{0,8}^2 + 16\, s_{1,7}^2 + 28\, s_{2,6} + 112\, s_{3,5} + 70\, s_{4,4}\,. \end{aligned} \tag{G.18}$$

Para obter o valor do Momento (G.18) é necessário o cálculo dos momentos de oitava ordem, $s_{\theta,\gamma}$ com $\theta+\gamma=8$. Usando um procedimento análogo ao realizado anteriormente obtemos:

$$s_{0,8} = E\{[\boldsymbol{u}(n)]_1^0[\boldsymbol{u}(n)]_2^8\} = 105\, s_{0,2}^4 = 105\, \sigma_u^8\,, \tag{G.19}$$

$$s_{1,7} = E\{[\boldsymbol{u}(n)]_1^1[\boldsymbol{u}(n)]_2^7\} = 105\, s_{0,2}^3\, s_{1,1} = 105\, \sigma_u^6\, s_{1,1}\,, \tag{G.20}$$

$$s_{2,6} = E\{[\boldsymbol{u}(n)]_1^2[\boldsymbol{u}(n)]_2^6\} = 15\, s_{0,2}^4 + 90\, s_{0,2}^2\, s_{1,1}^2 = 15\, \sigma_u^8 + 90\, \sigma_u^4\, s_{1,1}^2\,, \tag{G.21}$$

$$s_{3,5} = E\{[\boldsymbol{u}(n)]_1^3[\boldsymbol{u}(n)]_2^5\} = 60\, s_{0,2}\, s_{1,1}^3 + 45\, s_{0,2}^3\, s_{1,1} = 60\, \sigma_u^2\, s_{1,1}^3 + 45\, \sigma_u^6\, s_{1,1}\,, \tag{G.22}$$

e

$$\begin{aligned} s_{4,4} &= E\{[\boldsymbol{u}(n)]_1^4[\boldsymbol{u}(n)]_2^4\} \\ &= 24\, s_{1,1}^4 + 72\, s_{1,1}^2\, s_{0,2}^2 + 9\, s_{0,2}^4 = 24\, s_{1,1}^4 + 72\, s_{1,1}^2\, \sigma_u^4 + 9\, \sigma_u^8\,. \end{aligned} \tag{G.23}$$

O cálculo desse valor esperado finaliza o cálculo do momento de quarta ordem μ_1.

G.2.2 Cálculo de μ_2

Para o cálculo do momento de quarta ordem μ_2, o procedimento é análogo ao realizado na subseção anterior. Dessa maneira temos:

$$\begin{aligned}
\mu_2 =& E\left\{\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_m))^3\kappa(\boldsymbol{u}(n),\boldsymbol{u}(\omega_{m'}))\right\}\\
=& E\left\{[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)]^6[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{m'})]^2\right\}\\
=& \sum_{\ell_1=0}^{6}\sum_{\ell_2=0}^{2}\mathsf{C}_{6-\ell_1}^{6}\mathsf{C}_{2-\ell_2}^{2}a^{\ell_1+\ell_2}\sum_{(k_1+k_2=6-\ell_1)}\sum_{(k_1'+k_2'=2-\ell_2)}\mathsf{C}_{k_1,k_2}^{6-\ell_1}\mathsf{C}_{k_1',k_2'}^{2-\ell_2}\\
&\times E\left\{[\boldsymbol{u}(\omega_m)]_1^{k_1}[\boldsymbol{u}(\omega_m)]_2^{k_2}\right\}E\left\{[\boldsymbol{u}(\omega_{m'})]_1^{k_1'}[\boldsymbol{u}(\omega_{m'})]_2^{k_2'}\right\}E\left\{[\boldsymbol{u}]_1^{k_1+k_1'}[\boldsymbol{u}]_2^{k_2+k_2'}\right\}\\
=& \sum_{\ell_1=0}^{6}\sum_{\ell_2=0}^{2}\mathsf{C}_{6-\ell_1}^{6}\mathsf{C}_{2-\ell_2}^{2}a^{\ell_1+\ell_2}S(6-\ell_1,2-\ell_2)
\end{aligned}\tag{G.24}$$

em que $1\leq m,m'\leq M$ e $m\neq m'$ e

$$\begin{aligned}
S(6-\ell_1,2-\ell_2) =& \sum_{(k_1+k_2=6-\ell_1)}\sum_{(k_1'+k_2'=2-\ell_2)}\mathsf{C}_{k_1,k_2}^{6-\ell_1}\mathsf{C}_{k_1',k_2'}^{2-\ell_2}E\left\{[\boldsymbol{u}(\omega_m)]_1^{k_1}[\boldsymbol{u}(\omega_m)]_2^{k_2}\right\}\\
&\times E\left\{[\boldsymbol{u}(\omega_{m'})]_1^{k_1'}[\boldsymbol{u}(\omega_{m'})]_2^{k_2'}\right\}E\left\{[\boldsymbol{u}(n)]_1^{k_1+k_1'}[\boldsymbol{u}(n)]_2^{k_2+k_2'}\right\}.
\end{aligned}$$

Assim, devemos calcular $S(6-\ell_1,2-\ell_2)$, com $0\leq\ell_1\leq 6$ e $0\leq\ell_2\leq 2$. Vamos omitir os casos em que $\ell_1+\ell_2$ são ímpares por corresponderem a momentos de ordem ímpar de varáveis aleatórias gaussianas (ver Seção 4.3). Para efeito didático, calcularemos cada uma das somas $S(i,j)$ de μ_2 separadamente.

Com $\ell_1 = \ell_2 = 0$ temos:

$$\begin{aligned} S(6,2) =& 30\, s_{0,2}^2 (s_{0,8} + s_{2,6}) + 60\, s_{2,0}\, s_{1,1}\, s_{1,7} + 180\, s_{0,2}^3 s_{1,1} (s_{1,7} + s_{3,5}) \\ &+ 360\, s_{0,4}\, s_{1,1}\, s_{2,6} + (90\, s_{0,8} + 360\, s_{0,2}^2 s_{1,1}^2)(s_{2,6} + s_{4,4}) \\ &+ s_{3,5} (180\, s_{0,2}^3\, s_{1,1} + 720\, s_{0,2}\, s_{1,1}^3) + s_{4,4} (360\, s_{0,2}^2 s_{1,1}^2 + 240\, s_{1,1}^4). \end{aligned} \tag{G.25}$$

Com $\ell_1 = 0$ e $\ell_2 = 2$ temos:

$$S(6,0) = 450\, s_{0,2}^6 + 2700 s_{0,2}^4\, s_{1,1}^2 + 30 (3\, s_{0,2}^3 + s_{0,2}\, s_{1,1}^2)^2 + 20 (9\, s_{0,2}^2\, s_{1,1} + 6\, s_{1,1}^3)^2 \tag{G.26}$$

Com $\ell_1 = 2$ e $\ell_2 = 0$ temos:

$$\begin{aligned} S(4,2) =& 6\, s_{0,2}^3 (s_{0,6} + s_{4,2}) + 12\, s_{0,2}^2\, s_{1,1}\, s_{0,5} + 24\, s_{0,2}^2\, s_{1,1} (s_{1,5} + s_{3,3}) \\ &+ 12 (s_{0,2}^3 + s_{0,2}\, s_{1,1}^2)\, s_{4,2} + 48\, s_{0,2}\, s_{1,1}^2\, s_{2,4} + 12\, s_{3,3}\, s_{1,1}\, (s_{0,2}^2 + s_{1,1}^2). \end{aligned} \tag{G.27}$$

Com $\ell_1 = 2$ e $\ell_2 = 2$ temos:

$$S(4,0) = 18\, s_{0,2}^4 + 72 (s_{0,2}\, s_{1,1})^2 + 6 (s_{0,2}^2 + 2\, s_{1,1}^2)^2. \tag{G.28}$$

Com $\ell_1 = 4$ e $\ell_2 = 0$ temos:

$$S(2,2) = 8\, s_{0,2}^4 + 8\, s_{1,1}^4 + 32\, s_{0,2}^2\, s_{1,1}^2. \tag{G.29}$$

Com $\ell_1 = 4$ e $\ell_2 = 2$ temos:

$$S(2,0) = 2 s_{0,2}^2 + 2\, s_{1,1}^2. \tag{G.30}$$

Com $\ell_1 = 6$ e $\ell_2 = 0$ temos:

$$S(0,2) = 2 s_{0,2}^2 + 2\, s_{1,1}^2. \tag{G.31}$$

Com $\ell_1 = 6$ e $\ell_2 = 2$ temos:

$$S(0,0) = 1\,. \tag{G.32}$$

Como todos os momentos $s_{\theta,\gamma}$ já foram calculados anteriormente, Subseção G.2.1, completamos a avaliação do momento de quarta ordem μ_2.

G.2.3 Cálculo de μ_3

Para calcular o momento de quarta ordem μ_3, usaremos um procedimento análogo ao realizado na Subseção G.2.2. Dessa maneira temos:

$$\begin{aligned}
\mu_3 =& E\Big\{[a + \boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)]^4[a + \boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{m'})]^4\Big\} \\
=& \sum_{\ell_1=0}^{4}\sum_{\ell_2=0}^{4} \mathsf{C}_{4-\ell_1}^{4}\mathsf{C}_{4-\ell_2}^{4} a^{\ell_1+\ell_2} \sum_{(k_1+k_2=4-\ell_1)}\ \sum_{(k_1'+k_2'=4-\ell_2)} \mathsf{C}_{k_1,k_2}^{4-\ell_1}\mathsf{C}_{k_1',k_2'}^{4-\ell_2} \\
& \times E\Big\{[\boldsymbol{u}(\omega_m)]_1^{k_1}[\boldsymbol{u}(\omega_m)]_2^{k_2}\Big\} E\Big\{[\boldsymbol{u}(\omega_{m'})]_1^{k_1'}[\boldsymbol{u}(\omega_{m'})]_2^{k_2'}\Big\} \\
& \times E\Big\{[\boldsymbol{u}(n)]_1^{k_1+k_1'}[\boldsymbol{u}(n)]_2^{k_2+k_2'}\Big\} \\
=& \sum_{\ell_1=0}^{4}\sum_{\ell_2=0}^{4} \mathsf{C}_{4-\ell_1}^{4}\mathsf{C}_{4-\ell_2}^{4} a^{\ell_1+\ell_2} S(4-\ell_1, 4-\ell_2)\,.
\end{aligned} \tag{G.33}$$

em que $1 \leq m, m' \leq M$ e $m \neq m'$.

Ao expandir a soma de (G.33), é possível verificar que apenas $S(4,4)$ ainda não foi calculado, todos os demais já foram obtidos na Subseção G.2.2. Assim, para $S(4,4)$, fazemos:

$$\begin{aligned}
S(4,4) =& 18\, s_{0,2}^4 (s_{8,0} + s_{4,4}) + 288\, s_{0,2}^2 s_{1,1} (s_{2,6} + s_{4,4}) + 36 (s_{0,2}^2 + s_{1,1}^2)^2 s_{5,3}) \\
& + 144\, s_{0,2}^2 s_{1,1} (s_{7,1} + 72\, s_{0,2}^2 s_{2,6} (s_{0,2}^4 + s_{1,1}^2) \\
& + 288\, s_{0,2}^2 s_{1,1} s_{3,5} (s_{0,2}^2 + s_{1,1}^2)\,.
\end{aligned} \tag{G.34}$$

Note que os valores esperados $s_{\theta,\gamma}$ já foram todos devidamente calculados na Subseção G.2.1. Isso conclui o cálculo de μ_3.

G.2.4 Cálculo de μ_4

Até agora os momentos de quarta ordem calculados possuíam apenas dois ou três vetores de variáveis aleatórias, $\boldsymbol{u}(n)$, $\boldsymbol{u}(\omega_m)$ e $\boldsymbol{u}(\omega_{m'})$. Isso não acontece com μ_4, que é obtido por:

$$\begin{aligned}
\mu_4 &= E\Big\{[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)]^4[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{m'})]^2[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{m''})]^2\Big\} \\
&= \sum_{\ell_1=0}^{4}\sum_{\ell_2=0}^{2}\sum_{\ell_3=0}^{2} \mathsf{C}^4_{4-\ell_1}\mathsf{C}^2_{2-\ell_2}\mathsf{C}^2_{2-\ell_2}\, a^{\ell_1+\ell_2+\ell_3} \\
&\quad\times \sum_{(k_1+k_2=4-\ell_1)}\ \sum_{(k_1'+k_2'=4-\ell_2)}\ \sum_{(k_1''+k_2''=2-\ell_3)} \mathsf{C}^{4-\ell_1}_{k_1,k_2}\mathsf{C}^{2-\ell_2}_{k_1',k_2'}\mathsf{C}^{2-\ell_2}_{k_1'',k_2''} \\
&\quad\times E\Big\{[\boldsymbol{u}(\omega_m)]_1^{k_1}[\boldsymbol{u}(\omega_m)]_2^{k_2}\Big\}\, E\Big\{[\boldsymbol{u}(\omega_{m'})]_1^{k_1'}[\boldsymbol{u}(\omega_{m'})]_2^{k_2'}\Big\} \\
&\quad\times E\Big\{[\boldsymbol{u}(\omega_{m''})]_1^{k_1''}[\boldsymbol{u}(\omega_{m''})]_2^{k_2''}\Big\}\, E\Big\{[\boldsymbol{u}(n)]_1^{k_1+k_1'+k_1''}[\boldsymbol{u}(n)]_2^{k_2+k_2'+k_2''}\Big\} \\
&= \sum_{\ell_1=0}^{4}\sum_{\ell_2=0}^{2}\sum_{\ell_3=0}^{2} \mathsf{C}^4_{4-\ell_1}\mathsf{C}^2_{2-\ell_2}\mathsf{C}^2_{2-\ell_3} a^{\ell_1+\ell_2+\ell_3} S(4-\ell_1, 2-\ell_2, 2-\ell_3)\,.
\end{aligned} \tag{G.35}$$

em que $1 \leq m, m', m'' \leq M$ e m, m', m'' são distintos.

Note que se uma das componentes do somatório $S(i,j,\ell)$ for nula, podemos fazer:

$$S(i,j,0) = S(i,0,j) = S(0,i,j) = S(i,j),\ \ \forall i,j. \tag{G.36}$$

Desta maneira, devemos encontrar as expressões de $S(4,2,2)$ e $S(2,2,2)$ em função dos momentos de $\boldsymbol{u}(n)$, porque as demais expressões já foram calculadas anteriormente. Isto pode ser feito com o seguinte

procedimento:

$$\begin{aligned} S(4,2,2) =& 6\, s_{2,0}^4 (s_{0,8} + s_{4,4} + 2\, s_{2,6}) + 24\, s_{0,2}^3\, s_{1,1} (s_{1,7} + 3\, s_{3,5}) \\ &+ 12\, s_{0,2}^2 (s_{0,2}^3 + s_{1,1}^2)(s_{6,2} + s_{4,4}) + 24\, s_{0,2}^3\, s_{1,1} (s_{1,7} + s_{3,5}) \\ &+ 96\, s_{0,2}^2\, s_{1,1}^2 (s_{2,6} + s_{4,4}) + 48\, s_{0,2}^2 s_{1,1}\, s_{3,5} (s_{0,2}^3 + s_{1,1}^2) \\ &+ 24\, s_{0,2}^2\, s_{1,1}^2\, s_{2,6} + 96\, s_{0,2}\, s_{1,1}^3\, s_{3,5} + 24\, s_{0,2}\, s_{4,4} (s_{6,0} + s_{1,1}^2) \end{aligned} \tag{G.37}$$

e

$$\begin{aligned} S(2,2,2) =& 2\, s_{0,2}^3 (s_{0,6} + 3\, s_{2,4}) + 12\, s_{1,1}\, s_{0,2}^2 (s_{1,5} + s_{3,3}) + 24\, s_{0,2}\, s_{1,1}^2\, s_{2,4} \\ &+ 8\, s_{1,1}^3\, s_{3,3}. \end{aligned} \tag{G.38}$$

Novamente, omitimos o caso em que a soma $(\ell_1 + \ell_2 + \ell_3)$ é ímpar, pois neste caso $S(\ell_1, \ell_2, \ell_3)$ é nulo.

G.2.5 Cálculo de μ_5

Para finalizarmos os cálculos dos momentos de quarta ordem, fazemos agora o cálculo de μ_5:

$$\begin{aligned}
\mu_5 =& E\Big\{[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_m)]^2[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{m'})]^2[a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{m''})]^2 \\
&\times [a+\boldsymbol{u}^\top(n)\boldsymbol{u}(\omega_{m'''})]^2\Big\} \\
=& \sum_{\ell_1=0}^{2}\sum_{\ell_2=0}^{2}\sum_{\ell_3=0}^{2}\sum_{\ell_4=0}^{2} \mathsf{C}_{2-\ell_1}^{2}\mathsf{C}_{2-\ell_2}^{2}\mathsf{C}_{2-\ell_3}^{2}\mathsf{C}_{2-\ell_4}^{2}\, a^{\ell_1+\ell_2+\ell_3+\ell_4} \sum_{(k_1+k_2=2-\ell_1)} \\
&\times \sum_{(k_1''+k_2''=2-\ell_3)}\ \sum_{(k_1'''+k_2'''=2-\ell_4)} \mathsf{C}_{k_1,k_2}^{2-\ell_1}\mathsf{C}_{k_1',k_2'}^{2-\ell_2}\mathsf{C}_{k_1'',k_2''}^{2-\ell_2}\mathsf{C}_{k_1''',k_2'''}^{2-\ell_2} \\
&\times E\left\{[\boldsymbol{u}(\omega_m)]_1^{k_1}[\boldsymbol{u}(\omega_i)]_2^{k_2}\right\} E\left\{[\boldsymbol{u}(\omega_{m'})]_1^{k_1'}[\boldsymbol{u}(\omega_{m'})]_2^{k_2'}\right\} \\
&\times E\left\{[\boldsymbol{u}(\omega_{m''})]_1^{k_1''}[\boldsymbol{u}(\omega_{m''})]_2^{k_2''}\right\} E\left\{[\boldsymbol{u}(\omega_{m'''})]_1^{k_1'''}[\boldsymbol{u}(\omega_{m'''})]_2^{k_2'''}\right\} \\
&\times E\left\{[\boldsymbol{u}(n)]_1^{k_1+k_1'+k_1''+k_1'''}[\boldsymbol{u}(n)]_2^{k_2+k_2'+k_2''+k_2'''}\right\}
\end{aligned} \tag{G.39}$$

em que $1 \leq m, m', m'', m''' \leq M$ e m, m', m'', m''' são distintos.

Novamente fazendo,

$$S(i,j,\ell,0) = S(i,j,0,\ell) = S(i,0,j,\ell) = S(0,i,j,\ell) = S(i,j,\ell),\ \ \forall i,j,\ell. \tag{G.40}$$

Note que apenas o valor esperado $S(2,2,2,2)$ de (G.39) ainda não foi calculado. Para isso, fazemos:

$$\begin{aligned}
S(2,2,2,2) =& 2\,s_{0,2}^4\,(s_{0,8}+4\,s_{2,6}+3\,s_{4,4}) + 16\,s_{0,2}^3\,s_{1,1}\,(s_{1,7}+3\,s_{3,5}) \\
&+ 48\,s_{0,2}^2\,s_{1,1}^2\,(s_{2,6}+s_{4,4}) + 24\,s_{0,2}\,s_{1,1}^3\,s_{3,5} \\
&+ 16\,s_{1,1}\,s_{4,4}
\end{aligned} \tag{G.41}$$

Isto completa o cálculo dos momentos de quarta ordem do *kernel* polinomial do segundo grau com $\boldsymbol{u}(n)$ i.i.d. gaussiano no $\mathbb{R}^2$.

ANEXO H – PROGRAMAS USADOS

H.1 Kernel gaussiano e polinomial

```
%==================================================
% function babel tests the coherence level.
% function [bab,coh]=babel(K)
%
% input of the function
% K : matrix gram of the kernel.
%==================================================

function [bab,coh]=babel(K)

K=K./sqrt(diag(K)*diag(K)');
K=K-diag(diag(K));

coh=max(max(abs(K)));
bab=max(sum(abs(K)));
```

```
%==================================================
% function [G,H] = makeG(M,Mom1,Mom2,Mom3,Mom4,Mom5, ...
    eta, rmd, rod)
% inputs of the function:
%
% M                          : the dictionary lenght
% Mom1, Mom2, Mom3, Mom4, Mom5 : the 4th order ...
    statistical moments
% eta                        : the step-size of the ...
    KLMS agorithm
% rmd                        : the main-diagonal ...
    component of the
% correlation matrix Rkk.
```

```
% rod                          : the off-diagonal ...
    component of the
% correlation matrix Rkk.
%
% the outputs of the function
% G : state transition matrix
% H : family matrices
%================================================
function [G,H] = makeG(M,Mom1,Mom2,Mom3,Mom4,Mom5, eta, ...
    rmd, rod)

H = zeros(M,M,M,M);
for l=1:M

    for p=1:M

        for i=1:M
            for j= 1:M
                if l==p && i==j && i==l

                    H(i,j,l,l) = 1-2*eta*rmd + Mom1*(eta^2);

                elseif l==p && i==j && i~=l
                    H(i,j,l,l) = Mom3*(eta^2);

                elseif l==p && i~=j && (i==l || j==l)

                    H(i,j,l,l) = (eta^2)*Mom2 - eta*rod ;

                elseif l==p &&  i~=j && i~=l && j~=l
                    H(i,j,l,l) = (eta^2)*Mom4;
                end

            end
        end
    end
end
```

```
45
46
47  for l=1:M
48      for p=1:M
49          for i=1:M
50              for j=1:M
51
52                  if l≠p && i==l && j ==p
53
54                      H(i,j,l,p) = 0.5*(1 −2*eta*rmd ...
                            +2*(eta^2)*Mom3);
55
56                  elseif l≠p && i==p && j==l
57
58                      H(i,j,l,p) = 0.5*(1 − 2*eta*rmd ...
                            +2*(eta^2)*Mom3);
59
60                  elseif i==j && l≠p && (i==p || i==l)
61                      H(i,j,l,p) = Mom2*eta^2 − eta*rod;
62
63                  elseif i==j && l≠p && (i≠p || i≠l)
64                      H(i,j,l,p) = Mom4*eta^2;
65
66                  elseif l≠p && i≠j && (i==p || j==l)
67                      H(i,j,l,p) = 0.5*(2*Mom4*eta^2 − ...
                            eta*rod);
68
69                  elseif l≠p && i≠j && (i==l || j==p)
70                      H(i,j,l,p) = 0.5*(2*Mom4*eta^2 − ...
                            eta*rod);
71
72                  elseif l≠p && i≠j && i≠l && j≠p
73                      H(i,j,l,p) = Mom5*eta^2;
74
75                  end
76              end
77          end
```

```
    end
end

G = [];

for l=1:M

    for p=1:M

        aux = H(:,:,l,p)';

        G = [G; aux(1:M^2)];
    end

end
```

```
%================================================
% function [ATmd, ATod] = M_Mom(M,m1,m2,m3,m4,m5)
%
% This function creates a matrix of statiscal moments, ...
    based on inputs
% M   : dictionary length
% m1  : the 4th order statiscal moment, mu_1
% m2  : the 4th order statiscal moment, mu_2
% m3  : the 4th order statiscal moment, mu_3
% m4  : the 4th order statiscal moment, mu_4
% m5  : the 4th order statiscal moment, mu_5
%================================================
function [ATmd, ATod] = M_Mom(M,m1,m2,m3,m4,m5)

ATmd = zeros(M,M,M); %main diagonal components

for i=1:M

    for p=1:M
```

```
19
20          for q=1:M
21
22              if p==q && p==i
23
24                  ATmd(p,q,i) = m1;
25
26              elseif p==q && p≠i
27                  ATmd(p,q,i) = m3;
28
29              elseif p==i || q==i
30
31                  ATmd(p,q,i) = m2;
32
33              else
34                  ATmd(p,q,i) = m4;
35
36
37              end
38          end
39      end
40  end
41
42
43
44  comb = combnk(1:M,2);
45  [Lin Col] = size(comb);
46
47  if comb(1,1) ≠1
48
49     comb = comb(Lin:—1:1,:);
50
51  end
52
53
54  ATod = zeros(M,M,Lin); %off—diagonal components
55
```

```
for i=1:Lin

    for p=1:M

        for q=1:M


         if p==q && p == comb(i,1) ||  p==q && p == ...
             comb(i,2)

             ATod(p,q,i) = m2;

         elseif p==q && p ≠ comb(i,1) ||  p==q && p ≠ ...
             comb(i,2)

             ATod(p,q,i) = m4;

         elseif p == comb(i,1) && q == comb(i,2)

             ATod(p,q,i) = m3;
         elseif q == comb(i,1) && p == comb(i,2)

             ATod(p,q,i) = m3;


         elseif p == comb(i,1) || q == comb(i,2) || q == ...
             comb(i,1) || p == comb(i,2)

            ATod(p,q,i) = m4;

         else

            ATod(p,q,i) = m5;

         end


```

```
90          end
91      end
92  end
```

```
1  %==================================================
2  % Behavior of the dictionary lenght
3  % function dic_length = dic_behavior(it_dic, rlz, su, ...
       rho, dms, nc_v, ker, parameter)
4  % This function computes the average of the dictionary ...
       length for the input
5  % parameters:
6  % it_dic    : number of iterations for the seach the ...
       stable dimension of the
7  %             dicionaries.
8  % rlz       : number of Monte Carlo realization
9  % su        : standard deviation of the input signal
10 % rho       : correlation level
11 % dms       : number of components of the signal u(n).
12 % nc_v      : the coherence threshold
13 % ker       : used kernel function
14 % parameter : parameter of the kernel function
15 %
16 %
17 % The output of this function is a matrix dic_lenght, ...
       where the colum
18 % represents the coherence threshold variation and the ...
       row represents the
19 % kernel parameter variation.
20 %--------------------------------------------------
21 % values for dms: 1 or 2.
22 % values for ker: 'gaussian' or 'polynomial2'
23 % the parameter and nc_v can be defined as scalar or vector.
24 %==================================================
25 function dic_length = dic_behavior(it_dic, rlz, su, rho, ...
       dms, nc_v, ker, parameter)
```

```
26
27  col = length(nc_v);
28  lin = length(parameter);
29
30  CDic= zeros(lin,col);
31
32
33  for c=1:col
34
35      nc = nc_v(1,c);
36
37      for l=1:lin
38
39          switch lower(ker)
40
41              case 'gaussian'
42                  xi = parameter(l);
43
44                  M_med = 0;
45
46                  for r=1:rlz
47
48                      if dms == 1
49                          u_dic(1,:) = su*randn(1,it_dic);
50                      elseif dms == 2
51                          u_dic(1,:) = su*randn(1,it_dic);
52                          u_dic(2,:) = rho*u_dic(1,:) + ...
                                su*sqrt(1-rho^2)*randn(1,it_dic);
53                      end
54
55
56                      Dic = bdictinary_gauss(xi,nc, ...
                            u_dic); % function that build ...
                            the dicitonary polynomial
57
58                      [dms,M] = size(Dic);
59
```

```
60                      M_med = M + M_med;
61
62                  end
63
64
65              case 'polynomial2'
66
67                  cte = parameter(l);
68                  deg =2; % degree of the polynomial kernel
69                  M_med = 0;
70
71                  for r=1:rlz
72
73                      if dms == 1,
74                          u_dic(1,:) = su*randn(1,it_dic);
75                      elseif dms == 2,
76                          u_dic(1,:) = su*randn(1,it_dic);
77                          u_dic(2,:) = rho*u_dic(1,:) + ...
                                su*sqrt(1-rho^2)*randn(1,it_dic);
78                      end
79
80                      Dic = bdictinary_polynom(deg,cte,nc, ...
                            u_dic);
81                      [dms,M] = size(Dic);
82
83                      M_med = M + M_med;
84
85                  end
86
87          end
88
89          CDic(l,c) = (M_med/rlz);
90
91      end
92
93  end
94
```

```
dic_length = round(CDic);
```

```
%================================================
% function eta_max = max_stepsize(M, rmd, rod, Mom1, ...
    Mom2, Mom3, Mom4, Mom5)
%
% This function find the stability limits using the ...
    Gerschgorin disc test,
% using Gaussian kernel (positive-values kernel) or ...
    Polynomial kernel.
%
% Inputs
% M   : dictionary size
% rmd : autocorrelation of inputsignal
% rod : crosscorelation of outputsignal
% Mom_i : 4th order moments
%================================================
function eta_max = max_stepsize(M, rmd, rod, Mom1, Mom2, ...
    Mom3, Mom4, Mom5)

if M ==1
    eta_max= 2*rmd/Mom1;

elseif M ==2

    eta_1 = (2*rmd + 2*rod)/(Mom1 + 2*Mom2 + Mom3);

    theta1 = (2*rmd - 2*rod);
    theta2 = (Mom1 - 2*Mom2 + Mom3);

    if theta1>0 && theta2>0
        eta_max = min(eta_1, theta1/theta2);
    else
        eta_max= eta_1;
    end
```

```
30
31 elseif M≥3
32     eta_1 = 2*(rmd + (M—1)*rod)/(Mom1 + 2*(M—1)*Mom2 + ...
           (M—1)*Mom3 + (M—1)*(M—2)*Mom4);
33
34     theta1 = 2*(rmd — (M—1)*rod);
35     theta2 = (Mom1 — 2*(M—1)*Mom2 + (M—1)*Mom3 + ...
           (M—1)*(M—2)*Mom4);
36
37     eta_2 = theta1/theta2;
38
39
40     if (theta1 > 0) && (theta2 > 0)
41
42         eta_max1 = min(eta_1, eta_2);
43
44     elseif (theta2 < 0) && (theta1 ≠0)
45
46         eta_max1 = eta_1;
47
48     elseif (theta1 < 0) && (theta2 > 0)
49
50         eta_max1 = —inf;
51
52     elseif theta1 == 0
53
54         eta_max1 = NaN;
55
56     end
57
58
59     theta3= (—Mom1 + (M—1)*Mom3 + 4*(M—1)*(M—2)*Mom4 + ...
           (M—1)*(M—2)*(M—3)*Mom5);
60
61     eta_3 = 2*(M—2)*(rmd + (M—1)*rod)/theta3;
62
63     theta4 = (—Mom1 + (M—1)*Mom3 —4*(M—1)*(M—2)*Mom4 + ...
```

```
        (M-1)*(M-2)*(M-3)*Mom5);

    eta_4 = (M-2)*theta1/theta4;


    if (theta3>0) && (theta1 > 0) && (theta4 > 0)

        eta_max2 = min(eta_3, eta_4);

    elseif (theta3>0) && (theta4 < 0)  && (theta1~=0)
        eta_max2 = eta_3;

    elseif (theta3>0) && (theta4 > 0) && (theta1 < 0)
        eta_max2= -inf;


    elseif (theta3<0) && (theta1 > 0) && (theta4 > 0)

        eta_max2 = eta_4;

    elseif (theta3<0) && (theta4 < 0) && (theta1~=0)
        eta_max2 = inf;

    elseif (theta3<0) && (theta4 > 0) && (theta1 < 0)
         eta_max2 = -inf;

    elseif theta1 == 0

        eta_max2 = NaN;
    end


    eta_max = min(eta_max1, eta_max2);

end
```

```
%=================================================
% N_inf = time_conv2(M,Rk,Cv_ss, c0,c_inf,epsilon,G)
%
% N_inf is the necessary time to reach the convergence,
% with the inputs:
% M       : dictionary size
% Cv_ss   : steady-state of correlation matrix of the error
% weight-vector C_v(n)
% c0      : lexicographic representation of Cv(0)
% c_inf   : lexicographic representation of Cv_ss
% epsilon : tolerance
% G       : state-transition matrix
%=================================================

function N_inf = time_conv2(M,Rk,Cv_ss, c0,c_inf,epsilon,G)

n=0;

dif = 1000 ;

Jex_ss =trace(Rk*Cv_ss); %steady-state of Excess MSE


while norm(dif) ≥ epsilon

    cn = (G^n)*(c_inf - c0') + c_inf;

    Cv_n = zeros(M);

    t=1;
    for i=1:M
        for j=1:M

            Cv_n(i,j) = cn(t);
            t=t+1;
        end
```

```
    end

    dif = trace(Rk*Cv_n) - Jex_ss;
    N_inf = n;
    clc
    n = n+1;

end
```

```
%==================================================
% Performance analysis of Kernel LMS - Statistic Simulation
%==================================================
% The dictionary update:
% In this program the dimension control is made using the
% bdictinary.m that generate the dictionary with the ...
    parameters (kernel
% bandwidth, coherence level and input signal), the ...
    dictionary lenght used
% is the lenght of dictionary obtained.
%==================================================
% KLMS using on the Nonlinear System Identification
%==================================================

close all;
clear all;
clc;

ker = 'gaussian'; %input('select the kernel polynomial2 ...
    or gaussian: '); % this program parameter can be ...
    gaussian.
example = 'example1'; %input('select the example: ');    ...
    % select the example test 1,2 or 3.
N = 500; %input('Set the iteration number: ');
param = 0.0225; %input('Input kernel parameter: ');
eta = 0.0393; %input('Input the step-size of KLMS: ');
```

```
%% Inicialization
it_dic =500;
rlz = 500;

switch lower(example)

    case 'example1'

        su = 0.15;
        rho = 0.5;
        sw = 1e-2;
        dms= 1;
        Jmax= -22.35;
        NN= N+1;

        switch lower(ker)
            case 'gaussian'
                nc_v= 1e-5; epsilon= 0.5*1e-4;
            case 'polynomial2'
                nc_v= 0.7;  epsilon= 2.5*1e-4;
        end

    case 'example2'

        su = 0.125;
        rho = 0.5;
        sw = 1e-3;
        dms= 2;
        Jmax= -20.25;
        NN= N+2;
        switch lower(ker)
            case 'gaussian'
                nc_v= 1e-4;    epsilon= 1e-3;
            case 'polynomial2'
                nc_v = 0.85;  epsilon = 2.5*1e-4;
        end

```

```
59
60      case 'example3'
61          su = 0.25;
62          rho = 0.5;
63          sw = 1e-2;
64          dms= 2;
65          Jmax= -19.75;
66          NN=N+2;
67          switch lower(ker)
68              case 'gaussian'
69                  epsilon= 1e-4; nc_v= 1e-1;
70              case 'polynomial2'
71                  nc_v = 0.6;    epsilon = 5*1e-4;
72          end
73  end
74
75  M = dic_behavior(it_dic, rlz, su, rho, dms, nc_v, ker, ...
        param);
76
77  switch lower(ker)
78      case 'gaussian'
79          [Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] = ...
                mom_gaussiano(M, su, dms, rho, param);
80      case 'polynomial2'
81          [Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] = ...
                mom_polynomial(M, su, dms, rho, param);
82  end
83
84  %% Variables initialization
85  etd = zeros(1,N);
86  corr_dn =0;
87  Pkd =0;
88  Rk = (rmd-rod)*eye(M)+rod*ones(M);
89  m_med = M;
90
91  sb=waitbar(0,'Wait ...'); %status bar
92  u_dic = zeros(dms,it_dic);
```

```
93
94  for r=1:rlz
95
96      M = 1e10;
97
98      while M ≠ m_med
99
100         switch lower(example)
101
102             case 'example1'
103                 u_dic = su*randn(dms,it_dic);
104
105                 switch lower(ker)
106                   case 'gaussian'
107                     Dic0 = bdictinary_gauss(param,nc_v, ...
                            u_dic); % function that build ...
                            the dicitonary
108                     [dms,M] = size(Dic0);
109
110                   case 'polynomial2'
111                     Dic0 = ...
                            bdictinary_polynom(param,nc_v, ...
                            u_dic); % function that build ...
                            the dicitonary
112                     [dms,M] = size(Dic0);
113                 end
114
115             case 'example2'
116                 u_dic = zeros(dms,it_dic);
117
118                 for l=1:it_dic
119                     u_dic(1,l) = su*randn(1,1);
120                     u_dic(2,l) = rho*u_dic(1,l) + ...
                            su*sqrt(1 - rho^2)*randn(1,1);
121                 end
122
123                 switch lower(ker)
```

```
124                 case 'gaussian'
125                   Dic0 = bdictinary_gauss(param,nc_v, ...
                          u_dic); % function that build ...
                          the dicitonary
126                   [dms,M] = size(Dic0);
127
128                 case 'polynomial2'
129                   Dic0 = ...
                          bdictinary_polynom(param,nc_v, ...
                          u_dic); % function that build ...
                          the dicitonary
130                   [dms,M] = size(Dic0);
131               end
132
133
134
135           case 'example3'
136               u_dic = zeros(dms,it_dic);
137
138               for l=1:it_dic
139                   u_dic(1,l) = su*randn(1,1);
140                   u_dic(2,l) = rho*u_dic(1,l) + ...
                          su*sqrt(1 - rho^2)*randn(1,1);
141               end
142
143               switch lower(ker)
144                  case 'gaussian'
145                    Dic0 = bdictinary_gauss(param,nc_v, ...
                          u_dic); % function that build ...
                          the dicitonary
146                    [dms,M] = size(Dic0);
147
148                  case 'polynomial2'
149                    Dic0 = ...
                          bdictinary_polynom(param,nc_v, ...
                          u_dic); % function that build ...
                          the dicitonary
```

```
150                     [dms,M] = size(Dic0);
151                 end
152
153             end
154
155
156         end
157         Dic = Dic0;
158
159         %% Variables initialization
160         e = zeros(1,N);     % error
161         v=1:M;
162         V = v'/sqrt(trace(Rk*(v'*v))); %value used for V0 is ...
                calculated using the KLMS
163         %model, so that the statistic simulation curve
164         %uses the same inialization of the model.
165
166         un = zeros(dms,NN);
167         yn = zeros(NN,1);
168         dn = zeros(NN,1);
169         d= zeros(NN,1);
170
171
172         switch lower(example)
173
174             case 'example1'
175                 for n = 2:NN
176
177                     un(1,n) = su*randn(1,1);
178                     yn(n,1) = (un(n-1))^3 + (yn(n-1,1))/(1 + ...
                            (yn(n-1,1))^2);
179                     dn(n,1) = yn(n,1);
180                     d(n,1) = dn(n,1) + sw*randn(1,1);
181
182                     %% Computation of the Kernel gram matrix
183                     Dic_t = [un(:,n) Dic];
184                     K = zeros(M,1);
```

```
185                 for l=1:M
186                     switch lower(ker)
187                       case 'gaussian'
188                         difer = un(:,n) - Dic(l);
189                         K(l,1) = ...
                                exp(-(difer'*difer)/(2*param^2));
190                       case 'polynomial2'
191                         i_prod = un(:,n)'*Dic(:,l);
192                         K(l,1) = ((param + i_prod)^deg);
193                     end
194                 end
195
196                 %% Calculus of mean-squared error
197                 e(n) = d(n) - V'*K;
198                 V = V + eta*e(n)*K; % weight-error vector
199                 Dic = Dic_t(:,1:M);
200             end
201             e = e(2:NN);
202             etd = etd + e.^2; % sum of squared error
203             %Pkd = Pkd + K*d(N);
204             %corr_dn = d.^2 + corr_dn;
205
206             waitbar(r/rlz,sb)  %status bar
207
208         case 'example2'
209             A = [1; 0.5];
210
211             for n = 3:NN
212                 un(1,n) = su*randn(1,1);
213                 un(2,n) = rho*un(1,n) + ...
                        su*sqrt(1-rho^2)*randn(1,1);
214                 yn(n,1) = A'*un(:,n) - 0.2*yn(n-1,1) + ...
                        0.35*yn(n-2,1) ;
215
216                 if yn(n,1) >= 0
217                    dn(n,1) = yn(n,1)/(3*(0.1 + ...
                           0.9*(yn(n,1))^2)^(1/2));
```

```
218                 else
219                    dn(n,1) = ...
                           -(yn(n,1)^2)*(1-exp(0.7*yn(n,1)))/3;
220                 end
221                 d(n,1) = dn(n,1) + sw*randn(1,1);
222
223                 %% Computation of the Kernel gram matrix
224                 Dic_t = [un(:,n) Dic];
225                 K = zeros(M,1);
226                 for l=1:M
227                     switch lower(ker)
228                       case 'gaussian'
229                         difer = un(:,n) - Dic(l);
230                         K(l,1) = ...
                                exp(-(difer'*difer)/(2*param^2));
231                       case 'polynomial2'
232                         i_prod = un(:,n)'*Dic(:,l);
233                         K(l,1) = ((param + i_prod)^deg);
234                     end
235                 end
236                 %% Calculus of mean-squared error
237
238                 e(n) = d(n) - V'*K; % error
239                 V = V + eta*e(n)*K; % weight-error vector
240
241                 Dic = Dic_t(:,1:M);
242             end
243
244             e = e(3:NN);
245             etd = etd + e.^2; % sum of squared error
246             %Pkd = Pkd + K*d(N);
247             %corr_dn = d.^2 + corr_dn;
248
249             waitbar(r/rlz,sb)  %status bar
250
251         case 'example3'
252             A = [0.1044; 0.0883];
```

```
253
254             for n = 3:NN
255                 un(1,n) = su*randn(1,1);
256                 un(2,n) = rho*un(1,n) + ...
                        su*sqrt(1-rho^2)*randn(1,1);
257                 yn(n,1) = A'*un(:,n) + 1.4138*yn(n-1,1) ...
                        - 0.6065*yn(n-2,1) ;
258                 dn(n,1) = 0.3163*yn(n,1)/(sqrt(0.10 ...
                        +0.9*yn(n,1)^2));
259                 d(n,1) = dn(n,1) + sw*randn(1,1);
260
261                 %% Computation of the Kernel gram matrix
262                 Dic_t = [un(:,n) Dic];
263                 K = zeros(M,1);
264
265                 for l=1:M
266                     switch lower(ker)
267
268                       case 'gaussian'
269                         difer = un(:,n) - Dic(l);
270                         K(l,1) = ...
                                exp(-(difer'*difer)/(2*param^2));
271                       case 'polynomial2'
272                         i_prod = un(:,n)'*Dic(:,l);
273                         K(l,1) = ((param + i_prod)^deg);
274                     end
275                 end
276                 %% Calculus of mean-squared error
277                 e(n) = d(n) - V'*K;
278                 V = V + eta*e(n)*K; % weight-error vector
279                 Dic = Dic_t(:,1:M);
280             end
281             e = e(3:NN);
282             etd = etd + e.^2; % sum of squared error
283             %Pkd = Pkd + K*d(N);
284             %corr_dn = d.^2 + corr_dn;
285
```

```
286             waitbar(r/rlz,sb)  %status bar
287     end
288
289 end
290 close(sb)
291
292 etd=etd/rlz; % mean-squared error
293
294 %Pkd_stat = Pkd/rlz;
295 %Corr_dn = corr_dn(N)/rlz;
296
297 %save ./gaussian_results/stat.mat Pkd_stat Corr_dn
298 %save ./gaussian_results/ss_res.mat etd M rho rlz su ...
        Sigma_0 N NN sw eta xi dms
299
300 figure
301 plot(1:N,10*log10(etd(1:N)),'b')
302 title('\bf Theoretical Model and Monte Carlo Simulation ');
303 xlabel('\bf n')
304 ylabel('\bf MSE [dB]')
305 grid
306
307 %load ./gaussian_results/mod_res.mat Jms Jms_inf Jex ...
        Jex_inf Jmin C_v M N eta N_inf
308 % figure
309 % plot(1:N, 10*log10(etd(1:N)), 'b', 1:N, ...
        10*log10(Jms(1:N)), 'r')
310 % xlabel('\bf iterations','FontSize',18)
311 % ylabel('\bf MSE [dB]','FontSize',18)
312 % grid
```

H.2 Kernel gaussiano

```
1 %================================================
```

```
% BDICTIONARY  creates a dictionary from the parameters ...
    requested.
% dictionary = bdictinary(xi,lim_coh, uc0) returns an ...
    N-by-M matrix
% containing the sparse dictionary obtained from xi ...
    (kernel gaussian
% bandwidth) , nc (threshold coherence) and uc0 ( input ...
    signal ).
% In this case M is the dictionary length and N is the ...
    length of each component
% of the dictionary.
%================================================

function dictionary = bdictinary_gauss(xi,lim_coh, uc0)

    % dictionary initialization
    dictionary = uc0(:,1);
    [dms,N] = size(uc0);
    n0 = 2;
    fora = 0;

while n0 <= N && fora < N/3

        Dic_t = [uc0(:,n0) dictionary];
        [dms MM]= size(Dic_t);

        K0 = zeros(MM);

        for l0=1:MM

            for c0=1:MM

                difer0 = Dic_t(:,c0)-Dic_t(:,l0);

                K0(l0,c0) = exp(-(difer0'*difer0)/(2*xi^2));

            end
```

```
        end

        [bab0,coh0] = babel(K0);

        if coh0 ≤ lim_coh

           dictionary = Dic_t;
           fora = 0;
        else
            fora = 1 + fora;
        end

        n0 = n0+1;


end
```

```
%=================================================
% function [Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, ...
    Mom5] = mom_gaussiano(M, su, dms, rho, xi)
%
% For the polynomial kernel of 2nd order, the ...
    statistical moments
% 1st order moment : Expv
% 2nd order moments : rmd (main-diagonal of Rkk) and rod ...
    (off-diagonal of Rkk)
% 4th order moments : Mom1 (mu_1), Mom2 (mu_2), ... Mom5 ...
    (mu_5)
%
% are obtained from this function based on inputs:
% M    : dictionary size
% su   : standard deviation of input signal
% rho  : correlation level
% xi   : kernel bandwidth.
%=================================================
```

```
15
16 function [Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] ...
       = mom_gaussiano(M, su, dms, rho, xi)
17 mu=0;
18
19 I=eye(dms);
20 %% Moments of the input signal
21 % Correlation matrix input signal - AR(1) System
22
23 if dms == 2
24    Sigma_0 = (su^2)*[1 rho; rho 1];
25
26 elseif dms == 1
27     Sigma_0=su^2;
28 elseif dms ≥ 3
29     disp('Error! rewrite this program')
30 end
31
32 % Expected value kw(n)
33 O=[I -I;-I I];
34 Ry=[Sigma_0 zeros(dms); zeros(dms) Sigma_0];
35 Mu = mu*ones(2,1);
36
37 aux00 = eye(2*dms) + O*Ry/(xi^2);
38 %aux01 = -0.5*Mu'*(inv(Ry))*(eye(2*dms) - inv(aux00))*Mu;
39
40 Expv =1/(det(aux00)^(1/2));
41
42 % Correlation matrix - kw(n)
43
44 aux10 = eye(2*dms) + 2*O*Ry/(xi^2);
45 %aux11 = -0.5*Mu'*(inv(Ry))*(eye(2*dms) - inv(aux10))*Mu;
46
47 rmd = 1/(det(aux10)^(1/2)); % autocorrelation
48
49 %--
50 Mu2 = mu*ones(3,1);
```

```
O2 = [2*I -I -I;-I I zeros(dms); -I  zeros(dms) I];
Ry2 = [Sigma_0 zeros(dms) zeros(dms); zeros(dms) Sigma_0 ...
    zeros(dms); zeros(dms) zeros(dms) Sigma_0];

aux20 = eye(3*dms) + Ry2*O2/(xi^2);
%aux21 = -0.5*Mu2'*(inv(Ry2))*(eye(3*dms) - inv(aux20))*Mu2;

rod = 1/(det(aux20)^(1/2)); % cross-correlation


%% 4th order moment - Elimination of gaussian hypothesis

% case: i=j=l=p
Oper1 = [4*eye(dms) -4*eye(dms);-4*eye(dms) 4*eye(dms)];

matrix1a = eye(2*dms) + Oper1*(1/xi^2)*Ry;
matrix2a = ...
    -0.5*(mu^2)*ones(1,2*dms)*(inv(Ry))*(eye(2*dms) - ...
    inv(matrix1a))*ones(2*dms,1);
Mom1 = exp(matrix2a)/(det(matrix1a)^(1/2));

% case: (i=j=p≠l) or (i=j=l≠p) or (j=l=p≠i) or (i=l=p≠j)
Oper2 = [4*eye(dms) -3*eye(dms) -1*eye(dms); -3*eye(dms) ...
    3*eye(dms) 0*eye(dms); -1*eye(dms) 0*eye(dms) ...
    1*eye(dms)];

matrix1b = eye(3*dms) + Oper2*(1/xi^2)*Ry2;
matrix2b = ...
    -0.5*(mu^2)*ones(1,3*dms)*(inv(Ry2))*(eye(3*dms) - ...
    inv(matrix1b))*ones(3*dms,1);
Mom2 = exp(matrix2b)/(det(matrix1b)^(1/2));


% case: (i=j ≠ p=l) or (i=p ≠ l=j) or (i=l ≠ p=j)
Oper3 = [4*eye(dms) -2*eye(dms) -2*eye(dms); -2*eye(dms) ...
    2*eye(dms) 0*eye(dms); -2*eye(dms) 0*eye(dms) ...
    2*eye(dms)];
```

```
79
80  matrix1c = eye(3*dms) + Oper3*(1/xi^2)*Ry2;
81  matrix2c = ...
        -0.5*(mu^2)*ones(1,3*dms)*(inv(Ry2))*(eye(3*dms) - ...
        inv(matrix1c))*ones(3*dms,1);
82  Mom3 = exp(matrix2c)/(det(matrix1c)^(1/2));
83
84
85  % case: (i=j and ≠ p ≠ l) or (i=p and ≠ j ≠l) or (i=l ...
        and ≠ j ≠ p)
86  % (j=p and ≠ i ≠ p) or (j=l and ≠ i ≠ p) or (l=p and ≠ i ...
        ≠ j)
87  Oper4 = [4*eye(dms) -2*eye(dms) -1*eye(dms) ...
        -1*eye(dms);-2*eye(dms) 2*eye(dms) 0*eye(dms) ...
        0*eye(dms); -1*eye(dms) 0*eye(dms) 1*eye(dms) ...
        0*eye(dms);-1*eye(dms) 0*eye(dms) 0*eye(dms) ...
        1*eye(dms)];
88
89  Ry4 = [Sigma_0 zeros(dms) zeros(dms) zeros(dms) ; ...
        zeros(dms) Sigma_0 zeros(dms) zeros(dms);zeros(dms) ...
        zeros(dms) Sigma_0 zeros(dms); zeros(dms) zeros(dms) ...
        zeros(dms) Sigma_0];
90
91
92  matrix1d = eye(4*dms) + Oper4*(1/xi^2)*Ry4;
93  matrix2d = ...
        -0.5*(mu^2)*ones(1,4*dms)*(inv(Ry4))*(eye(4*dms) - ...
        inv(matrix1d))*ones(4*dms,1);
94
95  Mom4 = exp(matrix2d)/(det(matrix1d)^(1/2));
96
97  % case: i≠j≠p≠l
98  Oper5 = [4*eye(dms) -1*eye(dms) -1*eye(dms) -1*eye(dms) ...
        -1*eye(dms); -1*eye(dms) 1*eye(dms) 0*eye(dms) ...
        0*eye(dms) 0*eye(dms); -1*eye(dms) 0*eye(dms) ...
        1*eye(dms) 0*eye(dms) 0*eye(dms); -1*eye(dms) ...
        0*eye(dms) 0*eye(dms) 1*eye(dms) 0*eye(dms); ...
```

```
    -1*eye(dms) 0*eye(dms) 0*eye(dms) 0*eye(dms) ...
    1*eye(dms)];

Ry5 = [Sigma_0 zeros(dms) zeros(dms) zeros(dms) ...
    zeros(dms) ; zeros(dms) Sigma_0 zeros(dms) ...
    zeros(dms) zeros(dms); zeros(dms) zeros(dms) Sigma_0 ...
    zeros(dms) zeros(dms); zeros(dms) zeros(dms) ...
    zeros(dms) Sigma_0 zeros(dms);zeros(dms) zeros(dms) ...
    zeros(dms) zeros(dms) Sigma_0];


matrix1e = eye(5*dms) + Oper5*(1/xi^2)*Ry5;
matrix2e = ...
    -0.5*(mu^2)*ones(1,5*dms)*(inv(Ry5))*(eye(5*dms) - ...
    inv(matrix1e))*ones(5*dms,1);

Mom5 = exp(matrix2e)/(det(matrix1e)^(1/2));


if M==1
    rod = 0;
    Mom2=0;
    Mom3=0;
    Mom4=0;
    Mom5=0;

elseif M==2;

    Mom4=0;
    Mom5=0;

elseif M==3;
    Mom5=0;
end
```

```
%================================================
% This function was used to estimate the ...
    crosscorrelation of the desired
% signal and inputsignal presented in the Simulation ...
    Results of the thesis.
%================================================
% function corr_dn = cross_stat_gauss(example,rlz, N, ...
    sw, su, rho, lim_coh, M, xi);
%
% inputs of the function
% example  : desired signal of example
% rlz      : number of Monte carlo realization
% N        : number of iteration
% sw       : noise standard deviation
% su       : u_1(n) standard deviation
% rho      : correlation level of AR(1) process
% lim_coh  : coherence level
% M        : dictionary size
% xi       : kernel bandwidth
%
%  Values for example: 'example1'  – desired signal of ...
    example 1.
%                      'example2'  – desired signal of ...
    example 2.
%                      'example3'  – desired signal of ...
    example 3.
%================================================

function Pkd_stat = cross_stat_gauss(example,rlz, N, sw, ...
    su, rho, lim_coh, M, xi)

%% Simulation

NN=N+2;

N_dic=500;

```

```
% Variables initialization
Pkd = zeros(M,1);

for r=1:rlz
    M0=1000;

    switch lower(example)

    case 'example1'


        while M0≠ M
            u_dic = su*randn(1, N_dic);

            Dic0 = bdictinary_gauss(xi,lim_coh, u_dic); ...
                % function that build the dicitonary

            [dms,M0] = size(Dic0);

        end

        Dic = Dic0;


        %% Variables initialization

        un = zeros(dms,NN);

        yn = zeros(NN,1);
        dn = zeros(NN,1);
        d= zeros(NN,1);

        for n = 2:NN

            un(1,n) = su*randn(1,1);

            yn(n,1) = (un(n-1))^3 + (yn(n-1,1))/(1 + ...
```

```
            (yn(n-1,1))^2);

        dn(n,1) = yn(n,1);

        d(n,1) = dn(n,1) + sw*randn(1,1);

        %% Computation of the Kernel gram matrix
        Dic_t = [un(:,n) Dic];
        K = zeros(M,1);

        for l=1:M

            difer0 = un(:,n) - Dic(:,l);

            K(l,1) = exp(-(difer0'*difer0)/(2*xi^2));


        end

        Dic = Dic_t(:,1:M);
    end
    Pkd = Pkd + K*d(NN);

    case 'example2'

        while M0~= M
            u_dic = zeros(2, N_dic);
            for n=1:N_dic
                u_dic(1,n) = su*randn(1,1);
                u_dic(2,n) = rho*u_dic(1,n) + ...
                    su*sqrt(1-rho^2)*randn(1,1);
            end

            Dic0 = bdictinary_gauss(xi,lim_coh, ...
                u_dic); % function that build the ...
                dicitonary

```

```
100                 [dms,M0] = size(Dic0);
101
102             end
103
104             Dic = Dic0;
105
106
107             %% Variables initialization
108
109             un = zeros(dms,N+3);
110
111             yn = zeros(NN,1);
112             dn = zeros(NN,1);
113             d= zeros(NN,1);
114
115             A = [1; 0.5];
116
117
118             for n = 3:NN
119
120                 un(1,n) = su*randn(1,1);
121                 un(2,n) = rho*un(1,n) + ...
                        su*sqrt(1-rho^2)*randn(1,1);
122
123                 yn(n,1) = A'*un(:,n) - 0.2*yn(n-1,1) + ...
                        0.35*yn(n-2,1) ;
124
125                 if yn(n,1) >= 0
126
127                     dn(n,1) = yn(n,1)/(3*(0.1 + ...
                            0.9*(yn(n,1))^2)^(1/2));
128
129                 else
130
131                     dn(n,1) = ...
                            -(yn(n,1)^2)*(1-exp(0.7*yn(n,1)))/3;
132
```

```
133             end
134
135             d(n,1) = dn(n,1) + sw*randn(1,1);
136
137             %% Computation of the Kernel gram matrix
138             Dic_t = [un(:,n) Dic];
139             K = zeros(M,1);
140
141             for l=1:M
142
143                 difer0 = un(:,n) - Dic(:,l);
144
145                 K(l,1) = ...
                        exp(-(difer0'*difer0)/(2*xi^2));
146
147
148             end
149
150             Dic = Dic_t(:,1:M);
151         end
152         Pkd = Pkd + K*d(NN);
153
154         case 'example3'
155
156             while M0~= M
157                 u_dic = zeros(2, N_dic);
158                 for n=1:N_dic
159                     u_dic(1,n) = su*randn(1,1);
160                     u_dic(2,n) = rho*u_dic(1,n) + ...
                            su*sqrt(1-rho^2)*randn(1,1);
161                 end
162
163                 Dic0 = bdictinary_gauss(xi,lim_coh, ...
                        u_dic); % function that build ...
                        the dicitonary
164
165                 [dms,M0] = size(Dic0);
```

```
166
167                 end
168
169                 Dic = Dic0;
170
171
172                 %% Variables initialization
173
174                 un = zeros(dms,N+3);
175
176                 yn = zeros(NN,1);
177                 dn = zeros(NN,1);
178                 d= zeros(NN,1);
179
180                 A = [0.1044; 0.0883];
181
182
183                 for n = 3:NN
184
185                     un(1,n) = su*randn(1,1);
186                     un(2,n) = rho*un(1,n) + ...
                            su*sqrt(1-rho^2)*randn(1,1);
187
188                     yn(n,1) = A'*un(:,n) + ...
                            1.4138*yn(n-1,1) - ...
                            0.6065*yn(n-2,1) ;
189                     dn(n,1) = 0.3163*yn(n,1)/(sqrt(0.10 ...
                            +0.9*yn(n,1)^2));
190
191                     d(n,1) = dn(n,1) + sw*randn(1,1);
192
193                     %% Computation of the Kernel gram matrix
194                     Dic_t = [un(:,n) Dic];
195                     K = zeros(M,1);
196
197                     for l=1:M
198
```

```
                        difer0 = un(:,n) - Dic(:,l);

                        K(l,1) = ...
                            exp(-(difer0'*difer0)/(2*xi^2));


                    end

                    Dic = Dic_t(:,1:M);
                end

            Pkd = Pkd + K*d(NN);
    end

end

Pkd_stat = Pkd/rlz;
```

```
%================================================
% Jmin_v = MMSE_behav_gauss(example, rlz, N, sw, dms, ...
    su, rho, lim_coh,
% M_med, xi_v)
% This function computs the behavior of Minimum MSE
%
% Define the tests parameters
% example  : desired signal of example
% rlz      : number of Monte carlo realization
% N        : number of iteration
% sw       : noise standard deviation
% su       : u_1(n) standard deviation
% rho      : correlation level of AR(1) process
% lim_coh  : coherence level
% M_       : vector (or scalar) of dictionary size
% xi_v    : vector (or scalar) of kernel bandwiths
%
```

```
% and the outputs:
% the behavior of MMSE, which is Jmin_v.
%================================================

function Jmin_v = MMSE_behav_gauss(example, rlz, N, sw, ...
    su, rho, lim_coh, M_med, si_v)

Corr_dn = autocorrelation_dn(example,rlz, N, sw, su, rho);

Jmin_v = zeros(1,length(cte_v));

for p=1:length(cte_v)

    M= M_med(p);
    xi = xi_v(p);

    Pkd_stat = cross_stat_gauss(example,rlz, N, sw, su, ...
        rho, lim_coh, M, xi);

    % Expected value kw(n)
    O=[I -I;-I I];
    Ry=[Sigma_0 zeros(dms); zeros(dms) Sigma_0];

    % Correlation matrix - kw(n)

    aux10 = eye(2*dms) + 2*O*Ry/(xi^2);
    auto_corr = 1/(det(aux10)^(1/2));

    %--

    O2 = [2*I -I -I;-I I zeros(dms); -I  zeros(dms) I];
    Ry2 = [Sigma_0 zeros(dms) zeros(dms); zeros(dms) ...
        Sigma_0 zeros(dms); zeros(dms) zeros(dms) Sigma_0];

    aux20 = eye(3*dms) + Ry2*O2/(xi^2);

    cross_corr = 1/(det(aux20)^(1/2));
```

```
51
52      Rk = (auto_corr-cross_corr)*eye(M)+cross_corr*ones(M);
53
54      % Wiener solution
55
56      alpha_ot = Rk\Pkd_stat; % wiener solution
57
58      Jmin_v(p) = Corr_dn - Pkd_stat'*alpha_ot;  % Minimum ...
            error power
59
60  end
61
62  figure
63  plot(cte_v(1:p), 10*log10(Jmin_v(1:p)),'-o ...
        r',cte_v(1:p), (10*log10(Corr_dn))*ones(1,p),'-d b' )
64  xlabel('\bf constante do kernel polinomial','FontSize',18)
65  ylabel('\bf MMSE [dB]','FontSize',18)
66  grid
```

```
1  %================================================
2  % function [sumario] = stab_test_gauss ...
       (M_med,su,dms,rho,xi_v)
3  %
4  % Stability test of
5  % c(n+1) = Gc(n) + eta^2 r Jmin
6  % using the Gerschgorin disc test and eigenvalues of ...
       matrix G test.
7  % Input:
8  % M_med : vector (or scalar) dictionary size
9  % su    : standard deviation of input signal u(n)
10 % rho   : correlation level of u(n)
11 % xi_v : vector (or scalar) of kernel bandwidths
12 %
13 % Output
14 % sumario : is a matrix (or vector) when
```

```
%
% 1st column : constant term of the polynomial kernel
% 2nd column : dictionary size
% 3rd column : autocorrelation of kernel
% 4th column : crosscorrelation of kernel
% 5th — 9th column : 4th order moments
% 10th column: maximum step—size by eigenvalues of G test
% 11th column: maximum step—size by Gerschgorin disc test
%================================================

function [sumario] = stab_test_gauss (M_med,su,dms,rho,xi_v)

nx = length(xi_v);
sumario = zeros(nx,11);

for l=1:nx

    xi = xi_v(l);
    M = M_med(l);

    [Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] = ...
        mom_gaussiano(M, su, dms, rho, xi);


    %% Maximum step—size


    eta_max_modelo = max_stepsize(M, rmd, rod, Mom1, ...
        Mom2, Mom3, Mom4, Mom5);


    eta = 1e—5;
    s1 = M^2;
    s2 = s1;

    while (s1 == M^2) &&  (s2 == M^2)

```

```
50
51          [G,H] = makeG(M,Mom1,Mom2,Mom3,Mom4,Mom5, eta, ...
                rmd, rod);
52
53          clear lam_G test1 test2
54
55          lam_G = eig(G);
56
57          test1 = lam_G < 1;
58          test2 = lam_G >-1;
59
60          s1 = sum(test1);
61          s2 = sum(test2);
62
63          eta_simul = eta;
64
65          eta = eta + 1e-5;
66      end
67
68      sumario(l,:) = [xi M rmd rod Mom1 Mom2 Mom3 Mom4 ...
            Mom5  eta_simul eta_max_modelo] ;
69
70
71  end
72
73  indc = find(sumario==-inf);
74
75  for p=1:length(indc)
76
77      sumario(indc(p))=NaN;
78
79  end
```

```
1  %================================================
2  % Performance analysis of Kernel LMS - Model of the ...
```

```
    statistic behavior
%==================================================
% KLMS Gaussian on the Nonlinear System Identification
% Experiment from: Example 1
%==================================================

close all
clc
clear all

it_dic =500;
rlz = 500;

su = 0.15;
rho = 0.5;
sw = 1e-2;
dms= 1;
nc_v= 1e-5;
Jmax= -22.35;
epsilon= 0.5*1e-4;

cte = input('set the kernel bandwidth of Gaussian ...
    kernel: '); % set cte

dic_length = dic_behavior(it_dic, rlz, su, rho, dms, ...
    nc_v, 'polynomial2', cte);

M = dic_length;

Jmin = MMSE_behav_gauss('example1', rlz, 100, sw, su, ...
    dms, rho, nc_v, M, cte);

disp('||    kernel param   |    M     |   eta   | ...
    Jmin(dB)   |   Jmse(dB)   | Jex(dB)   | N_inf   ||')
Results = klms_gauss_ss(Jmin, Jmax, M, su, dms, rho, ...
    cte, epsilon);

```

```
34 eta = Results(:,3);
35
36 N = input('set the number of iteration: '); % select N < ...
       Results(1,7)
37
38
39
40
41 %% Moments of the input signal
42
43 [Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] = ...
       mom_gaussiano(dms, M, su,rho, cte);
44
45 Rk = (rmd-rod)*eye(M)+rod*ones(M);
46
47 %% Eigenvalue and eigenvector - Correlation Matrix
48 [Theta,Lambda] = eig(Rk);    % Lambda is the diagonal ...
       matrix of eigenvalues and Theta is a full matrix ...
       whose columns are the corresponding eigenvectors
49
50 lambda_max = rmd + (M-1)*rod; % maximum eigenvalue
51 lambda_min = rmd - rod; % minimum eigenvalue
52
53 lam = [lambda_min*ones(1,M-1) lambda_max];
54
55 %% Matrix Operator calculus
56
57 if M==1
58     ATmd = Mom1;
59 else
60
61     [ATmd, ATod] = M_Mom(M,Mom1,Mom2,Mom3,Mom4,Mom5);
62 end
63
64
65 %% Mean-Square Error: Transient and Steady-State
66 % ---- Variables initialization
```

```
v=1:M;
C_v = (v'*v)/trace(Rk*(v'*v));

Jex=zeros(1,N);
Jms= zeros(1,N);

%% Maximum step-size

if M==1

    k1 = Mom1;
    k2=0;
    k3=0;
    k4=0;
else

    k1 = Mom1 + (M-1)*Mom3;
    k2 = 2*(M-1)*Mom2 + (M-1)*(M-2)*Mom4;
    k3 = 2*Mom2 +(M-2)*Mom4;
    k4 = 4*(M-2)*Mom4 + 2*Mom3 + (M-3)*(M-2)*Mom5;

end


for n=1:N
    Jex(n)= trace(Rk*C_v); % variation of the excess ...
        mean-squared error variating
    Jms(n) = Jmin + Jex(n);

    T=zeros(M,M);
    x=1;
    for i=1:M
        for j=i:M

            if i==j

                T(i,j) = trace(ATmd(:,:,i)*C_v);
```

```
103
104             else
105                 T(i,j) = trace(ATod(:,:,x)*C_v);
106                 T(j,i) = trace(ATod(:,:,x)*C_v);
107                 x=x+1;
108             end
109         end
110     end
111
112     C_v = C_v - eta*(Rk*C_v + C_v*Rk) + (eta^2)*T + ...
            (eta^2)*Jmin*Rk;
113
114 end
115
116 %% Steady state
117
118 NF = eta*Jmin*(rod*(2*rmd-eta*k1) - rmd*(2*rod-eta*k3));
119 DF =(eta*k2-2*(M-1)*rod)*(2*rod-eta*k3) - (eta*k4 - ...
        2*(rmd+(M-2)*rod))*(2*rmd-eta*k1);
120
121 F=NF/DF;
122 E = (F*(eta*k2-2*(M-1)*rod) + eta*rmd*Jmin)/(2*rmd-eta*k1);
123
124 Cv_ss = (E-F)*eye(M) + (F)*ones(M);
125
126 Jex_ss = M*(rmd*E + (M-1)*rod*F);
127
128 Jms_ss = Jex_ss+ Jmin;
129
130
131
132 %% time
133
134 C_v0 = (v'*v)/trace(Rk*(v'*v));
135
136 c0 = C_v0(1:M^2);
137
```

```
[G,H] = makeG(M,Mom1,Mom2,Mom3,Mom4,Mom5, eta, rmd, rod);

r = Rk(1:M^2)';

c_inf = (eta^2)*Jmin*((eye(M^2)-G)\r);


N_inf = time_conv2(M,Rk,Cv_ss, c0,c_inf,epsilon,G);

C_vinf = zeros(M);
t=1;
for i=1:M
    for j=1:M

        C_vinf(i,j) = c_inf(t);
        t=t+1;
    end
end

Jex_inf = trace(C_vinf*Rk);
Jms_inf = Jex_inf + Jmin;

%save ./gaussian_results/mod_res.mat Jms Jms_inf Jex ...
    Jex_inf Jmin C_v M N eta N_inf

figure
plot(1:N, 10*log10(Jmin*ones(1,N)),'r', 1:N, ...
    10*log10(Jms(1:N)), 'b', ...
1:N, 10*log10(Jex(1:N)), 'k', 1:N, ...
    10*log10(Jms_inf*ones(1,N)), '--b', ...
1:N, 10*log10(Jex_inf*ones(1,N)),'--k' )
xlabel('\bf iterations','FontSize',18 )
ylabel('\bf MSE [dB]','FontSize',18)
legend(['Minimum MSE'],['MSE'],['Excess ...
    MSE'],['Steady-State MSE'],['Steady-State Excess MSE'])
grid
```

```
%================================================
% Performance analysis of Kernel LMS – Model of the ...
    statistic behavior
%================================================
% KLMS Gaussian on the Nonlinear System Identification
% Experiment from: Example 2
%================================================

close all
clc
clear all

it_dic =500;
rlz = 500;

su = 0.125;
rho = 0.5;
sw = 1e-3;
dms= 2;
nc_v= 1e-4;
Jmax= -20.25;
epsilon= 1e-3;

cte = input('set the kernel bandwidth of Gaussian ...
    kernel: '); % set cte

dic_length = dic_behavior(it_dic, rlz, su, rho, dms, ...
    nc_v, 'gaussian', cte);

M = dic_length;

Jmin = MMSE_behav_gauss('example1', rlz, 100, sw, su, ...
    dms, rho, nc_v, M, cte);

disp('||    kernel param   |    M     |   eta   | ...
    Jmin(dB)   |   Jmse(dB)   | Jex(dB)   | N_inf   ||')
```

```
Results = klms_gauss_ss(Jmin, Jmax, M, su, dms, rho, ...
    cte, epsilon);

eta = Results(:,3);

N = input('set the number of iteration: '); % select N < ...
    Results(1,7)




%% Moments of the input signal

[Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] = ...
    mom_gaussiano(dms, M, su,rho, cte);

Rk = (rmd-rod)*eye(M)+rod*ones(M);

%% Eigenvalue and eigenvector - Correlation Matrix
[Theta,Lambda] = eig(Rk);    % Lambda is the diagonal ...
    matrix of eigenvalues and Theta is a full matrix ...
    whose columns are the corresponding eigenvectors

lambda_max = rmd + (M-1)*rod; % maximum eigenvalue
lambda_min = rmd - rod; % minimum eigenvalue

lam = [lambda_min*ones(1,M-1) lambda_max];

%% Matrix Operator calculus

if M==1
    ATmd = Mom1;
else

    [ATmd, ATod] = M_Mom(M,Mom1,Mom2,Mom3,Mom4,Mom5);
end

```

```
64
65 %% Mean-Square Error: Transient and Steady-State
66 % ------ Variables initialization
67 v=1:M;
68 C_v = (v'*v)/trace(Rk*(v'*v));
69
70 Jex=zeros(1,N);
71 Jms= zeros(1,N);
72
73 %% Maximum step-size
74
75 if M==1
76
77     k1 = Mom1;
78     k2=0;
79     k3=0;
80     k4=0;
81 else
82
83     k1 = Mom1 + (M-1)*Mom3;
84     k2 = 2*(M-1)*Mom2 + (M-1)*(M-2)*Mom4;
85     k3 = 2*Mom2 +(M-2)*Mom4;
86     k4 = 4*(M-2)*Mom4 + 2*Mom3 + (M-3)*(M-2)*Mom5;
87
88 end
89
90
91 for n=1:N
92     Jex(n)= trace(Rk*C_v); % variation of the excess ...
          mean-squared error variating
93     Jms(n) = Jmin + Jex(n);
94
95     T=zeros(M,M);
96     x=1;
97     for i=1:M
98         for j=i:M
99
```

```
            if i==j

                T(i,j) = trace(ATmd(:,:,i)*C_v);

            else
                T(i,j) = trace(ATod(:,:,x)*C_v);
                T(j,i) = trace(ATod(:,:,x)*C_v);
                x=x+1;
            end
        end
    end

    C_v = C_v - eta*(Rk*C_v + C_v*Rk) + (eta^2)*T + ...
        (eta^2)*Jmin*Rk;

end

%% Steady state

NF = eta*Jmin*(rod*(2*rmd-eta*k1) - rmd*(2*rod-eta*k3));
DF =(eta*k2-2*(M-1)*rod)*(2*rod-eta*k3) - (eta*k4 - ...
    2*(rmd+(M-2)*rod))*(2*rmd-eta*k1);

F=NF/DF;
E = (F*(eta*k2-2*(M-1)*rod) + eta*rmd*Jmin)/(2*rmd-eta*k1);

Cv_ss = (E-F)*eye(M) + (F)*ones(M);

Jex_ss = M*(rmd*E + (M-1)*rod*F);

Jms_ss = Jex_ss+ Jmin;



%% time

C_v0 = (v'*v)/trace(Rk*(v'*v));
```

```
135
136 c0 = C_v0(1:M^2);
137
138 [G,H] = makeG(M,Mom1,Mom2,Mom3,Mom4,Mom5, eta, rmd, rod);
139
140 r = Rk(1:M^2)';
141
142 c_inf = (eta^2)*Jmin*((eye(M^2)-G)\r);
143
144
145 N_inf = time_conv2(M,Rk,Cv_ss, c0,c_inf,epsilon,G);
146
147 C_vinf = zeros(M);
148 t=1;
149 for i=1:M
150     for j=1:M
151
152         C_vinf(i,j) = c_inf(t);
153         t=t+1;
154     end
155 end
156
157 Jex_inf = trace(C_vinf*Rk);
158 Jms_inf = Jex_inf + Jmin;
159
160 %save ./gaussian_results/mod_res.mat Jms Jms_inf Jex ...
        Jex_inf Jmin C_v M N eta N_inf
161
162 figure
163 plot(1:N, 10*log10(Jmin*ones(1,N)), 'r', ...
        1:N,10*log10(Jms(1:N)), 'b', 1:N, ...
        10*log10(Jex(1:N)), 'k', 1:N, ...
        10*log10(Jms_inf*ones(1,N)), '--b', 1:N,      ...
        10*log10(Jex_inf*ones(1,N)), '--k')
164 xlabel('\bf iterations','FontSize',18 )
165 ylabel('\bf MSE [dB]','FontSize',18)
166 legend(['Minimum MSE'],['MSE'],['Excess ...
```

```
    MSE'],['Steady-State MSE'],['Steady-State Excess MSE'])
grid
```

```
%================================================
% Performance analysis of Kernel LMS - Model of the ...
    statistic behavior
%================================================
% KLMS Gaussian on the Nonlinear System Identification
% Experiment from: Example 3
%================================================

close all
clc
clear all

it_dic =500;
rlz = 500;

su = 0.25;
rho = 0.5;
sw = 1e-2;
dms= 2;
nc_v= 1e-1;
Jmax= -19.75;
epsilon= 1e-4;

cte = input('set the constant term of Gaussian kernel: ...
    '); % set cte

dic_length = dic_behavior(it_dic, rlz, su, rho, dms, ...
    nc_v, 'gaussian', cte);

M = dic_length;

Jmin = MMSE_behav_gauss('example3', rlz, 100, sw, su, ...
```

```
    dms, rho, nc_v, M, cte);

disp('||    kernel param   |    M     |   eta   | ...
    Jmin(dB)   |   Jmse(dB)   | Jex(dB)   | N_inf   ||')
Results = klms_gauss_ss(Jmin, Jmax, M, su, dms, rho, ...
    cte, epsilon);

eta = Results(:,3);

N = input('set the number of iteration: '); % select N < ...
    Results(1,7)




%% Moments of the input signal

[Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] = ...
    mom_gaussiano(dms, M, su,rho, cte);

Rk = (rmd-rod)*eye(M)+rod*ones(M);

%% Eigenvalue and eigenvector - Correlation Matrix
[Theta,Lambda] = eig(Rk);    % Lambda is the diagonal ...
    matrix of eigenvalues and Theta is a full matrix ...
    whose columns are the corresponding eigenvectors

lambda_max = rmd + (M-1)*rod; % maximum eigenvalue
lambda_min = rmd - rod; % minimum eigenvalue

lam = [lambda_min*ones(1,M-1) lambda_max];

%% Matrix Operator calculus

if M==1
    ATmd = Mom1;
else
```

```
60
61      [ATmd, ATod] = M_Mom(M,Mom1,Mom2,Mom3,Mom4,Mom5);
62  end
63
64
65  %% Mean-Square Error: Transient and Steady-State
66  % ------ Variables initialization
67  v=1:M;
68  C_v = (v'*v)/trace(Rk*(v'*v));
69
70  Jex=zeros(1,N);
71  Jms= zeros(1,N);
72
73  %% Maximum step-size
74
75  if M==1
76
77      k1 = Mom1;
78      k2=0;
79      k3=0;
80      k4=0;
81  else
82
83      k1 = Mom1 + (M-1)*Mom3;
84      k2 = 2*(M-1)*Mom2 + (M-1)*(M-2)*Mom4;
85      k3 = 2*Mom2 +(M-2)*Mom4;
86      k4 = 4*(M-2)*Mom4 + 2*Mom3 + (M-3)*(M-2)*Mom5;
87
88  end
89
90
91  for n=1:N
92      Jex(n)= trace(Rk*C_v); % variation of the excess ...
            mean-squared error variating
93      Jms(n) = Jmin + Jex(n);
94
95      T=zeros(M,M);
```

```
    x=1;
    for i=1:M
        for j=i:M

            if i==j

                T(i,j) = trace(ATmd(:,:,i)*C_v);

            else
                T(i,j) = trace(ATod(:,:,x)*C_v);
                T(j,i) = trace(ATod(:,:,x)*C_v);
                x=x+1;
            end
        end
    end

    C_v = C_v - eta*(Rk*C_v + C_v*Rk) + (eta^2)*T + ...
        (eta^2)*Jmin*Rk;

end

%% Steady state

NF = eta*Jmin*(rod*(2*rmd-eta*k1) - rmd*(2*rod-eta*k3));
DF =(eta*k2-2*(M-1)*rod)*(2*rod-eta*k3) - (eta*k4 - ...
    2*(rmd+(M-2)*rod))*(2*rmd-eta*k1);

F=NF/DF;
E = (F*(eta*k2-2*(M-1)*rod) + eta*rmd*Jmin)/(2*rmd-eta*k1);

Cv_ss = (E-F)*eye(M) + (F)*ones(M);

Jex_ss = M*(rmd*E + (M-1)*rod*F);

Jms_ss = Jex_ss+ Jmin;


```

```
131
132 %% time
133
134 C_v0 = (v'*v)/trace(Rk*(v'*v));
135
136 c0 = C_v0(1:M^2);
137
138 [G,H] = makeG(M,Mom1,Mom2,Mom3,Mom4,Mom5, eta, rmd, rod);
139
140 r = Rk(1:M^2)';
141
142 c_inf = (eta^2)*Jmin*((eye(M^2)-G)\r);
143
144
145 N_inf = time_conv2(M,Rk,Cv_ss, c0,c_inf,epsilon,G);
146
147 C_vinf = zeros(M);
148 t=1;
149 for i=1:M
150     for j=1:M
151
152         C_vinf(i,j) = c_inf(t);
153         t=t+1;
154     end
155 end
156
157 Jex_inf = trace(C_vinf*Rk);
158 Jms_inf = Jex_inf + Jmin;
159
160 %save ./gaussian_results/mod_res.mat Jms Jms_inf Jex ...
      Jex_inf Jmin C_v M N eta N_inf
161
162 figure
163 plot(1:N, 10*log10(Jmin*ones(1,N)), 'r', ...
      1:N,10*log10(Jms(1:N)),'b', 1:N, 10*log10(Jex(1:N)), ...
      'k', 1:N, 10*log10(Jms_inf*ones(1,N)),'--b', 1:N, ...
            10*log10(Jex_inf*ones(1,N)), '--k' )
```

```
xlabel('\bf iterations','FontSize',18 )
ylabel('\bf MSE [dB]','FontSize',18)
legend(['Minimum MSE'],['MSE'],['Excess ...
    MSE'],['Steady-State MSE'],['Steady-State Excess MSE'])
grid
```

H.3 Kernel polinomial

```
%=================================================
% BDICTIONARY  creates a dictionary from the parameters ...
    requested.
% dictionary = bdictinary_polynom(degree,cte,lim_coh, ...
    uc0) returns an N-by-M matrix
% containing the sparse dictionary obtained from lim_coh ...
    (threshold coherence) and uc0 ( input signal ).
% In this case M is the dictionary length and N is the ...
    length of each component
% of the dictionary.
%=================================================

function dictionary = ...
    bdictinary_polynom(degree,cte,lim_coh, uc0)

    % dictionary initialization
    dictionary = uc0(:,1);
    [dms,N] = size(uc0);
    n0 = 2;
    fora = 0;

while n0 ≤ N && fora < N/3

        Dic_t = [uc0(:,n0) dictionary];
        [dms MM]= size(Dic_t);

```

```
        K0 = zeros(MM);

        for l0=1:MM

            for c0=1:MM

                inner_prod = Dic_t(:,c0)'*Dic_t(:,l0);

                %ki = ((cte + ...
                    Dic_t(:,l0)'*Dic_t(:,l0))^degree);
                %kj = ((cte + ...
                    Dic_t(:,c0)'*Dic_t(:,c0))^degree);

                K0(l0,c0) = ((cte + ...
                    inner_prod)^degree);%/sqrt(ki*kj);

            end
        end

        [bab0,coh0] = babel(K0);

        if coh0 ≤ lim_coh

           dictionary = Dic_t;
           fora = 0;
        else
            fora = 1 + fora;
        end

        n0 = n0+1;


end
```

```
%================================================
```

```
% function [Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, ...
    Mom5] = mom_polynomial(M, su, dms, rho, cte)
%
% For the polynomial kernel of 2nd order, the ...
    statistical moments
% 1st order moment : Expv
% 2nd order moments : rmd (main-diagonal of Rkk) and rod ...
    (off-diagonal of Rkk)
% 4th order moments : Mom1 (mu_1), Mom2 (mu_2), ... Mom5 ...
    (mu_5)
%
% are obtained from this function based on inputs:
% M    : dictionary size
% su   : standard deviation of input signal
% rho  : correlation level
% cte  : constant term of polynomial kernel of 2nd order.
%==================================================

function [Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] ...
    = mom_polynomial(M, su, dms, rho, cte)

if dms == 2
    s12 = rho*(su^2);

elseif dms == 1
    s12=0;
elseif dms ≥ 3
    disp('Error! rewrite this program')
end

    %%  Expected value - E{kw(n)}
    Expv = cte + 2*(su^2)^2 + 2*s12^2 ;

    %% Correlation matrix - E {kw(n)*k^Tw(n)}

    rmd = cte^4 + 12*(cte^2)*(su^4 + (s12^2)) + ...
        2*(3*su^4)^2 + 8*(3*s12*(su^2))^2 + 6*(su^4 + ...
```

```
        2*(s12^2))^2;

rod = cte^4 + 4*(cte^2)*(su^4 + (s12^2)) + 6*(su^8) ...
    + 24*(su^4)*(s12^2) + 2*(su^4 + 2*(s12^2))^2;

%% 4th order moment of the Polynomial kernel

s40 = 3*su^4;
s31 = 3*s12*su^2;
s22 = su^4 + 2*s12^2;
%-------------------------------------------------------
s60 = 15*(su^6);
s51 = 15*(su^4)*s12;
s42 = 3*(su^6) + 12*(su*s12)^2;
s33 = 9*s12*(su^4) + 6*(s12^3);
%-------------------------------------------------------
s80= 105*su^8;
s71= 105*s12*su^6;
s62= 15*su^8 + 90*(su^4)*(s12^2);
s53=  60*(su^2)*(s12^3) + 45*(su^6)*(s12^1);
s44= 9*su^8 + 72*(su^4)*(s12^2) + 24*s12^4;
%-------------------------------------------------------
S_80 = 2*(s80^2) + 16*(s71^2) + 28*(s62^2) + ...
    112*(s53^2) + 70*(s44^2);
%-------------------------------------------------------
S_62 = 30*(su^4)*(s80 +s62) + 60*s12*(su^2)*s71 + ...
    180*(su^6)*s12*(s71 + s53) + 360*(su^4)*s12*s62 ...
    + (90*(su^8) + 360*(su^4)*(s12^2))*(s62+s44) + ...
    (180*s12*su^6 + 720*(su^2)*(s12^3))*s53 + ...
    (360*(su^4)*(s12^2) + 240*s12^4)*s44;

S_61 = 0;

S_60 = 2*(s60)^2 + 12*(s51)^2 + 30*(s42)^2 + 20*(s33)^2;
%-------------------------------------------------------
S_52 = 0;
S_51 = 0;
```

```
63      S_50 = 0;
64      %-------------------------------------------------------
65      S_44 = 18*(su^8)*(s80+ s44) + 288*(su^4)*s12*(s62 ...
            +s44) + 36*((su^4 + s12^2)^2)*s44 + ...
            144*(su^6)*s12*(s71 + s53) + 72*(su^4)*(su^4 + ...
            s12^2)*s62 + 288*(su^2)*s12*(su^4 + s12^2)*s53;
66
67      S_42 = 6*(su^6)*(s60 + s42) + 12*s12*(su^4)*s51 + ...
            24*(su^4)*s12*(s51 + s33) + 12*(su^6 + ...
            (su^2)*(s12^2))*s42 + 48*(su^2)*(s12^2)*s42 + ...
            12*s12*((su^4) + s12^2)*s33;
68
69      S_41 = 0;
70
71      S_40 = 2*(s40)^2 + 8*(s31)^2 + 6*(s22)^2;
72      %-------------------------------------------------------
73      %-------------------------------------------------------
74      S_32 = 0;
75      S_31 = 0;
76      S_30 = 0;
77      %-------------------------------------------------------
78      S_22 = 8*(su^8) + 8*(s12^4) + 32*(su^4)*(s12^2);
79
80      S_21 = 0;
81
82      S_20 =2*su^4 + 2*s12^2;
83      %-------------------------------------------------------
84      S_12 = 0;
85      S_11 = 0;
86      S_10 = 0;
87      %-------------------------------------------------------
88      %-------------------------------------------------------
89      S_02 = S_20;
90      S_01 = S_10;
91      S_00 = 1;
92      %-------------------------------------------------------
93
```

```
94      %-------------------------------------------------------
95      % case: i=j=l=p
96
97      Mom1 = cte^8 + 14*(cte^6)*S_20 + 70*(cte^4)*S_40 + ...
            14*(cte^2)*S_60 + S_80;
98      %-------------------------------------------------------
99
100     %-------------------------------------------------------
101     % case: (i=j=p≠l) or (i=j=l≠p) or (j=l=p≠i) or (i=l=p≠j)
102     aux0 = (cte^0)*S_62 + 2*cte*S_61 + (cte^2)*S_60;
103     aux1 = (cte^1)*S_52 + 2*(cte^2)*S_51 + (cte^3)*S_50;
104     aux2 = (cte^2)*S_42 + 2*(cte^3)*S_41 + (cte^4)*S_40;
105     aux3 = (cte^3)*S_32 + 2*(cte^4)*S_31 + (cte^5)*S_30;
106     aux4 = (cte^4)*S_22 + 2*(cte^5)*S_21 + (cte^6)*S_20;
107     aux5 = (cte^5)*S_12 + 2*(cte^6)*S_11 + (cte^7)*S_10;
108     aux6 = (cte^6)*S_02 + 2*(cte^7)*S_01 + (cte^8)*S_00;
109
110     Mom2 = aux0 + 6*aux1 + 15*aux2 + 20*aux3 + 15*aux4 + ...
            6*aux5 + aux6;
111
112     %-------------------------------------------------------
113     % case: (i=j ≠ p=l) or (i=p ≠ l=j) or (i=l ≠ p=j)
114     Mom3 = (1 + cte^4)*S_44 + (12*cte^2)*S_42 + ...
            (cte^4)*S_40 + (6*cte^6 + 36*cte^4)*S_22 + ...
            (12*cte^6)*S_20 + cte^8;
115
116     %-------------------------------------------------------
117     % case: (i=j and ≠ p ≠ l) or (i=p and ≠ j ≠l) or ...
            (i=l and ≠ j ≠ p)
118     % (j=p and ≠ i ≠ p) or (j=l and ≠ i ≠ p) or (l=p and ...
            ≠ i ≠ j)
119     S_222 = 2*(su^6)*(s60 + 3*s42) + 12*s12*(su^4)*(s51 ...
            + s33) + 24*(su^2)*(s12^2)*s42 + 8*(s12^3)*s33;
120
121     S_422 = 6*(su^8)*(s80 + s44 + 2*s62) + ...
            24*(su^6)*s12*(s71 + 3*s53) + 12*(su^4)*(su^6 + ...
            s12^2)*(s62 + s44) + 24*(su^6)*(s12)*(s71+s53) + ...
```

```
        96*(su^4)*(s12^2)*(s62+s44) + ...
        48*(su^2)*(s12)*(su^6 + s12^2)*s53 + ...
        24*(su^4)*(s12^2)*s62 + 96*(su^2)*(s12^3)*s53 + ...
        24*(s12^2)*(su^6 + s12^2)*s44;


    Mom4 = (cte^8) + (cte^6)*(2*S_20) + (cte^4)*(2*S_22) ...
        + 6*(cte^6)*(S_20) + 12*(cte^4)*(S_22) + ...
        6*(cte^2)*S_222 + (cte^4)*S_40 + 2*(cte^2)*S_42 ...
        + S_422;
    %-------------------------------------------------------
    % case: i≠j≠p≠l
    aux00 = cte^8 + 8*(cte^6)*(su^4) + 8*(cte^6)*(s12^2);
    aux44 = s31*(48*(cte^4)*s12*su^2) + ...
        s40*12*(cte*su)^4 + s22*(24*(cte^4)*(s12^2) + ...
        12*(cte^4)*(su^4));
    aux66 = s51*(48*(cte^2)*(su^4)*s12) + ...
        s33*16*(cte^2)*(2*su^2*s12^2 +  s12*su^4 + ...
        2*s12^3) + s60*8*(cte*su)^2 + s42*(24*(cte*su)^2 ...
        + 32*(cte*su*s12)^2 + 64*(su*s12)^2);
    aux88 = s62*8*(su^4)*(su^4 + 8*s12^2) + ...
        s44*(48*(su^4)*(s12^2) + 6*su^8 + 16*s12^4 ) + ...
        s71*16*s12*su^6 + s53*8*(su^2)*s12*(6*su^4 + ...
        8*s12) + s80*2*su^8;


    Mom5 = aux00 + aux44 + aux66 + aux88;

    if M==1
        rod = 0;
        Mom2=0;
        Mom3=0;
        Mom4=0;
        Mom5=0;

    elseif M==2;

```

```
144         Mom4=0;
145         Mom5=0;
146
147     elseif M==3;
148         Mom5=0;
149     end
```

```
1  %================================================
2  % This function was used to estimate the ...
       crosscorrelation of the desired
3  % signal and inputsignal presented in the Simulation ...
       Results of the thesis.
4  %================================================
5  % function corr_dn = cross_stat_poly(example,rlz, N, sw, ...
       su, rho, lim_coh, M, cte);
6  %
7  % inputs of the function
8  % example  : desired signal of example
9  % rlz      : number of Monte carlo realization
10 % N        : number of iteration
11 % sw       : noise standard deviation
12 % su       : u_1(n) standard deviation
13 % rho      : correlation level of AR(1) process
14 % lim_coh  : coherence level
15 % M        : dictionary size
16 % cte      : contant term of polynomial kernel
17 %
18 %  Values for example: 'example1'  – desired signal of ...
       example 1.
19 %                      'example2'  – desired signal of ...
       example 2.
20 %                      'example3'  – desired signal of ...
       example 3.
21 %================================================
22
```

```
function Pkd_stat = cross_stat_poly(example,rlz, N, sw, ...
    su, rho, lim_coh, M, cte)

deg = 2;

%% Simulation

NN=N+2;

N_dic=500;

% Variables initialization
Pkd = zeros(M,1);

for r=1:rlz
    M0=1000;

    switch lower(example)

    case 'example1'


        while M0≠ M
            u_dic = su*randn(1, N_dic);

            Dic0 = bdictinary_polynom(deg,cte,lim_coh, ...
                u_dic); % function that build the dicitonary

            [dms,M0] = size(Dic0);

        end

        Dic = Dic0;


        %% Variables initialization

```

```
58          un = zeros(dms,NN);
59
60          yn = zeros(NN,1);
61          dn = zeros(NN,1);
62          d= zeros(NN,1);
63
64          for n = 2:NN
65
66              un(1,n) = su*randn(1,1);
67
68              yn(n,1) = (un(n-1))^3 + (yn(n-1,1))/(1 + ...
                    (yn(n-1,1))^2);
69
70              dn(n,1) = yn(n,1);
71
72              d(n,1) = dn(n,1) + sw*randn(1,1);
73
74              %% Computation of the Kernel gram matrix
75              Dic_t = [un(:,n) Dic];
76              K = zeros(M,1);
77
78              for l=1:M
79                  i_prod = un(:,n)'*Dic(:,l);
80
81                  K(l,1) = ((cte + i_prod)^deg);
82
83
84              end
85
86              Dic = Dic_t(:,1:M);
87          end
88          Pkd = Pkd + K*d(NN);
89
90          case 'example2'
91
92              while M0~= M
93                  u_dic = zeros(2, N_dic);
```

```
94              for n=1:N_dic
95                  u_dic(1,n) = su*randn(1,1);
96                  u_dic(2,n) = rho*u_dic(1,n) + ...
                        su*sqrt(1-rho^2)*randn(1,1);
97              end
98
99              Dic0 = ...
                    bdictinary_polynom(deg,cte,lim_coh, ...
                    u_dic); % function that build the ...
                    dicitonary
100
101             [dms,M0] = size(Dic0);
102
103         end
104
105         Dic = Dic0;
106
107
108         %% Variables initialization
109
110         un = zeros(dms,N+3);
111
112         yn = zeros(NN,1);
113         dn = zeros(NN,1);
114         d= zeros(NN,1);
115
116         A = [1; 0.5];
117
118
119         for n = 3:NN
120
121             un(1,n) = su*randn(1,1);
122             un(2,n) = rho*un(1,n) + ...
                    su*sqrt(1-rho^2)*randn(1,1);
123
124             yn(n,1) = A'*un(:,n) - 0.2*yn(n-1,1) + ...
                    0.35*yn(n-2,1) ;
```

```
125
126                 if yn(n,1) ≥ 0
127
128                     dn(n,1) = yn(n,1)/(3*(0.1 + ...
                            0.9*(yn(n,1))^2)^(1/2));
129
130                 else
131
132                     dn(n,1) = ...
                            −(yn(n,1)^2)*(1−exp(0.7*yn(n,1)))/3;
133
134                 end
135
136                 d(n,1) = dn(n,1) + sw*randn(1,1);
137
138                 %% Computation of the Kernel gram matrix
139                 Dic_t = [un(:,n) Dic];
140                 K = zeros(M,1);
141
142                 for l=1:M
143
144                     i_prod = un(:,n)'*Dic(:,l);
145                     K(l,1) = ((cte + i_prod)^deg);
146
147
148                 end
149
150                 Dic = Dic_t(:,1:M);
151             end
152             Pkd = Pkd + K*d(NN);
153
154             case 'example3'
155
156                 while M0≠ M
157                     u_dic = zeros(2, N_dic);
158                     for n=1:N_dic
159                         u_dic(1,n) = su*randn(1,1);
```

```
160                     u_dic(2,n) = rho*u_dic(1,n) + ...
                            su*sqrt(1-rho^2)*randn(1,1);
161                 end
162
163                 Dic0 = bdictinary_polynom( deg, cte, ...
                        lim_coh, u_dic ); % function ...
                        that build the dicitonary
164
165                 [dms,M0] = size(Dic0);
166
167             end
168
169             Dic = Dic0;
170
171
172             %% Variables initialization
173
174             un = zeros(dms,N+3);
175
176             yn = zeros(NN,1);
177             dn = zeros(NN,1);
178             d= zeros(NN,1);
179
180             A = [0.1044; 0.0883];
181
182
183             for n = 3:NN
184
185                 un(1,n) = su*randn(1,1);
186                 un(2,n) = rho*un(1,n) + ...
                        su*sqrt(1-rho^2)*randn(1,1);
187
188                 yn(n,1) = A'*un(:,n) + ...
                        1.4138*yn(n-1,1) - ...
                        0.6065*yn(n-2,1) ;
189                 dn(n,1) = 0.3163*yn(n,1)/(sqrt(0.10 ...
                        +0.9*yn(n,1)^2));
```

```
190
191                     d(n,1) = dn(n,1) + sw*randn(1,1);
192
193                     %% Computation of the Kernel gram matrix
194                     Dic_t = [un(:,n) Dic];
195                     K = zeros(M,1);
196
197                     for l=1:M
198                         i_prod = un(:,n)'*Dic(:,l);
199
200                         K(l,1) = ((cte + i_prod)^deg);
201
202
203                     end
204
205                     Dic = Dic_t(:,1:M);
206                 end
207
208             Pkd = Pkd + K*d(NN);
209     end
210
211 end
212
213 Pkd_stat = Pkd/rlz;
```

```
1 %================================================
2 % Jmin_v = MMSE_behav_poly(example, rlz, N, sw, dms, su, ...
      rho, lim_coh,
3 % M_med, cte_v)
4 % This function computs the behavior of Minimum MSE
5 %
6 % Define the tests parameters
7 % example  : desired signal of example
8 % rlz      : number of Monte carlo realization
9 % N        : number of iteration
```

```
% sw       : noise standard deviation
% su       : u_1(n) standard deviation
% rho      : correlation level of AR(1) process
% lim_coh  : coherence level
% M_       : vector (or scalar) of dictionary size
% cte_v    : vector (or scalar) of term constant of ...
    polynomial kernel
%
% and the outputs:
% the behavior of MMSE, which is Jmin_v.
%=================================================

function Jmin_v = MMSE_behav_poly(example, rlz, N, sw, ...
    su, dms,rho, lim_coh, M_med, cte_v)

Corr_dn = autocorrelation_dn(example,rlz, N, sw, su, rho);

Jmin_v = zeros(1,length(cte_v));

for p=1:length(cte_v)

    M= M_med(p);
    cte = cte_v(p);

    Pkd_stat = cross_stat_poly(example,rlz, N, sw, su, ...
        rho, lim_coh, M, cte);
    %% Model
    % Correlation matrix — kw(n)
   [Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] = ...
       mom_polynomial(M, su, dms, rho, cte);

    Rk = (rmd—rod)*eye(M)+rod*ones(M);
    % Wiener solution
    alpha_ot = Rk\Pkd_stat; % wiener solution
    Jmin_v(p) = Corr_dn — Pkd_stat'*alpha_ot;  % Minimum ...
        error power

```

```
42 end
```

```
1  %==================================================
2  % function [sumario] = stab_test_polynom ...
       (M_med,su,rho,cte_v)
3  %
4  % Stability test of
5  % c(n+1) = Gc(n) + eta^2 r Jmin
6  % using the Gerschgorin disc test and eigenvalues of ...
       matrix G test.
7  % Input:
8  % M_med : vector (or scalar) dictionary size
9  % su    : standard deviation of input signal u(n)
10 % rho   : correlation level of u(n)
11 % cte_v : vector (or scalar) constant term of polynomial ...
       kernel with degree
12 %         2
13 % Output
14 % sumario : is a matrix (or vector) when
15 %
16 % 1st column : constant term of the polynomial kernel
17 % 2nd column : dictionary size
18 % 3rd column : autocorrelation of kernel
19 % 4th column : crosscorrelation of kernel
20 % 5th — 9th column : 4th order moments
21 % 10th column: maximum step—size by eigenvalues of G test
22 % 11th column: maximum step—size by Gerschgorin disc test
23 %==================================================
24
25 function [sumario] = stab_test_polynom ...
       (M_med,su,dms,rho,cte_v)
26
27 nx = length(cte_v);
28 sumario = zeros(nx,11);
29
```

```
for l=1:nx

    cte = cte_v(l);
    M = M_med(l);

    [Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] = ...
        mom_polynomial(M, su, dms, rho, cte);


    %% Maximum step-size


    eta_max_modelo = max_stepsize(M, rmd, rod, Mom1, ...
        Mom2, Mom3, Mom4, Mom5);


    eta = 1e-5;
    s1 = M^2;
    s2 = s1;

    while (s1 == M^2) &&  (s2 == M^2)


        [G,H] = makeG(M,Mom1,Mom2,Mom3,Mom4,Mom5, eta, ...
            rmd, rod);

        clear lam_G test1 test2

        lam_G = eig(G);

        test1 = lam_G < 1;
        test2 = lam_G >-1;

        s1 = sum(test1);
        s2 = sum(test2);

        eta_simul = eta;
```

```
64
65          eta = eta + 1e-5;
66      end
67
68      sumario(l,:) = [cte M rmd rod Mom1 Mom2 Mom3 Mom4 ...
            Mom5  eta_simul eta_max_modelo] ;
69
70
71  end
72
73  indc = find(sumario==-inf);
74
75  for p=1:length(indc)
76
77      sumario(indc(p))=NaN;
78
79  end
```

```
1
2   %=================================================
3   % Performance analysis of Kernel LMS - Model of the ...
        statistic behavior
4   %=================================================
5   % KLMS Polynomial on the Nonlinear System Identification
6   % Experiment from: Example 1
7   %=================================================
8
9   close all
10  clc
11  clear all
12
13  it_dic =500;
14  rlz = 500;
15
16  su = 0.15;
```

```
17 rho = 0.5;
18 sw = 1e-2;
19 dms= 1;
20 nc_v= 0.7;
21 Jmax= -22.35;
22 epsilon= 2.5*1e-4;
23
24 cte = input('set the constant term of Polynomial kernel: ...
       '); % set cte
25
26 dic_length = dic_behavior(it_dic, rlz, su, rho, dms, ...
       nc_v, 'polynomial2', cte);
27
28 M = dic_length;
29
30 Jmin = MMSE_behav_poly('example1', rlz, 100, sw, su, ...
       dms, rho, nc_v, M, cte);
31
32 disp('||    kernel param    |    M     |   eta    | ...
       Jmin(dB)   |   Jmse(dB)   | Jex(dB)   | N_inf   ||')
33 Results = klms_poly_ss(Jmin, Jmax, M, su, dms, rho, cte, ...
       epsilon);
34
35 eta = Results(:,3);
36
37 N = input('set the number of iteration: '); % select N < ...
       Results(1,7)
38
39 %% Moments of the input signal
40
41 [Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] = ...
       mom_polynomial(dms, M, su,rho, cte);
42
43 Rk = (rmd-rod)*eye(M)+rod*ones(M);
44
45 %% Eigenvalue and eigenvector - Correlation Matrix
46 [Theta,Lambda] = eig(Rk);    % Lambda is the diagonal ...
```

```
    matrix of eigenvalues and Theta is a full matrix ...
    whose columns are the corresponding eigenvectors

lambda_max = rmd + (M-1)*rod; % maximum eigenvalue
lambda_min = rmd - rod; % minimum eigenvalue


lam = [lambda_min*ones(1,M-1) lambda_max];

%% Matrix Operator calculus

if M==1
    ATmd = Mom1;
else

    [ATmd, ATod] = M_Mom(M,Mom1,Mom2,Mom3,Mom4,Mom5);
end


%% Mean-Square Error: Transient and Steady-State
% ------ Variables initialization
v=1:M;
C_v = (v'*v)/trace(Rk*(v'*v));

Jex=zeros(1,N);
Jms= zeros(1,N);

%% Maximum step-size

if M==1

    k1 = Mom1;
    k2=0;
    k3=0;
    k4=0;
else

```

```
    k1 = Mom1 + (M-1)*Mom3;
    k2 = 2*(M-1)*Mom2 + (M-1)*(M-2)*Mom4;
    k3 = 2*Mom2 +(M-2)*Mom4;
    k4 = 4*(M-2)*Mom4 + 2*Mom3 + (M-3)*(M-2)*Mom5;

end


for n=1:N
    Jex(n)= trace(Rk*C_v); % variation of the excess ...
        mean-squared error variating
    Jms(n) = Jmin + Jex(n);

    T=zeros(M,M);
    x=1;
    for i=1:M
        for j=i:M

            if i==j

                T(i,j) = trace(ATmd(:,:,i)*C_v);

            else
                T(i,j) = trace(ATod(:,:,x)*C_v);
                T(j,i) = trace(ATod(:,:,x)*C_v);
                x=x+1;
            end
        end
    end

    C_v = C_v - eta*(Rk*C_v + C_v*Rk) + (eta^2)*T + ...
        (eta^2)*Jmin*Rk;

end

%% Steady state

```

```
NF = eta*Jmin*(rod*(2*rmd-eta*k1) - rmd*(2*rod-eta*k3));
DF =(eta*k2-2*(M-1)*rod)*(2*rod-eta*k3) - (eta*k4 - ...
    2*(rmd+(M-2)*rod))*(2*rmd-eta*k1);

F=NF/DF;
E = (F*(eta*k2-2*(M-1)*rod) + eta*rmd*Jmin)/(2*rmd-eta*k1);

Cv_ss = (E-F)*eye(M) + (F)*ones(M);

Jex_ss = M*(rmd*E + (M-1)*rod*F);

Jms_ss = Jex_ss+ Jmin;

%% time

C_v0 = (v'*v)/trace(Rk*(v'*v));

c0 = C_v0(1:M^2);

[G,H] = makeG(M,Mom1,Mom2,Mom3,Mom4,Mom5, eta, rmd, rod);

r = Rk(1:M^2)';

c_inf = (eta^2)*Jmin*((eye(M^2)-G)\r);

N_inf = time_conv2(M,Rk,Cv_ss, c0,c_inf,epsilon,G);

C_vinf = zeros(M);
t=1;
for i=1:M
    for j=1:M

        C_vinf(i,j) = c_inf(t);
        t=t+1;
    end
end

```

```
Jex_inf = trace(C_vinf*Rk);
Jms_inf = Jex_inf + Jmin;

%save ./polynomial_results/mod_res.mat Jms Jms_inf Jex ...
    Jex_inf Jmin C_v M N eta N_inf

figure
plot(1:N, 10*log10(Jmin*ones(1,N)),'r', ...
    1:N,10*log10(Jms(1:N)),'b', ...
    1:N,10*log10(Jex(1:N)),'k', ...
    1:N,10*log10(Jms_inf*ones(1,N)), ...
    '--b',1:N,10*log10(Jex_inf*ones(1,N)),'--k' )
xlabel('\bf iterations','FontSize',18 )
ylabel('\bf MSE [dB]','FontSize',18)
legend(['Minimum MSE'],['MSE'],['Excess ...
    MSE'],['Steady-State of MSE'],['Steady-State of ...
    Excess MSE'])
grid
```

```
%=================================================
% Performance analysis of Kernel LMS - Model of the ...
    statistic behavior
%=================================================
% KLMS Polynomial on the Nonlinear System Identification
% Experiment from: Example 2
%=================================================

close all
clc
clear all

it_dic =500;
rlz = 500;

su = 0.125;
```

```
rho = 0.5;
sw = 1e-3;
dms = 2;
nc_v = 0.85;
Jmax = -20.25;
epsilon = 2.5*1e-4;

cte = input('set the constant term of Polynomial kernel: ...
    '); % set cte

dic_length = dic_behavior(it_dic, rlz, su, rho, dms, ...
    nc_v, 'polynomial2', cte);

M = dic_length;

Jmin = MMSE_behav_poly('example2', rlz, 100, sw, su, ...
    dms, rho, nc_v, M, cte);

disp('||    kernel param   |    M     |   eta   | ...
    Jmin(dB)   |   Jmse(dB)   | Jex(dB)   | N_inf   ||')
Results = klms_poly_ss(Jmin, Jmax, M, su, dms, rho, cte, ...
    epsilon);

eta = Results(:,3);

N = input('set the number of iteration: '); % select N < ...
    Results(1,7)

%% Moments of the input signal

[Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] = ...
    mom_polynomial(dms, M, su,rho, cte);

Rk = (rmd-rod)*eye(M)+rod*ones(M);

%% Eigenvalue and eigenvector - Correlation Matrix
[Theta,Lambda] = eig(Rk);    % Lambda is the diagonal ...
```

```
    matrix of eigenvalues and Theta is a full matrix ...
    whose columns are the corresponding eigenvectors

lambda_max = rmd + (M-1)*rod; % maximum eigenvalue
lambda_min = rmd - rod; % minimum eigenvalue


lam = [lambda_min*ones(1,M-1) lambda_max];

%% Matrix Operator calculus

if M==1
    ATmd = Mom1;
else

    [ATmd, ATod] = M_Mom(M,Mom1,Mom2,Mom3,Mom4,Mom5);
end


%% Mean-Square Error: Transient and Steady-State
% ------ Variables initialization
v=1:M;
C_v = (v'*v)/trace(Rk*(v'*v));

Jex=zeros(1,N);
Jms= zeros(1,N);

%% Maximum step-size

if M==1

    k1 = Mom1;
    k2=0;
    k3=0;
    k4=0;
else

```

```
    k1 = Mom1 + (M-1)*Mom3;
    k2 = 2*(M-1)*Mom2 + (M-1)*(M-2)*Mom4;
    k3 = 2*Mom2 +(M-2)*Mom4;
    k4 = 4*(M-2)*Mom4 + 2*Mom3 + (M-3)*(M-2)*Mom5;

end


for n=1:N
    Jex(n)= trace(Rk*C_v); % variation of the excess ...
        mean-squared error variating
    Jms(n) = Jmin + Jex(n);

    T=zeros(M,M);
    x=1;
    for i=1:M
        for j=i:M

            if i==j

                T(i,j) = trace(ATmd(:,:,i)*C_v);

            else
                T(i,j) = trace(ATod(:,:,x)*C_v);
                T(j,i) = trace(ATod(:,:,x)*C_v);
                x=x+1;
            end
        end
    end

    C_v = C_v - eta*(Rk*C_v + C_v*Rk) + (eta^2)*T + ...
        (eta^2)*Jmin*Rk;

end

%% Steady state

```

```
NF = eta*Jmin*(rod*(2*rmd-eta*k1) - rmd*(2*rod-eta*k3));
DF =(eta*k2-2*(M-1)*rod)*(2*rod-eta*k3) - (eta*k4 - ...
    2*(rmd+(M-2)*rod))*(2*rmd-eta*k1);

F=NF/DF;
E = (F*(eta*k2-2*(M-1)*rod) + eta*rmd*Jmin)/(2*rmd-eta*k1);

Cv_ss = (E-F)*eye(M) + (F)*ones(M);

Jex_ss = M*(rmd*E + (M-1)*rod*F);

Jms_ss = Jex_ss+ Jmin;

%% time

C_v0 = (v'*v)/trace(Rk*(v'*v));

c0 = C_v0(1:M^2);

[G,H] = makeG(M,Mom1,Mom2,Mom3,Mom4,Mom5, eta, rmd, rod);

r = Rk(1:M^2)';

c_inf = (eta^2)*Jmin*((eye(M^2)-G)\r);

N_inf = time_conv2(M,Rk,Cv_ss, c0,c_inf,epsilon,G);

C_vinf = zeros(M);
t=1;
for i=1:M
    for j=1:M

        C_vinf(i,j) = c_inf(t);
        t=t+1;
    end
end

```

```
Jex_inf = trace(C_vinf*Rk);
Jms_inf = Jex_inf + Jmin;

%save ./polynomial_results/mod_res.mat Jms Jms_inf Jex ...
    Jex_inf Jmin C_v M N eta N_inf

figure
plot(1:N, 10*log10(Jmin*ones(1,N)),'r', ...
    1:N,10*log10(Jms(1:N)), 'b', 1:N, ...
    10*log10(Jex(1:N)), 'k', 1:N, ...
    10*log10(Jms_inf*ones(1,N)), '--b', 1:N, ...
    10*log10(Jex_inf*ones(1,N)),'--k' )
xlabel('\bf iterations','FontSize',18 )
ylabel('\bf MSE [dB]','FontSize',18)
legend(['Minimum MSE'],['MSE'],['Excess ...
    MSE'],['Steady-State of MSE'],['Steady-State of ...
    Excess MSE'])
grid
```

```
%=================================================
% Performance analysis of Kernel LMS - Model of the ...
    statistic behavior
%=================================================
% KLMS Polynomial on the Nonlinear System Identification
% Experiment from: Example 3
%=================================================

close all
clc
clear all

it_dic =500;
rlz = 500;

su = 0.25;
```

```
rho = 0.5;
sw = 1e-2;
dms = 2;
nc_v = 0.6;
Jmax = -19.75;
epsilon = 5*1e-4;

cte = input('set the constant term of Polynomial kernel: ...
    '); % set cte

dic_length = dic_behavior(it_dic, rlz, su, rho, dms, ...
    nc_v, 'polynomial2', cte);

M = dic_length;

Jmin = MMSE_behav_poly('example3', rlz, 100, sw, su, ...
    dms, rho, nc_v, M, cte);

disp('||    kernel param   |    M     |   eta   | ...
    Jmin(dB)   |   Jmse(dB)   | Jex(dB)   | N_inf   ||')
Results = klms_poly_ss(Jmin, Jmax, M, su, dms, rho, cte, ...
    epsilon);

eta = Results(:,3);

N = input('set the number of iteration: '); % select N < ...
    Results(1,7)




%% Moments of the input signal

[Expv, rmd, rod, Mom1, Mom2, Mom3, Mom4, Mom5] = ...
    mom_polynomial(dms, M, su,rho, cte);

Rk = (rmd-rod)*eye(M)+rod*ones(M);
```

```
46
47 %% Eigenvalue and eigenvector – Correlation Matrix
48 [Theta,Lambda] = eig(Rk);    % Lambda is the diagonal ...
       matrix of eigenvalues and Theta is a full matrix ...
       whose columns are the corresponding eigenvectors
49
50 lambda_max = rmd + (M-1)*rod; % maximum eigenvalue
51 lambda_min = rmd - rod; % minimum eigenvalue
52
53
54 lam = [lambda_min*ones(1,M-1) lambda_max];
55
56 %% Matrix Operator calculus
57
58 if M==1
59     ATmd = Mom1;
60 else
61
62     [ATmd, ATod] = M_Mom(M,Mom1,Mom2,Mom3,Mom4,Mom5);
63 end
64
65
66 %% Mean-Square Error: Transient and Steady-State
67 % ------ Variables initialization
68 v=1:M;
69 C_v = (v'*v)/trace(Rk*(v'*v));
70
71 Jex=zeros(1,N);
72 Jms= zeros(1,N);
73
74 %% Maximum step-size
75
76 if M==1
77
78     k1 = Mom1;
79     k2=0;
80     k3=0;
```

```
    k4=0;
else

    k1 = Mom1 + (M-1)*Mom3;
    k2 = 2*(M-1)*Mom2 + (M-1)*(M-2)*Mom4;
    k3 = 2*Mom2 +(M-2)*Mom4;
    k4 = 4*(M-2)*Mom4 + 2*Mom3 + (M-3)*(M-2)*Mom5;

end


for n=1:N
    Jex(n)= trace(Rk*C_v); % variation of the excess ...
        mean-squared error variating
    Jms(n) = Jmin + Jex(n);

    T=zeros(M,M);
    x=1;
    for i=1:M
        for j=i:M

            if i==j

                T(i,j) = trace(ATmd(:,:,i)*C_v);

            else
                T(i,j) = trace(ATod(:,:,x)*C_v);
                T(j,i) = trace(ATod(:,:,x)*C_v);
                x=x+1;
            end
        end
    end

    C_v = C_v - eta*(Rk*C_v + C_v*Rk) + (eta^2)*T + ...
        (eta^2)*Jmin*Rk;

end
```

```
116
117 %% Steady state
118
119 NF = eta*Jmin*(rod*(2*rmd-eta*k1) - rmd*(2*rod-eta*k3));
120 DF =(eta*k2-2*(M-1)*rod)*(2*rod-eta*k3) - (eta*k4 - ...
        2*(rmd+(M-2)*rod))*(2*rmd-eta*k1);
121
122 F=NF/DF;
123 E = (F*(eta*k2-2*(M-1)*rod) + eta*rmd*Jmin)/(2*rmd-eta*k1);
124
125 Cv_ss = (E-F)*eye(M) + (F)*ones(M);
126
127 Jex_ss = M*(rmd*E + (M-1)*rod*F);
128
129 Jms_ss = Jex_ss+ Jmin;
130
131 %% time
132
133 C_v0 = (v'*v)/trace(Rk*(v'*v));
134
135 c0 = C_v0(1:M^2);
136
137 [G,H] = makeG(M,Mom1,Mom2,Mom3,Mom4,Mom5, eta, rmd, rod);
138
139 r = Rk(1:M^2)';
140
141 c_inf = (eta^2)*Jmin*((eye(M^2)-G)\r);
142
143 N_inf = time_conv2(M,Rk,Cv_ss, c0,c_inf,epsilon,G);
144
145 C_vinf = zeros(M);
146 t=1;
147 for i=1:M
148     for j=1:M
149
150         C_vinf(i,j) = c_inf(t);
151         t=t+1;
```

```
    end
end

Jex_inf = trace(C_vinf*Rk);
Jms_inf = Jex_inf + Jmin;

%save ./polynomial_results/mod_res.mat Jms Jms_inf Jex ...
    Jex_inf Jmin C_v M N eta N_inf

figure
plot(1:N, 10*log10(Jmin*ones(1,N)),'r', ...
    1:N,10*log10(Jms(1:N)),'b', 1:N, 10*log10(Jex(1:N)), ...
    'k', 1:N,10*log10(Jms_inf*ones(1,N)), '--b', ...
    1:N,10*log10(Jex_inf*ones(1,N)),'--k' )
xlabel('\bf iterations','FontSize',18 )
ylabel('\bf MSE [dB]','FontSize',18)
legend(['Minimum MSE'],['MSE'],['Excess ...
    MSE'],['Steady-State of MSE'],['Steady-State of ...
    Excess MSE'])
grid
```