UNIVERSIDADE FEDERAL DE UBERLÂNDIA
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ANÁLISE E PARALELIZAÇÃO DE ALGORITMOS APLICADOS À IDENTIFICAÇÃO DE SISTEMAS DINAMICOS NÃO-LINEARES COM MODELO NCARMA FRACIONÁRIO

GUILHERME RESENDE FERREIRA

OUTUBRO
2015

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ANÁLISE E PARALELIZAÇÃO DE ALGORITMOS APLICADOS À IDENTIFICAÇÃO DE SISTEMAS DINAMICOS NÃO-LINEARES COM MODELO NCARMA FRACIONÁRIO

Dissertação apresentada por *Guilherme Resende Ferreira* à Universidade Federal de Uberlândia para obtenção do título de Mestre em Engenharia Elétrica, aprovada em 19/10/2015, pela seguinte banca examinadora:

Prof. Fabio Vicenzi Romualdo da Silva, Dr. (UFU-FEELT)
Orientador
Prof. Josué Silva De Morais, Dr. (UFU-FEELT)
Co-Orientador
Prof. Marcio Jose da Cunha, Dr. (UFU-FEELT)
Prof. Henrique José Avelar, Dr. (CEFET-MG)

Uberlândia, 19 de Outubro de 2015.

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

F383a 2015

Ferreira, Guilherme Resende, 1991-
Análise e paralelização de algoritmos aplicados à identificação de sistemas dinâmicos não-lineares com modelo NCARMA Fracionário / Guilherme Resende Ferreira. - 2015.
152 f. : il.

Orientador: Fábio Vincenzi Romualdo da Silva.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica.
Inclui bibliografia.

1. Engenharia elétrica - Teses. 2. Identificação de sistemas - Teses. 3. Algoritmos de computador - Teses. 4. Sistemas dinamicos - Teses. I. Silva, Fábio Vincenzi Romualdo da, 1974-. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU: 621.3

ANÁLISE E PARALELIZAÇÃO DE ALGORITMOS APLICADOS À IDENTIFICAÇÃO DE SISTEMAS DINAMICOS NÃO-LINEARES COM MODELO NCARMA FRACIOÁRIO

GUILHERME RESENDE FERREIRA

Dissertação apresentada por Guilherme Resende Ferreira à Universidade Federal de Uberlândia como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica.

Fabio Vicenzi Romualdo da Silva**, Dr.**	**Prof. Darizon Alves de Andrade, Dr.**
(Orientador)	Coordenador do Curso de Pós-Graduação

O aspecto mais triste da vida de hoje é que a ciência
ganha em conhecimento mais rapidamente
que a sociedade em sabedoria

Isaac Asimov

Dedicatória:

Dedico este trabalho aos meus pais, Vera Lúcia e José Faber, a meu irmão, André Resende, e a minha namorada, Jéssica Pereira, pelo estímulo, carinho, paciência e compreensão.

AGRADECIMENTOS

Aos meus queridos pais, Vera Lúcia Resende e José Faber Rodrigues Ferreira, ao meu irmão e companheiro, André Resende Ferreira, pela dedicação, pelo amor incondicional, por incentivarem meus estudos e por me darem uma base familiar sólida que me permitiu ser tudo o que me tornei e que me permitirá continuar a crescer e forma moral, intelectual, sentimental e doutrinária.

A minha querida namorada Jéssica pelo amor verdadeiro, pelo carinho, pela compreensão, pela paciência, por ter estado ao meu lado durante todo o processo, especialmente nas horas mais difíceis, por me incentivar sempre a continuar indo em frente. Agradeço por ter você em minha vida tornando-a mais alegre, e por ter me dado todo o apoio pessoal que possibilitou passar por mestrado, agradeço principalmente por ter me ensinado sobre a essência do mais puro amor e a importância dele.

Aos meus familiares que torceram por pela minha felicidade e sucesso, que assistiram ao processo sempre me incentivando. Em especial ao meu falecido avô que me ensinou desde criança todos os ensinamentos básicos que me permitiram tornar-me um grande engenheiro.

Aos meus amigos que me acompanharam durante todo o processo, em especial aqueles que me ajudaram de forma direta ou indireta a tornar este trabalho possível.

Ao Prof. Josué Silva De Morais pela co-orientação, incentivo e motivação deste trabalho, e por todo o apoio, atenção, ensinamentos, amizade e confiança que ele me deu não só como orientador, mas também como professor e amigo.

Aos funcionários do departamento de Engenharia Elétrica da Universidade Federal de Uberlândia, e a todos os técnicos que sempre me receberam com carinho e disposição.

E por fim, à UFU, pela oportunidade de conclusão do mestrado, pelo apoio e infraestrutura necessários para a conclusão deste trabalho.

RESUMO

O objetivo principal deste trabalho é desenvolver um software com uso de um algoritmo Diferencial Evolutivo (DE) utilizado na identificação de sistemas dinâmicos não lineares com o uso do modelo NCARMA Fracionário, visando melhorar seu desempenho através da análise de algoritmos, paralelos ou sequenciais, utilizados pelo mesmo.

A avaliação é efetuada através de um aplicativo desenvolvido especificamente para este trabalho com o ambiente Qt utilizando a linguagem C++ 14 e técnicas de programação paralela.

Para atingir esse objetivo selecionou-se os principais algoritmos que são utilizados pelo software desenvolvido a fim de comparar metodologias e técnicas diferentes empregadas na solução dos mesmos.

Diferentes tipos de algoritmos foram testados a fim de se definir qual metodologia apresenta melhores resultados quando aplicada ao software desenvolvido, além de testes de métodos diferentes também se testou bibliotecas diferentes e testes dos algoritmos em suas formas paralelas e sequenciais.

Por fim, o aplicativo final é validado e testado quando à sua capacidade de modelar sistemas pré-definidos e quanto ao tempo total dispendido.

Palavras chaves: Diferencial Evolutivo, Identificação de Sistemas, NCARMA Fracionário, Tempo dispendido.

ABSTRACT:

The main goal of this paper is to contribute to the performance of a Differential evolutionary algorithm (DE) used in the identification of nonlinear dynamical systems using the NCARMA Fractional model, by comparing algorithms, parallel or sequential, used by the same.

The evaluation is made through an application developed specifically for this work with Qt environment using C ++ 14 language and parallel programming techniques.

To achieve this goal we selected the main algorithms that are used by software developed in order to compare different methodologies and techniques used to solve them.

Different types of algorithms have been tested in order to determine which method does best in each situation, as well as different testing methods, different libraries and sequential versus parallel computation.

Finally, the final application has been validated and tested as to their ability to model pre-defined systems considering the total time spent.

Key Words: Differential Evolutionary, System Identification, NCARMA fractional, Time Spent.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1:
Introdução

Com o passar dos anos o computador acabou por se tornar uma das ferramentas de maior importância já desenvolvidas pela humanidade. Atualmente ele está inserido na vida de muitas pessoas, acelerando e automatizando tarefas e promovendo inúmeras facilidades, possibilitando que usuários consigam realizar tarefas mais produtivas e interessantes (EVANS, 2011).

A utilização dos computadores, de uma forma generalizada, alcançou lugares antes inimagináveis, principalmente após o surgimento dos *smartphones,* celulares providos de alto poder de processamento,e da expansão do acesso à internet, considerando o acesso às redes de internet móvel como o 3G e o 4G e a expansão da internet a cabo que, dependendo do local, já atinge uma velocidade de centenas de megabits por segundo para usuários domésticos (SAUTER, 2009).

Consequentemente surge uma demanda por um maior poder computacional, que estimula o desenvolvimento de computadores com maior capacidade de processamento e memória,mas com dimensões reduzidas (HESHAM EL-REWINI, 2005).

Tendo em vista o suprimento desta demanda, existem algumas formas de se obter melhor desempenho computacional, dentre as quais destacam-se os modelos de computadores paralelos, que visam melhorar o desempenho através do uso de paralelismo computacional e algoritmos paralelos (BELL, 1994). O que vem se tornando uma tendência desde o ano de 2006, em que a *Intel* lançou o primeiro processador com múltiplos núcleos para computadores pessoais, desde então o processador passou a evoluir de forma a melhorar sua performance paralela, através da

inserção de mais núcleos de processamento no processador, e não mais apenas aumentando o *clock*, como vinha sendo feito até então (LEI CHAI, 2007).

Além dessas tendências atuais e futuras de computação para usuários domésticos, a computação também é aplicada em diversas áreas científicas que demandam por aplicações computacionais a fim de solucionar diversos problemas. Dentre elas destacam-se as que demandam por computação de alto desempenho, como a farmacologia, otimização aerodinâmica, área financeira, climatologia, mineração de dados, biologia, geologia, astronomia, mecânica de fluidos, Inteligência artificial (IA) e manipulação de grandes bancos de dados (SILVA, 2008)(PEVIANI, 2009)(HWANG. K., 1998).

Dentre as áreas mais tradicionais da ciência e da engenharia que demandam de uma computação cada vez mais robusta e otimizada, encontra-se a identificação de sistemas dinâmicos não lineares.Ela compõe um campo científico e metodológico ao lado de categorias teóricas e experimentais, sendo através desses domínios científicos que a computação demonstra ser uma ferramenta de grande valor na solução de problemas complexos(KE WANG, 2008)(BIDYADHAR SUBUDHI, 2008)(XIAOCEN XUE, 2012).

Dentro do contexto de identificação de sistemas e analisando o meio industrial e o meio acadêmico nota-se a necessidade de modelos matemáticos capazes de reproduzir o comportamento dinâmico de sistemas reais (LI FU, 2013) e (PAIVA, 1999), alguns dos quais apresentam grande complexidade matemática para serem identificados (LI FU, 2013), (CHIHA IBTISSEM, 2013) e (XIAOCEN XUE, 2012) demandando técnicas cada vez mais dispendiosas, do ponto de vista da capacidade de processamento computacional, para sua aquisição (CHIHA IBTISSEM, 2013). E neste aspecto o

paralelismo computacional vem se mostrando uma ferramenta promissora (HWANG. K., 1998).

Para obter maior eficiência e produtividade em processos que podem ser representados por modelos matemáticos, as indústrias recorrem a modelos matemáticos mais representativos, que podem ser utilizados em técnicas de controle mais modernas, em processos de simulação para treinamento de pessoal, diagnostico de falhas, avaliação de produção (MORAIS, 2013).

Existem diferentes tipos de representação de um modelo matemático de um sistema real e deve-se levar em conta as características distintas de cada um deles a fim de selecionar o que melhor representa as características da planta a ser modelada, além de se considerar se um modelo linear é capaz de representar o sistema ou se será necessário um modelo não linear. Quanto mais representativo é o modelo selecionado tende a ser mais dispendioso computacionalmente o algoritmo para identificá-lo (MORAIS, 2013).

Alguns autores apresentam técnicas com uma representatividade muito promissora, sem super parametrização, mas que exigem de um esforço computacional elevado para a sua obtenção (CHIHA IBTISSEM, 2013), (XIAOCEN XUE, 2012), (BIDYADHAR SUBUDHI, 2008).

Dentre os vários métodos utilizados em técnicas de identificação de sistemas apresentadas na literatura, como o uso de redes neurais e os métodos matemáticos (MORAIS, 2013), o uso de algoritmos genéticos (AG) e de algoritmos DE com objetivo de se auxiliar na identificação de sistemas, tem ganho grande destaque em pesquisas (CHIHA IBTISSEM, 2013), (BIDYADHAR SUBUDHI, 2008), (WEN-HSIEN HO, 2009), (SWATI SWAYAMSIDDHA, 2015), (HELON VICENTE HULTMANN AYALA, 2014), (JOEL H. VAN SICKEL, 2007), (SWATI SWAYAMSIDDHA, 2013),

(XIAOCEN XUE, 2012) e (KE WANG, 2008), devido a sua relativa simplicidade e eficiência para otimização de problemas com espaço de solução contínuo (RAINER STORN, 1995), além de se caracterizar como um método paralelizável.

O algoritmo DE foi apresentado inicialmente por Kenneth Price e Rainer Storn em 1996, na competição internacional de computação evolutiva de Nagoya (RAINER STORN, 1996). Esta técnica consiste basicamente de três operadores: mutação, cruzamento e seleção; e de três parâmetros: tamanho da população, fator de escala F e probabilidade de cruzamento (RAINER STORN, 1996) e (YOUYUN AO, 2009).

Embora seja considerado um algoritmo evolutivo, o DE não tem inspiração ou origem em nenhum processo natural, sendo que a forma como são geradas mutações na população, baseia-se em argumentos matemáticos e heurísticos e não em metáforas da natureza (GUIMARÃES, 2009).

Embora o algoritmo DE seja relativamente simples ele exige um alto desempenho computacional quando se trata de um problema complexo, como a identificação de sistemas dinâmicos não lineares(MORAIS, 2013). Para melhorar o tempo de execução dos algoritmos alguns autores propõem o uso de paralelismo computacional na solução de AG e DE (D.K. TASOULIS, 2004), (PAVEL KROMER, 2011), (CHENG XIAO, 2011) e (PAVEL KROMER, 2013).

1.1) Levantamento Bibliográfico

Durante a pesquisa de revisão bibliográfica, realizada em junho de 2015, utilizando-se do mecanismo de busca do IEEE Xplore e selecionando-se todas suas bases de dados disponíveis, encontrou-se um total de 9613 documentos, quando se procurando pela palavra-chave *"system identification"*. Os resultados por período de tempo podem ser visualizados na Tabela 1.

Tabela 1: Resultados dos periódicos por período de tempo para a palavra-chave *"system identification"*.

FONTE: Adaptado de (IEEE Xplore Digital Library, 2015).

ANO	RESULTADOS	%
ATÉ 1985	574	5,97
1986-1995	1517	15,78
1996-2005	2950	30,68
2006-2015	4572	47,56

Ao alterar a palavra chave para *"nonlinear system identification"*, encontrou-se um total de 741 documentos. Os resultados por período de tempo podem ser visualizados na Tabela 2.

Tabela 2: Resultados dos periódicos por período de tempo para a palavra-chave *"nonlinear system identification"*.

FONTE: Adaptado de (IEEE Xplore Digital Library, 2015).

ANO	RESULTADOS	%
ATÉ 1985	12	1,61
1986-1995	86	11,60
1996-2005	243	32,79
2006-2015	400	53,98

Alterando-se a palavra chave para "*nonlinear system identification*" e "*Neural Networks"* e "*genetic algorithm*", encontrou-se um total de 368 documentos. Os resultados por período de tempo podem ser visualizados na Tabela 3.

Tabela 3: Resultados dos periódicos por período de tempo para a palavra-chave "*nonlinear system identification*" e "*Neural Networks*" e "*genetic algorithm*".

FONTE: Adaptado de (IEEE Xplore Digital Library, 2015).

ANO	RESULTADOS	%
ATÉ 1985	0	0
1986-1995	8	2,17
1996-2005	117	31,79
2006-2015	243	66,03

Adicionalmente, quando a busca foi realizada com as palavras chave para "*nonlinear system identification*" e "*differential evolution*", encontrou-se um total de 794 documentos. Os resultados por período de tempo podem ser visualizados na Tabela 4.

Tabela 4: Resultados dos periódicos por período de tempo para a palavra-chave "*nonlinear system identification*" "*differential evolution*".

FONTE: Adaptado de (IEEE Xplore Digital Library, 2015).

ANO	RESULTADOS	%
ATÉ 1985	0	0
1986-1995	0	0
1996-2005	97	12,21
2006-2015	697	87,78

Ao alterar a palavra chave para "*nonlinear system identification"* e *"performance"*, encontrou-se um total de 2214 documentos. Os resultados por período de tempo podem ser visualizados na Tabela 5.

Tabela 5: Resultados dos periódicos por período de tempo para a palavra-chave "*nonlinear system identification*" e "*performance*".

FONTE: Adaptado de (IEEE Xplore Digital Library, 2015).

ANO	RESULTADOS	%
ATÉ 1985	46	2,07
1986-1995	260	11,74
1996-2005	706	31,88
2006-2015	1202	54,29

Ao alterar a palavra chave para "*nonlinear system identification*" e "*differential evolution*" e "*performance*", encontrou-se um total de 14 documentos. Os resultados por período de tempo podem ser visualizados na Tabela 6.

Tabela 6: Resultados dos periódicos por período de tempo para a palavra-chave "*nonlinear system identification*" " e "*differential evolution*" e *"performance*".

FONTE: Adaptado de (IEEE Xplore Digital Library, 2015).

ANO	RESULTADOS	%
ATÉ 1985	0	0
1986-1995	0	0
1996-2005	0	0
2006-2015	14	100

Um detalhe importante desses dados é que, em média, aproximadamente 70% dos trabalhos publicados estão concentrados na última década. O que evidencia o aumento das atividades em pesquisa na área principal do estudo.

Completando a análise, existem muitos trabalhos na área da identificação de sistemas, dos quais muitos visam sistemas não lineares. Na área de computação evolutiva e inteligência artificial destacam-se trabalhos com uso do algoritmo DE para

solução dos mesmos, e embora destacam-se os trabalhos que visam performance com identificação de sistemas não lineares, poucos visam a performance do algoritmo quando usando-se DE. A Tabela 7 sumariza as informações coletadas por meios das buscas no mecanismo IEEE Xplore.

Tabela 7: Resultados dos periódicos Totais. (Sumário).

FONTE: Adaptado de (IEEE Xplore Digital Library, 2015).

Palavra-Chave	**Resultados**
"System identification"	9613
"Nonlinear system identification"	741
"Nonlinear system identification" e "Neural Networks" e "Genetic algorithm".	368
"Nonlinear system identification" "Differential evolution".	794
"Nonlinear system identification" e "Performance"	2214
"Nonlinear system identification" ” e "Differential evolution" e "Performance"	14

Os trabalhospropostos na literutura,que unificam a identificação de sistemas com o uso do algoritmo DE visando melhorar a performance, buscam identificar melhor os modelos, mas não buscam eficiência de algoritmo ou performance do mesmo na forma de redução do esforço computacional.

1.2) Objetivo do trabalho

Considerando a situação em que os processadores atuais possuem múltiplos núcleos e que o software deve aproveitar ao máximo o poder do processamento e que

existe muita pesquisa em relação a identificação mas poucas focadas em performance e desempenho, propõe-se o desenvolvimento, análise, modelagem e modificação da forma de processamento dos algoritmos utilizados na identificação experimental de sistemas com uso do modelo NCARMA Fracionário proposto em (MORAIS, 2013), visando minimizar o esforço computacional e reduzir o tempo total de execução da identificação.

As principais contribuições deste trabalho são:

1. A analise de algoritmos, em sua forma sequencial e paralela, como o algoritmo de selação por roleta, de mutação, de cruzamento e algoritmos paralelos, utilizados na solução de problemas no processo de identificação de sistemas como a seleção de estrutura e o cálculo dos coeficientes da equação, afim de se selecionar os algoritmos mais adequados ao objetivo do trabalho. Além de analisar bibliotecas matemáticas diferentes, com o objetivo de se selecionar a mais adequada ao trabalho.
2. Apresentara proposta de um método para algoritomo paralelovisando obter tempos de execução melhores quando comparados aos tempos de execução do algoritmo paralelo tradicional com objetivo de melhorar o desempenho paralelodo software em geral.

1.3) Apresentação do trabalho

O capítulo 2 apresenta um detalhamento do tema de computação paralela, a fim de se ambientar melhor o trabalho através de um estudo de artigos e trabalhos na área de pesquisa da presente dissertação com foco no tema de computação paralela, porém apresentando também, de uma forma geral, a computação em si e as arquiteturas paralelas.

O capítulo 3 apresenta um detalhamento do tema de identificação de sistemas, visandoambientar o trabalho através de um estudo de artigos e trabalhos na área de pesquisa da presente dissertação com foco no tema de identificação de sistemas, porém apresentando também, de uma forma geral os temas de algoritmos genéticos, diferencial evolutivo e já mesclando a parte de paralelismo em algoritmos genéticos.

O capítulo 4 apresenta a metodologia aplicada ao presente trabalho, onde são detalhadas as técnicas e os modelos utilizados.

O capítulo 5apresenta os resultados e a análise de performance do aplicativo final e seus algoritmos em relação ao tempo gasto para identificar um determinado sistema.

O capítulo 6apresenta a validação do modelo, onde são utilizados modelos artificiais obtidos da literatura, em que estes apresentam características singulares que geram dificuldades aos algoritmos comumente utilizados, a fim de verificar a eficácia do modelo NCARMA Fracionário e do algoritmo DE utilizado em identificar o sistema.

O capítulo 7 apresenta as conclusões do trabalho e possíveis trabalhos futuros.

Capítulo 2:
Computação Paralela

Considerando que o foco deste trabalho é estudar técnicas de paralelismos aplicadas no processo de identificação de sistemas. Apresenta-se neste capitulo uma breve introdução ao universo do paralelismo computacional, uma breve história da computação, as principais arquiteturas e modelos de paralelismo computacional, além de conceitos como o de *Thread* e o paradigma da variável compartilhada.

2.1) História do Computador

A história do computador inicia-se juntamente a história da matemática por volta do ano 4.200 a.C., passando por várias culturas diferentes e pelo ábaco, até chegar à primeira máquina capaz de realizar operações matemáticas simples, somar, subtrair, multiplicar e dividir que data do século XVII construída por Wilhelm Schikard, porém tal máquina nunca foi encontrada, encontrou-se apenas alguma documentação sobre ela. Por isso atribui-se a Blaise Pascal a construção da primeira calculadora, que podia apenas somar e subtrair, posteriormente a mesma foi aprimorada por Gottfried Wihelm Leibniz que adicionou a capacidade de dividir e multiplicar, a partir daí a história passa por grandes mentes como Boole, Hilbert, Turing e von Neumann, entre outros, já no século XX até chegar nos computadores que conhecemos atualmente (HISTORY, 2015).

Já a ideia de programar uma máquina surge no século XVIII com a necessidade de que as máquinas de tecer fossem programáveis, e em 1801 Joseph Marie Jacquard inventa o primeiro tear programável, que era capaz de selecionar determinados padrões de desenho (FILHO, 2007).

Daí em diante evoluíram várias ideias passando por vários intelectuais de países diferentes, começaram a surgir as primeiras máquinas capazes de cálculos mais complexos, até chegarmos a Alan Mathison Turing, que idealizou e construiu a Máquina de Turing, introduzindo e definindo pela primeira vez o que significava o termo computar algo. Turing também trabalhou no projeto do computador ACE, em projetos de inteligência artificial e em projetos dos primeiros computadores digitais, chegou até a antever as linguagens conhecidas atualmente como linguagens de alto nível (FILHO, 2007).

O primeiro computador digital construído foi o COLOSSUS pelo prof. M. H. A. Newman e sua equipe em 1943, trabalho enormemente influenciado pelos resultados sobre computabilidade obtidos por Alan Turing (FILHO, 2007).

Grandes feitos aconteceram desde então, destacam-se os feitos: a IBM monta os primeiros computadores transistorizados em 1959, surge em 1963 o código ASCII a fim de padronizar a troca de informações entre computadores, em 1971 John Blankenbaker lança o primeiro computador pessoal o Kenbak I, daí em diante grandes acontecimentos marcaram a história do computador, temos a chegada da Intel, da Aple e da Microsoft, dentre muitos outros fatos importantes. Uma tabela cronológica até o ano de 2007, encontra-se disponível em (FILHO, 2007).

Com o avanço das pesquisas e o avanço tecnológico, os computadores sofreram grandes modificações ao longo do tempo. Como reflexo desta evolução formaram-se algumas gerações, devido à modificação dos estilos usados na construção e na programação dos mesmos. Tais gerações estão presentes na tabela 8 (HISTORY, 2015)

Tabela 8: Gerações de Computadores

Fonte: (HISTORY, 2015)

Geração/Período	Tecnologia e Arquitetura	Software e Sistema operacional	Sistema Representativo
Primeira (1946 - 1956)	Válvulas e memórias de tubos catódicos	Linguagem de máquina e assembly	COLOSSUS, ENIAC, IBM 701, Princeton IAS
Segunda (1956 – 1967)	Transistores, núcleos de ferrite, discos magnéticos, barramentos de I/O	Algol e Fortran, compiladores, processamento em lotes	IBM 7030, CDC 1604, Univac LARC
Terceira (1967 - 1978)	CI (Circuitos Integrados) (SSI)	Linguagem C, multiprogramação, *time-sharing*	PDT-11, IBM 360/370, CDC 6600
Quarta (1978 - 1989)	Microprocessadores VLSI, multiprocessadores	Multiprocessamento, compiladores paralelos, bibliotecas de troca de mensagens	IBM PC, VAX 9000, Cray X/MP
Quinta (1990 – Atualmente)	Circuitos ULSI, computadores paralelos escaláveis	Java, *microkermels*, *Multithreading*, OS distribuídos, www	IBM SP2, SGI Origin 2000, Digital TruCluster

Como pode ser visto na tabela 8 e evolução em termos de tecnologia de hardware começa com as válvulas na primeira geração, a segunda geração é marcada pelo uso de transistores, a terceira geração já surgem os primeiros circuitos integrados como o SSI (*Small-Scale Integrated*), já na quarta surgem os primeiros microprocessadores de larga escala ou VLSI (*Very Large-Scale Integrated*) e finalmente a quinta geração é caracterizada pelos computadores paralelos e circuitos ULSI (*Ultra Large-Scale Integrated*) (FILHO, 2007).

Considerando as tecnologias de software temos a primeira geração marcada pelo uso de linguagens de máquina e assembly, a segunda por linguagens como Algol e Fortran, na terceira geração predominou o uso a linguagem C, utilizado até os dias atuais, a quarta geração é marcada pelo início da programação paralela com compiladores paralelos e bibliotecas de comunicação e na quinta e atual geração as

tecnologias de software são voltadas a programação orientada a objeto, à programação paralela e distribuída (HISTORY, 2015).

Um dos grandes marcos da quinta geração ocorre em 2006 quando a Intel lança o primeiro processador com múltiplos núcleos para computadores pessoais (PCs), desde então o processador, voltado para PCs, passou a evoluir de forma a melhorar sua performance paralela e sua arquitetura e não mais apenas aumentando o *clock*(LEI CHAI, 2007).

Desde então processadores com múltiplos núcleos tornaram-se uma tendência da indústria e dos consumidores (LEI CHAI, 2007), de forma que atualmente não se encontram novos processadores para desktop a venda que não tenham múltiplos núcleos.

Consequentemente entender o processo do paralelismo e suas arquiteturas deixou de ser uma necessidade apenas de cientistas da área da computação e passou a ser essencial para programadores de áreas da ciência que se utilizam do computador como ferramenta de trabalho ou pesquisa (HESHAM EL-REWINI, 2005).

2.2) Computação Paralela

Problemas complexos que podem ser solucionados por computadores, como os envolvidos nas áreas de climatologia, IA, biologia, engenharias e na área da identificação de sistemas (HWANG. K., 1998), demandam computadores de alto desempenho para serem solucionados. Um dos métodos de se obter maior desempenho computacional dá-se através do uso de computadores paralelos (BELL, 1994).

Um sistema de computação paralela, como os usados por *clusters*, consiste em um conjunto de processadores interligados entre si com o objetivo de tornar a solução

de determinados problemas mais rápida em relação à modelos que utilizem um único processador(JÁJÀ, 1992).

Com o uso de arquiteturas paralelas, um problema pode ser dividido em *n* subproblemas, de forma que cada processador fica responsável por resolver uma parte destes *n* subproblemas. Desta forma o problema original é resolvido paralelamente, podendo obter uma redução em seu tempo total de execução (JÁJÀ, 1992)(HWANG. K., 1998).

2.3) Arquiteturas Paralelas

Diversas arquiteturas computacionais encontram-se disponíveis atualmente, dentre elas uma das mais utilizadas é a arquitetura sequencial tradicional idealizada por *Von Neumann*(HWANG. K., 1998).

Porém desde meados do século XXI o uso de tal arquitetura começou a se tornar um problema, pois já não era mais possível aumentar o desempenho de processadores sequenciais apenas aumentando sua frequência, dentre os maiores desafios para o continuo aumento da frequência destacam-se o problema de superaquecimento e o problema de consumo de energia (LEI CHAI, 2007).

Consequentemente os responsáveis pela arquitetura computacional desenvolveram o processador *multi-core*, que possuía dois ou mais processadores no mesmo *chip*(BURGER, 2005), tais processadores também são referenciados como *Chip Multiprocessor*(CMP) (LEI CHAI, 2007).

Por outro lado, *clusters*, máquinas compostas de agregados de computadores, já são usados a algumas décadas como modelos de computação paralela mais populares. Primeiramente, eles eram compostos por estações de trabalho (*COW* – Cluster of

Worstations), tais como o IBM SP2, então surgiram os *clusters* de PCs, nos quais máquinas são interligadas por redes de alta velocidade(OLIVEIRA, 2004). Com o aparecimento da arquitetura *multi-core*, os *clusters* adentraram uma nova era *multi-core,* de forma que atualmente a maioria dos supercomputadores, geralmente formados por *clusters*, já apresenta processadores *multi-core*(TOP500, 2015).

Porém não existe, dentro da área da computação paralela, um modelo único amplamente aceito, como na área de computação sequencial, embora existam tentativas de se criar tal modelo, de forma que os algoritmos paralelos são analisados pela complexidade de tempo e por recursos utilizados pelo mesmo (PEVIANI, 2009).

Em parte, a falta de padronização do modelo paralelo se devo ao fato de que a performance de algoritmos paralelos depende de diversos fatores, dentre os quais se destacam concorrência computacional, alocação de processadores e escalabilidade, comunicação e sincronização (JÁJÀ, 1992).

Desta forma computadores paralelos podem ser classificados através de suas características de arquitetura e de seus modos de operação (JÁJÀ, 1992).

2.3.1) Classificação de arquiteturas paralelas segundo Flynn

Uma das maneiras de se classificar arquiteturas de computadores é conhecida como taxonomia de Flynn (FLYNN, 1972), onde destacam-se os sistemas SIMD (*Single Instruction Multiple Data*) e o MIMD (*Multiple Instruction Multiple Data*).

O modelo SIMD baseia-se em todos os processadores executando sincronamente a mesma instrução, porém usando diferentes informações nesta instrução (FLYNN, 1972). Já no modelo MIMD os processadores podem executar diferentes instruções simultaneamente, de forma que cada processador armazena em sua própria memória

local o programa que está executando (FLYNN, 1972). As 4 categorias de classificação de Flnn estão na tabela 9.

Tabela 9: Categorias de classificação de Flnn

Fonte: (FLYNN, 1972)

Instrução/Dados	**Fluxo único de dados (SD)**	**Fluxo Múltiplo de dados (MD)**
Fluxo único de instrução (SI)	SISD	SIMD
Fluxo múltiplo de instrução (MI)	MISD	MIMD

A classe *SISD* (fluxo único de instrução e fluxo único de dados) refere-se aos computadores sequências, os quais são baseados na arquitetura de *Von Neumann*. Conforme a Figura 1, o fluxo de instruções (linha contínua) alimenta uma unidade de controle (C) que ativa a unidade central de processamento (P). A unidade P, por sua vez, atua sobre um único fluxo de dados (linha tracejada), que é lido, processado, e reescrito na memória (M).

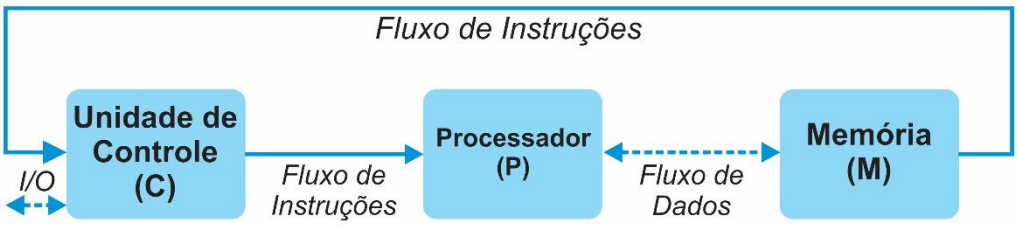

Figura 1: Ilustração do modelo SISD.

A classe *MISD* (fluxo múltiplo de instrução e fluxo único de dados) é uma classe que nunca foi implementada (HESHAM EL-REWINI, 2005). Nessa classe, múltiplos fluxos de instruções atuariam sobre um único fluxo de dados. Conforme a Figura 2, múltiplas unidades de processamento (P), cada uma com sua unidade de controle própria (C), recebem um fluxo diferente de instruções. Essas unidades de

processamento executam suas diferentes instruções sobre o mesmo fluxo de dados. Na prática, diferentes instruções operariam a mesma posição de memória simultaneamente, executando instruções diferentes (HESHAM EL-REWINI, 2005), o que é tecnicamente impossível ainda nos dias de hoje.

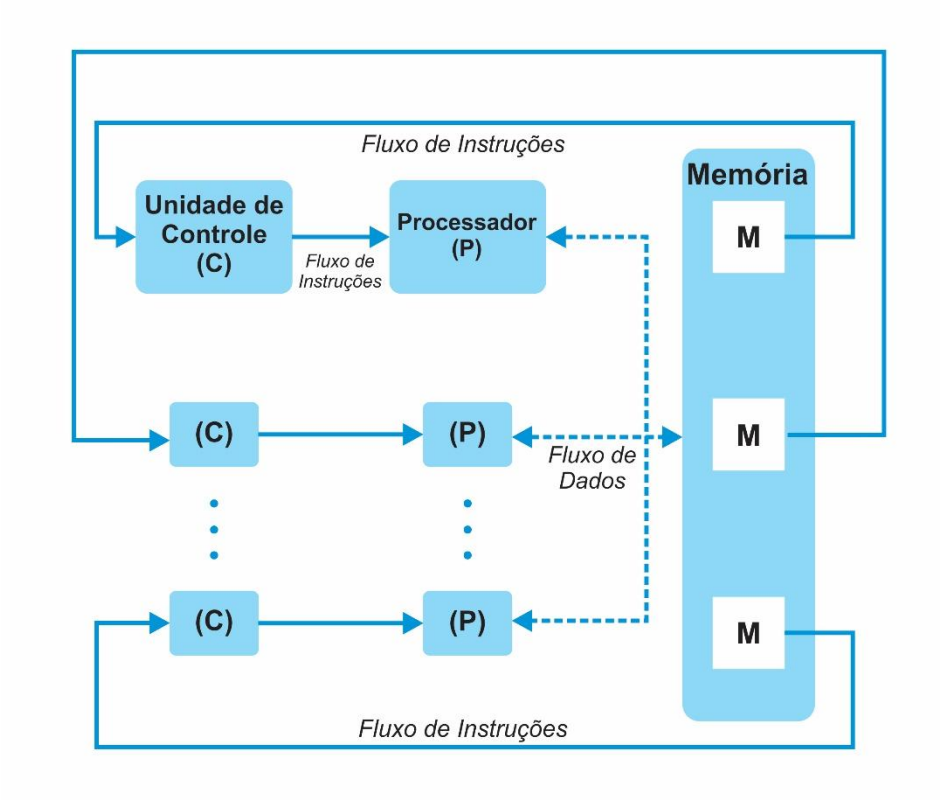

Figura 2:Ilustração do modelo MISD.

As duas classes restantes: *SIMD* e *MIMD*, são as que tratam de computadores paralelos, sendo que as representações mais importantes da classe *SIMD* (fluxo único de instrução e fluxo múltiplo de dados) são computadores e processadores vetoriais. Essas máquinas têm apenas uma unidade de controle e instrumentação de memória, que controla múltiplos processadores. Tais processadores não foram projetados para uso geral, e sim para aplicações específicas como processamento de imagens (HESHAM EL-REWINI, 2005).

Conforme a Figura 3, uma única instrução é executada ao mesmo tempo sobre múltiplos dados. O processamento é controlado por uma única unidade de controle (C), alimentado por um único fluxo de instruções. A mesma instrução é enviada para os diversos processadores (P) envolvidos na execução, e todos os processadores executam suas instruções em paralelo de forma síncrona sobre diferentes dados.

Existe a interpretação de que mesmo programa está sendo executado sobre diferentes dados, o que faz com que o princípio de execução *SIMD*, assemelhe-se ao paradigma de execução seqüencial (HESHAM EL-REWINI, 2005).

É importante ressaltar que, para que o processamento das diferentes posições de memória possa ocorrer em paralelo, a unidade de memória (M) não pode ser implementada como um único módulo de memória.

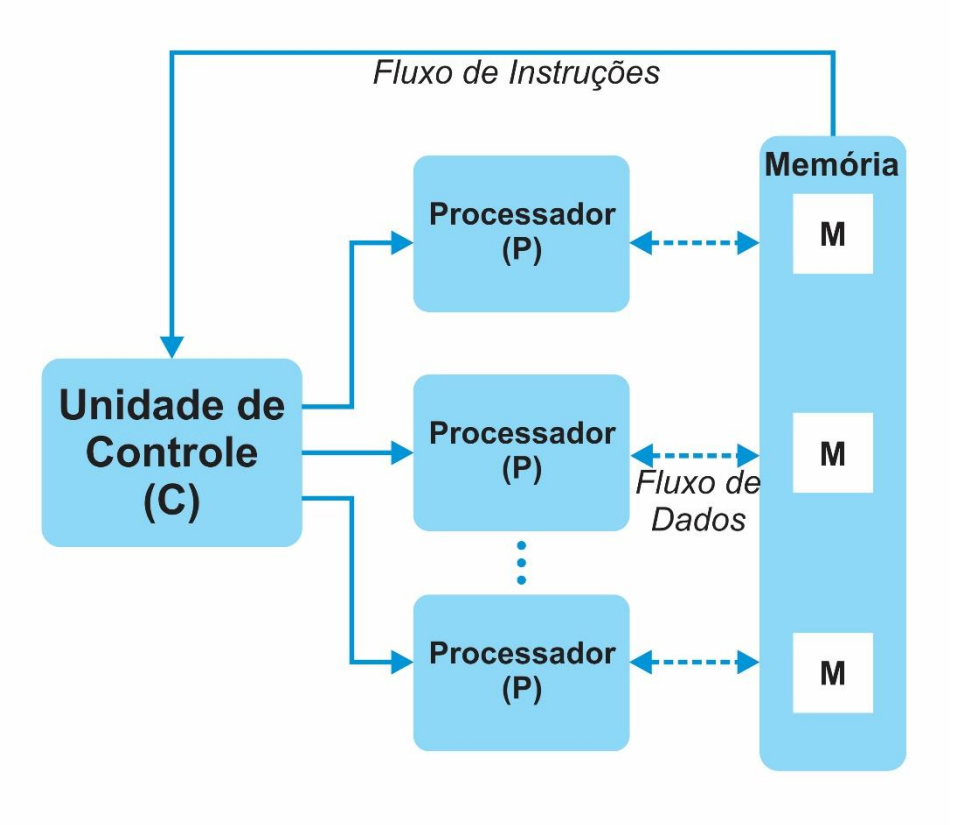

Figura 3: Ilustração do modelo SIMD.

Por fim a classe *MIMD* (fluxo múltiplo de instrução e fluxo múltiplo de dados) é a classe à qual pertence a maioria dos computadores paralelos atuais. Enquanto que em uma classe *SIMD*, apenas um fluxo de instruções, ou seja, um único programa pode ser executado, em uma máquina *MIMD*, cada unidade de controle (C) recebe um fluxo de instruções próprio, conforme pode ser visto na Figura 4. Dessa forma, cada processador executa suas próprias instruções sobre seus próprios dados de forma assíncrona. Assim como na classe *SIMD*, a unidade de memória (M) não pode ser implementada como um único módulo de memória, o que permitiria apenas uma operação por vez (HESHAM EL-REWINI, 2005).

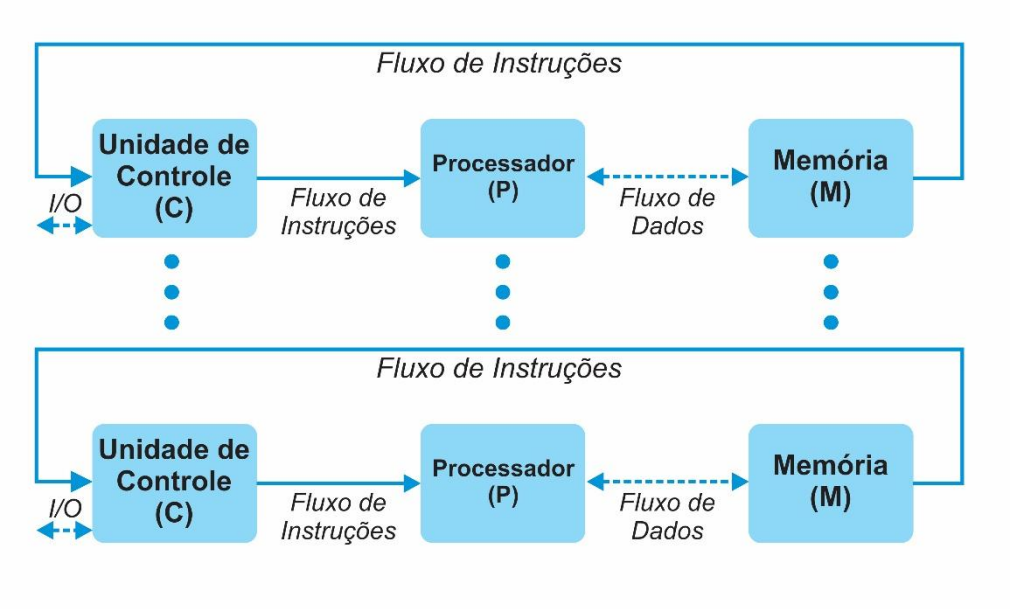

Figura 4: Ilustração do modelo MIMD.

Considerando ainda o sincronismo das operações os computadores paralelos classificam-se em computador paralelo síncrono, onde todos os processadores executam suas instruções (mesma instrução ou instruções diferentes) sob o controle de um único relógio comum (mesmo *clock*), ou computador paralelo assíncrono, onde não há um

relógio comum para todos os processadores, neste caso é necessário incluir pontos de sincronização entres os processadores (PEVIANI, 2009).

2.3.2) Modelo de arquitetura de comunicação entre Processador e Memória

Embora não exista um modelo de computação paralela unicamente utilizado, apresentam-se três modelos que são bastante utilizados no desenvolvimento e análise de algoritmos paralelos, o Modelo de Memória Compartilhada, o Modelo de Memória Distribuída e o modelo de memória compartilhada e distribuída (HESHAM EL-REWINI, 2005).

2.3.2.1) Modelo de Memória Compartilhada

O modelo de memória compartilhada consiste de um número de processadores, que executa seu próprio conjunto de instruções local e compartilham a mesma memória. A comunicação é feita através da troca de informações pela memória global, este modelo contém classificações quanto a forma que efetua leitura e escrita na memória compartilhada (PEVIANI, 2009). Tal modelo pode ser visualizado na figura 5.

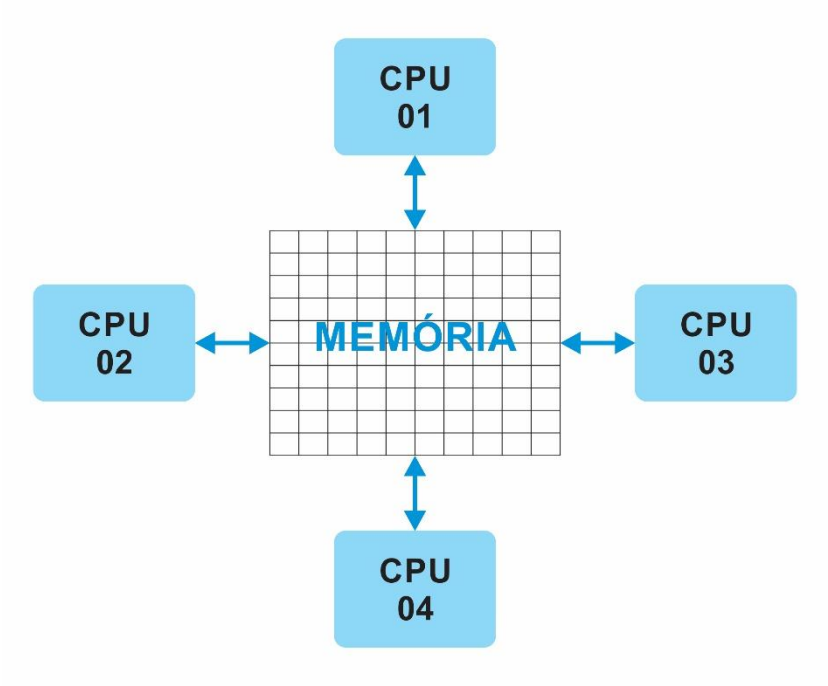

Figura 5:Modelo de memória compartilhada.

Este tipo de computador também é conhecido como UMA (*Uniform memory access*) ou multiprocessado. Ele permite acesso uniforme à memória uma vez que todos os processadores a acessam com a mesma latência e largura de banda(HESHAM EL-REWINI, 2005).

Uma das principais vantagens do uso desta arquitetura é a facilidade de programação oriunda do compartilhamento da memória, desta forma todos os dados estão disponíveis a todos os processadores, porém este fato também apresenta alguns problemas, como o problema de *data-race*, que ocorre quando duas ou mais *threads* tentam acessar a mesma posição de memória e pelo menos uma delas está tentando escrever nesta posição e *deadlock*, que ocorre quando duas ou mais *threads* ficam bloqueadas em estado de aguardo, uma bloqueando a outra e vice versa. Este modelo dispensa a preocupação com sincronização por parte do programador, porém a custo de um hardware mais complexo e com custo elevado. Tais cuidados, dentre outros fatores,

influenciam no aumento da complexidade do código e na dificuldade de compreensão do mesmo (DIG, 2010).

2.**3.2.2) Modelo de Memória Distribuída**

O modelo de memória distribuída, também conhecido como modelo de redes, consiste em distribuir a memória entre os vários processadores e não há memória global (PEVIANI, 2009), desta forma cada processador possui sua própria memória privada. Nesta classificação existem algumas topologias representativas, como a topologia Linear ou Anel, que consiste de N processadores ligados entre si de forma linear, onde o Anel acontece quando o primeiro processador se liga ao último. A malha, que é uma versão bidimensional da Linear. A topologia de árvore binária e por fim a topologia de Hipercubo (PEVIANI, 2009).

A figura 6 apresenta a organização entre os processadores (P) e os módulos de memória privada (M). A comunicação entre eles se dá através de um barramento de conexão, geralmente de alta velocidade. Esta arquitetura não apresenta conflitos no acesso à memória, desta forma atinge-se maiores níveis de escalabilidade do que na arquitetura de memória compartilhada (HESHAM EL-REWINI, 2005).

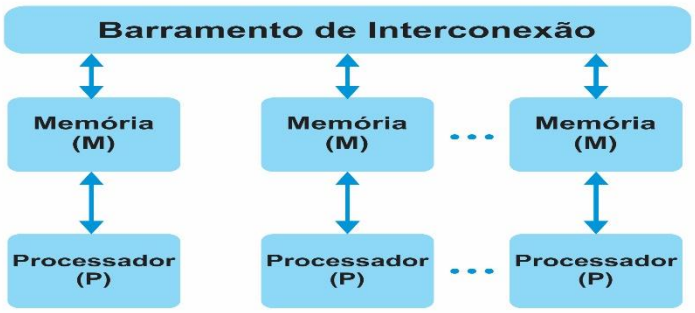

Figura 6: Modelo de memória distribuída.

Uma representação típica deste modelo são os *clusters* computacionais e os computadores paralelos maciços (MPPs). A vantagem do uso de *clusters* é que os mesmos se utilizam de redes de baixo custo, como Ethernet e Myrinet e possuem um custo benefício melhor. Porém não conseguem atingir os desempenhos obtidos pelas MPPs (OLIVEIRA, 2004).

2.3.2.3) Modelo de Memória Compartilhada e Distribuída (Mista)

A fim de se combinar as vantagens das duas arquiteturas (fácil programação e alto nível de escalabilidade), foi estabelecida um terceiro modelo de memória, o modelo de memória compartilhada e distribuída (*Distributed Shared Memory - DSM*) (HESHAM EL-REWINI, 2005).

Neste modelo cada processador (P) tem uma memória privada, e acesso à memória global (M), conforme figura 7. Desta forma a memória global fornece um meio simplificado de trocar dados entre os processadores e a memória privada fornece um meio de se obter maior nível de escalabilidade. Porém o uso da memória privada introduz alguns problemas, como manter os dados atualizados entre a memória privada a memória global. Para solucionar este problema são utilizadas técnicas de consistência e coerência de *cache* (HESHAM EL-REWINI, 2005).

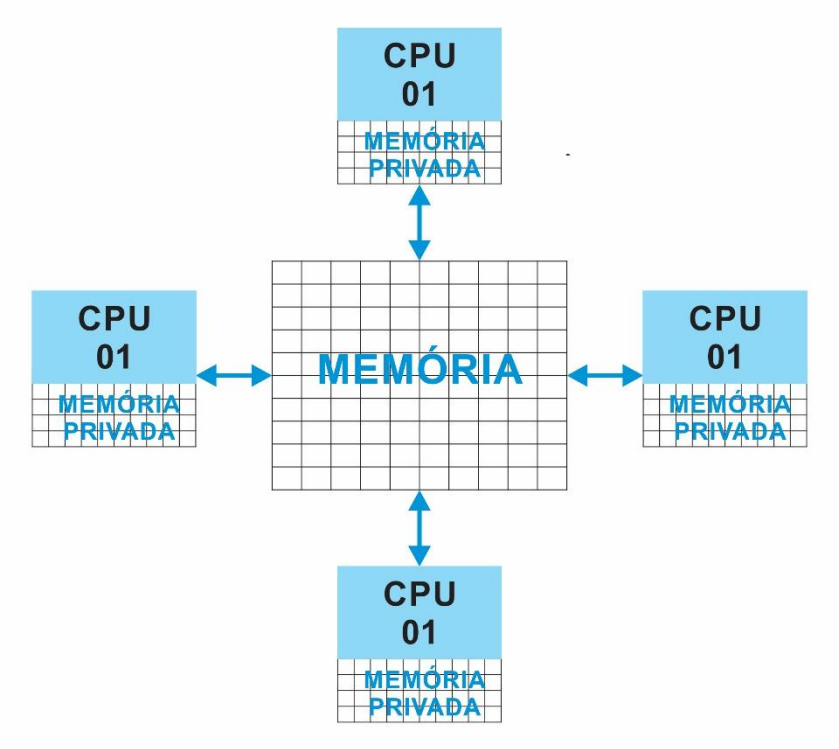

Figura 7: Modelo de memória Compartilhada e Distribuída.

Uma desses métodos que busca coerência de *cache*, é denominada Memória Virtual Compartilhada (*Shared Virtual Memory - SVM*). Nele o mecanismo de gerenciamento de memória de um sistema operacional tradicional é modificado para suportar esses serviços em nível de páginas ou de segmentos. A vantagem dessa abordagem é que não são necessárias alterações nas aplicações (BELL, 1994).

2.4) Threads

Um aspecto importante dos sistemas operacionais atuais é o suporte à *threads* em um mesmo processo. O *thread*, também denominado *light weight process*, é uma unidade básica da utilização do processo, e executa de forma sequencial no processador, sendo interrompida apenas para que o processador possa alternar entre diferentes *threads* (HESHAM EL-REWINI, 2005).

Um mesmo processo pode conter vários *threads*, porém um *thread* pertence a apenas um processo, não há compartilhamento de *threads* entre diferentes processos. Alternar entre diferentes *threads* é menos dispendioso quando comparado ao contexto de alternar entre diferentes processos (BELL, 1994).

O uso de processos concorrentes é explorado pois diferentes *threads* pertencentes à diferentes processos podem rodar em paralelo concorrentemente, embora *threads* pertencentes a um mesmo processo possam ser executadas em diferentes processadores também, ou diferentes núcleos de um mesmo processador (HESHAM EL-REWINI, 2005).

Os sistemas operacionais atuais, distribuições Linux, Windows, Android e IOS, já suportam o uso de *threads*, as linguagens de programação mais utilizadas, C++, JAVA, Delphi e Ruby, por exemplo, também já contemplam o suporte a *threads.*

2.5) Paradigma da Variável Compartilhada

O foco do trabalho se dá no modelo mais utilizado de computação paralela, o MIMD, de forma que os múltiplos processadores trabalham com memória compartilhada, e a troca de informação se dá através desta memória. O uso desses múltiplos processadores se dá através do uso de *threads* em um sistema *multithread*.

Consequentemente, qualquer local da memória pode ser acessado por qualquer *thread*, o que requer mecanismos de sincronização, como *mutex*, semáforos ou monitor, a fim de se evitar que uma mesma variável seja alterada ao mesmo tempo por mais de um thread, causando problemas como o de *data-race* (HENRI E. BAL, 1991),(BELL, 1994) *e*(HESHAM EL-REWINI, 2005).

As vantagens de se utilizar o modelo de memória compartilhada e *threads* incluem a facilidade de acesso a estruturas de dados complexas, a possibilidade da

passagem de dados por referência e a possibilidade de troca de dados sem o uso de estruturas e mecanismos de comunicação entre processos. A principal desvantagem é o não isolamento dos dados, o que gera a necessidade de estruturas de controle e sincronismo entre os *threads* (HENRI E. BAL, 1991).

2.6) Paradigma da Troca de Mensagens

Uma maneira natural de se programar em máquinas de memória distribuída é através da troca de mensagens, assim como, nas máquinas de memória compartilhada, é natural o uso de variáveis compartilhadas (BELL, 1994).

A principal vantagem do paradigma da troca de mensagens é que os processadores se comunicam através do envio e recebimento de mensagens. Assim, nesse modelo não há o conceito de memória compartilhada ou de processadores que possam acessar a memória de outro processador diretamente, o que evita a problemática do controle de acesso. Porém deve-se utilizar um protocolo de controle para a troca de mensagens.Compartilhar uma estrutura de dados torna-se uma tarefa mais complexa e os processos que estão trocando mensagens devem executar simultaneamente (HENRI E. BAL, 1991).

Em geral, ressalvando algumas aplicações específicas, o modelo de programação com comunicação por memória compartilhada é o mais utilizado, devido a suas potencialidades e simplicidade de programação (HESHAM EL-REWINI, 2005).

2.7) Programação Paralela Modelo MIMD

A fim de obter todo o poder de processamento do hardware paralelo, indiferente à arquitetura utilizada, um bom código deve ser utilizado. Desta forma destaca-se a

importância de um software dedicado com um elevado nível de abstração do sistema, com o objetivo de se programar o hardware com eficiência.

Neste sentido o uso de programação paralela mostra-se como uma necessidade quando o objetivo é extrair o máximo de desempenho dos processadores atuais. Porém, tornar o código paralelo, implica em alguns desafios (ZHEN LI, 2013) e (DIG, 2010), entre eles o cuidado com situações de *data-race,* e *deadlock*. Tais cuidados, dentre outros fatores, influenciam no aumento da complexidade do código e na dificuldade de compreensão do mesmo (DIG, 2010).

Porém, nem todo algoritmo melhora ao ser paralelizado, como por exemplo, quando o mesmo contém trechos de código que não podem ser executadas de forma assíncrona (ZHEN LI, 2013) ou grandes trechos de código apresentando dados compartilhados entre mais de uma thread, dados estes alocados na memória global compartilhada, que forçam o uso de semáforos e estruturas de controle de acesso ao dado, podendo tornar o paralelismo lento e ineficiente, além de que, alguns algoritmos são mais simples e rápidos em sua forma sequencial (DIG, 2010).

O desenvolvimento de software deve focar na resolução de tarefas cada vez mais complexas utilizando-se ao máximo da capacidade computacional disponível, sem desperdício de recursos computacionais. Para isso cada algoritmo utilizado pelo software deve ser analisado a fim de melhorar seu desempenho sequencial e paralelo, e reduzir o desperdício de recursos (ZHEN LI, 2013), (RANCE RODRIGUES, 2012) e (DIG, 2010).

2.8) Conclusão

Este capítulo apresenta uma breve introdução ao universo do paralelismo computacional, é apresentada uma breve história da computação, além das principais arquiteturas e modelos de paralelismo computacional. No presente trabalho optou –se por dar foco ao modelo de memória compartilhada e distribuída no modelo MIMD, pois é o modelo mais próximo aos encontrados nos processadores Intel e AMD presentes na maioria dos PC's atuais, tal modelo baseia-se em uma estrutura de memória compartilhada (Memória RAM) e em uma memória privativa (memória *cache*), mesmo que a arquitetura dos processadores varie de modelo para modelo ou se série para série (INTEL, 2015)(AMD, 2011).

O trabalho optou por dar foco no uso de *threads*, para as partes que se mostrarem vantajosas de serem paralelizadas, e no uso de variáveis compartilhadas, devido às vantagens de menor custo computacional e maior facilidade de programação, desta forma a trabalho atual propõe uma nova abordagem no uso de *threads* que mostrou ser mais eficiente em relação à abordagem tradicional e será apresentada nos capítulos posteriores.

.

Capítulo 3:
Identificação de sistemas

Neste capitulo é apresentado as técnicas de identificação que se propõe paralelizar, tais como o NARMAX Fracionário e seus predecessores, os algoritmos evolutivos e diferencias evolutivos e os indicadores matemáticos de validação usados nestas técnicas supra citadas.

3.1) Identificação de sistemas

O uso da identificação de sistemas tem se mostrado cada vez mais presente nas mais diversas áreas do conhecimento, ao mesmo tempo em que os sistemas atuais têm se tornado cada vez mais complexos (LI FU, 2013).

A técnica identificação experimental de sistemas consiste em determinar um modelo, matemático ou não, que melhor represente o conjunto de dados medidos, com uso de métodos otimizados e partindo de dados de entrada e saída do sistema a ser modelado, da seleção da classe do modelo e do uso de uma função de critério de erro. Este modelo pode ser usado para, por exemplo, obter uma noção do comportamento do sistema, predição, controle, estimação do estado e simulação(DING, 2011).

De uma forma mais simples pode-se definir identificação de sistemas como a construção de um modelo, matemático ou não, através dos dados experimentais de entrada e saída do sistema a ser modelado (LI FU, 2013).

Sistemas são identificados através de modelos matemáticos, cuja forma de se obter consiste de dois meios, o analítico/modelagem teórica que consiste em um modelo matemático formado por uma ou mais equações diferenciais e/ou equações algébricas, obtidas através de análises e preposições e o modelo experimental que consiste em

equações diferenciais obtidas através de entradas e saídas de um determinado processo dinâmico real (SANTOS, 2000).

Além da forma de obtenção, existem três principais tipos de modelagens, caixa branca, onde todos os parâmetros são conhecidos ou previamente estimados. Caixa cinza, onde parte dos parâmetros são conhecidos ou previamente estimados. E caixa preta, onde nenhuma informação do sistema está disponível além dos dados experimentais de entrada e saída (GARCIA, 1997). Há também dois tipos de modelos, o linear e o não linear. No caso dos lineares a obtenção do modelo é simples, porém restrita, já no caso dos não lineares a obtenção é complexa e o modelo é mais preciso e completo (BILLINGS, 1980).

A fim de se obter os modelos matemáticos que representam processos industriais dinâmicos e reais, deve-se utilizar de técnicas de modelagem de sistemas que sejam capazes de abstrair do sistema real um modelo matemático que descreva a dinâmica de tal sistema (IWASE *et al.*, 2002).

Devido a relativa simplicidade de modelos lineares e sua limitada capacidade de representar sistemas complexos, muitos trabalhos focam no uso de sistemas não lineares, que embora mais complexos e onerosos de ponto de vista computacional, têm uma representatividade melhor de sistemas complexos (MORAIS, 2013).

Um sistema não-linear é aquele que não atende ao princípio da sobreposição de efeitos, desta forma sistemas não-lineares criam novas frequências em regime permanente, sendo assim, o sinal de saída pode apresentar frequências que não estão presentes no sinal de entrada (MORAIS, 2013).

Do ponto de vista matemático um sistema não-linear é aquele que não pode ser modelado por meio de representações matemáticas lineares (PEARSON, 2003).Em modelos não lineares o conhecimento do tipo de não linearidade (NL) possibilita a

seleção de modelos de Wiener ou Hammerstein (NL estática) ou Bi linear e Volterra (NL dinâmica) (MORAIS, 2013).

Quando não se possui o conhecimento do tipo de NL deve-se usar modelos como o NARX (Nonlinear Autoregressive Model with Exogenous Variables) ou o NCARMA (Nonlinear Controlled Auto-Regressive Moving Average) também conhecido como NARMAX (Nonlinear Auto-Regressive Moving Average Model with Exogenous Variables) (MORAIS, 2013).

Existem ainda várias formas de se representar o modelo matemático de um sistema, dentre as quais destacam-se as representações NARMAX/NCARMA polinomial (LEONTARITIS, 1985), racional (BILLINGS, 1989) e fracionário(MORAIS, 2013) sendo este último o de interesse neste trabalho.

3.2) Modelo NCARMA Fracionário

O modelo proposto em (MORAIS, 2013) consiste do Modelo NCARMA com expoentes reais. O modelo está representado na equação (2.1):

$$y(k) = \frac{\sum_i c_i \prod_{j=1}^{n_y} y(k-j) \prod_{r=1}^{n_u} u(k-r) \prod_{q=1}^{n_e} e(k-q)}{\sum_i d_i \prod_{j=1}^{d_y} y(k-j) \prod_{r=1}^{d_u} u(k-r) \prod_{q=1}^{d_e} e(k-q)} + e(k) \qquad (2.1)$$

Onde:

- C_i e d_i da Equação (2.1) são os Coeficientes;
- n_y, n_u,n_e,d_y,d_u e d_e são os Expoentes reais ou os graus de não linearidade de cada termo da Equação (2.1);
- $y(k-j)^{n_y}$ é um dos Termos com suas variações de linearidade e ordem.
- O conjunto de termos multiplicados por um coeficiente é denominado regressor; Exemplo: $C_i * y(k-j)^{n_y} * u(k-r)^{n_u} * e(k-q)^{n_e}$;

- $e(k)$ Representa o erro;

3.3) Algoritmos Genéticos e Diferencial Evolutivo

O AG é um algoritmo heurístico e auto adaptativo, utilizado para otimização de problemas globais. É formado pela abstração e simplificação dos processos biológicos da evolução, seleção natural e hereditariedade propostos por Darwin (GUIMARÃES, 2009).

O AG possui algumas vantagens tais quais: ser um algoritmo de busca e otimização global; não necessitar de conhecimento a priori do sistema a identificar, além de ser capaz de, após N gerações, chegar a uma solução otimizada (LI FU, 2013) desde que seja possível obter valores inicias para o algoritmo de forma aleatória.

O AG possui, porém, algumas desvantagens, como o problema dos mínimos locais, onde a solução converge para um mínimo ou máximo local, desta forma nunca alcançando o mínimo ou máximo global, a convergência prematura, onde a variabilidade genética é descartada e o sistema não consegue achar uma solução global, bem como não ser capaz de identificar ao mesmo tempo a estrutura e os parâmetros do sistema (LI FU, 2013).

Já o algoritmo DE, por outro lado, é um algoritmo de otimização eficiente, efetivo, simples, compacto e robusto, capaz de lidar com funções não lineares (CHIHA IBTISSEM, 2013) e que aplica métodos heurísticos a fim de evitar o problema dos mínimos locais presente no AG original (PAIVA, 1999), embora ele não se baseie em metáforas da natureza, inspirações vindas do processo de observação e questionamento de processos naturais, e sim em argumentos matemáticos e heurísticos(GUIMARÃES, 2009).

O DE diferencia-se do AG original por propor uma etapa de mutação diferenciada, de forma que nesta etapa são gerados novos vetores de parâmetros através da adição da diferença ponderada entre dois vetores de parâmetros a um terceiro indivíduo (GUIMARÃES, 2009). Esta adição de um novo vetor durante o processo de mutação é que adiciona a capacidade solucionar o problema dos mínimos locais no DE (MORAIS, 2013).

Os algoritmos de aprendizado de máquina, como o DE, podem ser aplicados em sistemas paralelos e/ou heterogêneos, que são sistemas que usam de processamento gráfico além do processamento em CPU (*Central Processing Unit*), visando melhorar seu desempenho de processamento (LUIGI BIANCO, 2015). Porém, no caso de sistemas heterogêneos, o desenvolvimento mostra-se mais complexo que as aplicações cientificas e gráficas geralmente aplicadas a GPU (*Graphical Processing Unit*), o que torna um desafio implementar sistemas de aprendizado de máquina em um hardware heterogênico (LUIGI BIANCO, 2015).

Devido a tal complexidade alguns autores preferem explorar o paralelismo a nível de CPU utilizando-se de processadores com mais de um núcleo de processamento (D.K. TASOULIS, 2004)(CHENG XIAO, 2011)(PAVEL KROMER, 2013).

Além do uso de sistemas heterogêneos e do paralelismo a nível de CPU, a literatura contém trabalhos que tratam do uso de métodos híbridos, que são aqueles que misturam diferentes técnicas, como uso de redes neurais, AG, DE, dentre outras, para identificar sistemas não lineares (LI FU, 2013)(PAIVA, 1999)(CHIHA IBTISSEM, 2013)(XIAOCEN XUE, 2012)(BIDYADHAR SUBUDHI, 2008)(LEONTARITIS, 1985)(BILLINGS, 1989)(J. SJÖBERG, 1995)(L. LJUNG, 2006)(T. HASTIE, 2001)(FRITZSON, 2004)(BOHLIN, 2006)(VAPNIK, 1998)(J.A.K. SUYKENS, 2002)(BARRON, 1989).

3.4) Paralelismo em Algoritmos Genéticos e DE

O algoritmo DE possui características que o tornam propício a se beneficiar de métodos e técnicas de programação paralela, como a criação da população inicial, a função aptidão, o cruzamento e a mutação, todos altamente paralelizáveis (D.K. TASOULIS, 2004)(PAVEL KROMER, 2013).

Devido a tais características, paralelizar o DE pode implicar em ganhos de performance significantes comparados a relativa baixa complexidade e ao alto grau de paralelização ao qual o algoritmo é suscetível (D.K. TASOULIS, 2004)(CHENG XIAO, 2011)(PAVEL KROMER, 2013).

A escolha de uma linguagem e de um ambiente de programação são essenciais para implementação de técnicas e algoritmos paralelos ao aplicativo final, pois influenciam na facilidade de desenvolvimento, teste e análise de cada algoritmo utilizado.

3.5) Pesquisas Relacionadas ao Trabalho

Este tópico trata de um breve levantamento e estudo de publicações recentes de áreas correlatas a este trabalho, a fim de localizar melhor este trabalho dentre as diversas áreas do conhecimento. Nele são descritos e analisados alguns dos trabalhos que mostram maiorproximidade com este trabalho.

Na publicação (XIAOCEN XUE, 2012), o autor propõe uma abordagem para auxiliar na identificação de sistemas dinâmicos complexos utilizando uma nova rede *neuro-fuzzy*, que envolve mesclar uma rede neural com lógica *fuzzy* e algoritmos de evolução diferencial. De forma que foi desenvolvida uma rede neural de quatro camadas

fuzzy como estrutura e o algoritmo DE é usado a fim de otimizar tal rede. Além disto o autor utilizou um fator de contribuição para encontrar e eliminar partes irrelevantes do algoritmo.

O autor validou seu algoritmo utilizando-se da identificação de processos termais, como o forno de *Box-Jenkins* e a caldeira de leito fluidizado circulante. O autor concluiu que o uso do algoritmo de evolução diferencial apresenta vantagens em relação ao algoritmo padrão de back-progation geralmente utilizado em redes *fuzzy*.

O trabalho (ADITYA KUMAR, 2012) propõe a criação de uma ferramenta capaz de atualizar códigos em C++ do padrão estabelecido em 2003 para novos padrões da versão 11 através da *demacrofication,* o que inclui o uso de expressões constantes, *perfect forwarding* e expressões lambda. A ferramenta atualiza vários macros, atualizando o código legado com novas ferramentas e características da versão 11 do C++. Para validar a ferramenta o trabalho utilizou-se de bibliotecas externas, a fim de saber se tais bibliotecas poderiam ser atualizadas, o trabalho conclui que, nas bibliotecas analisadas de 68% a 98% poderia ser atualizado utilizando-se o C++ versão 11. E discute o porquê de tais números não serem alcançados por ferramentas completamente automatizadas.

Já o trabalho (MARKUS KUSANO, 2013) apresenta uma ferramenta de geração de mutações para programas em C++ que utilizem de múltiplos núcleos. Tais mutações são utilizadas para fins de teste do código, e funcionam através da inserção de falhas sistemáticas no aplicativo. O trabalho evidencia a importância do desenvolvimento de ferramentas que auxiliem no desenvolver e testar códigos escritos em C++ que façam uso de múltiplos núcleos.

Já na publicação (LI FU, 2013), o autor discute métodos tradicionais de identificação de sistemas lineares e métodos modernos de identificação de sistemas não

lineares, onde são discutidos os métodos que utilizam técnicas como redes neurais, lógica *fuzzy*, algoritmos genéticos, algoritmos de otimização baseados em inteligência de enxames, algoritmo de identificação de modelo auxiliar, algoritmo *multi-innovation* e algoritmo hierárquico. Após o autor discutir tais métodos o mesmo faz uma análise de tendências e a perspectiva da identificação de sistemas. O autor conclui que as teorias de identificação de sistemas vêm sendo estudadas e aplicadas de forma cada vez mais profunda e abrangente, além de vir obtendo grande sucesso em diversos campos de pesquisa.

E na publicação (CHIHA IBTISSEM, 2013), o autor propõe um método hibrido baseado em evolução diferencial e algoritmos de treinamento que utilizam redes neurais para melhorar a performance de redes neurais na identificação de sistemas não lineares. Para obter tais resultados o autor utiliza do algoritmo de otimização local do conjugado dos gradientes*(local optimization algorithm of conjugate gradients)* combinado com o algoritmo de evolução diferencial, a fim de treinar uma rede perceptron multicamadas que identifica o sistema não linear. No final do estudo o autor conclui que o método proposto apresenta resultados promissores no sentido de obter um tempo de convergência da rede melhor e um erro de identificação menor.

Já em (SWATI SWAYAMSIDDHA, 2013) o trabalho utilizou-se de algoritmos genéticos, evolução diferencial e redes neurais de *Chebyshev*a fim de identificar um modelo não linear dinâmico e seus parâmetros. Os resultados foram comparados com o algoritmo padrão de *back-propagation* e validados utilizando-se duas plantas complexas, uma apresentando não linearidade na entrada, e outra apresentando não linearidade na saída.

Em (ZHEN LI, 2013) o trabalho destaca o domínio dos processadores com múltiplos núcleos tanto em *desktops* quanto em servidores e a dificuldade de se escrever

códigos paralelos para tais sistemas, desta forma o trabalho apresenta uma ferramenta automática capaz de identificar os potenciais paralelismos em códigos sequenciais. O trabalho conclui que a ferramenta desenvolvida foi capaz de identificar os principais trechos de código que poderiam ser paralelizados.

A publicação (HELON VICENTE HULTMANN AYALA, 2014) apresenta um procedimento para seleção de modelo e estimação de parâmetros para identificação de sistemas baseado em redes neurais com função de ativação de base radial e no algoritmo de evolução diferencial de livre busca. Além de adotar uma técnica de evolução diferencial em cascata e de decomposição de problemas. Para se obter os resultados promissores o autor utilizou duas populações distintas, uma para identificar os parâmetros e outra para identificar os atrasos. Os resultados do trabalho provarem-se válidos e promissores.

Em (NOSHADI, 2014) o trabalho tem como objetivo identificar um Sistema de mancais magnéticos ativo, para tanto utilizou-se um algoritmo genético. O sistema a ser identificado pode ser considerado uma caixa cinza e o modelo obtido está no domínio da frequência.

Já na publicação (SWATI SWAYAMSIDDHA, 2015), o autor utiliza-se de técnicas de evolução diferencial a fim de identificar sistemas não lineares com múltiplas entradas e múltiplas saídas (MIMO), no trabalho é proposta uma modificação ao algoritmo original que busca resolver o problema da convergência prematura. Além disso o autor faz uma comparação de performance de diferentes variantes do algoritmo DE a fim de encontrar a que melhor se adeque ao seu algoritmo modificado.

Primeramente foram analisados trabalhos ligados ao tema da identificação de sistemas, em seguida analisou-se trabalhos ligados ao C++ e a multiplos núcleos, a fim

de salientar melhor o que a comunidade científica está produzindo atualmente nestas áreas, foram pesquisados trabalhos que unissem o uso do algoritmos de computação evolutiva ou de inteligência artificial com a identificação de sistemas não lineares e o uso de múltiplos núcleos de processamento, porém não encontrou-se nenhuma publicação neste sentido.

O presente trabalho, então,pretende unir estas três áreas, criando um aplicativo genérico que utiliza do algoritmo diverencial evolutivo para identificar um sistema dinãmico não linear, seu modelo e seus parâmetros,de forma a utilizar recursos computacionais como o paralelismo da melhor forma possível.

Capítulo 4:
Metodologia

São apresentadas neste capítulo as técnicas estocásticas e determinísticas aplicadas na metodologia proposta. Além de apresentar o algoritmo como um todo e detalhar suas partes.

A metodologia apresentada neste trabalho baseia-se na metodologia proposta em (MORAIS, 2013), onde foram feitas alterações não na metodologia em si mas nos algoritmos utilizados em cada etapa do processo, ou seja, o geral permaneceu inalterado porém cada etapa, da alocação dos dados à validação do modelo, foi reescrita utilizando do padrão C++ em sua versão 14 e de bibliotecas matemáticas disponíveis globalmente, além de propor e utilizar um método de paralelismo mais eficiente que o método padrão. visando analisar os algoritmos computacionais utilizados por cada etapa a fim de reduzir o esforço computacional requerido pelo software de identificação de sistemas.

Desta forma o presente trabalho utilizou do modelo NCARMA Fracionário e do método de identificação propostos em (MORAIS, 2013) a fim de analisar os algoritmos propostos por tal método.

4.1) Visão geral do aplicativo:

O algoritmo DE utilizado para a identificação de sistemas, MIMO ou SISO, consiste das seguintes etapas, baseadas em (MORAIS, 2013):

- Etapa 1: Alocação e preparação dos dados de entrada e saída coletados;
- Etapa 2: Inicio do Algoritmo DE - Criação da população Inicial de tamanho N;

- Etapa 3: Mutação da população Inicial - É criada outra população derivada da inicial, porém modificada através de mutações para gerar diversidade populacional;
- Etapa 4: Cruzamento entre a população inicial e a que sofreu mutação -É criada uma terceira população baseada na troca de informações entre a original e a que sofreu mutação;
- Etapa 5: Agrupam-se as três populações resultantes e eliminam-se os piores regressores de cada equação utilizando-se o algoritmo ERR;
- Etapa 6: Calculam-se os coeficientes de todas as equações através do método dos mínimos quadrados e a aptidão através do método BIC;
- Etapa 7: São selecionadas as N melhores equações para próxima etapa;
- Etapa 8: Verifica-se se alguma das equações da população atende ao critério de parada ou se o mesmo é atingido por outro meio. Se sim, segue-se para a etapa nove, se não, volta-se a etapa 3;
- Etapa 9: Mede-se a capacidade do modelo de representar o sistema. Caso seja adequado o processo é finalizado e a equação encontrada é demonstrada, caso contrário os dados são ineficazes e devem ser substituídos, voltando-se a etapa um;

O fluxograma do aplicativo pode ser observado na figura 8.

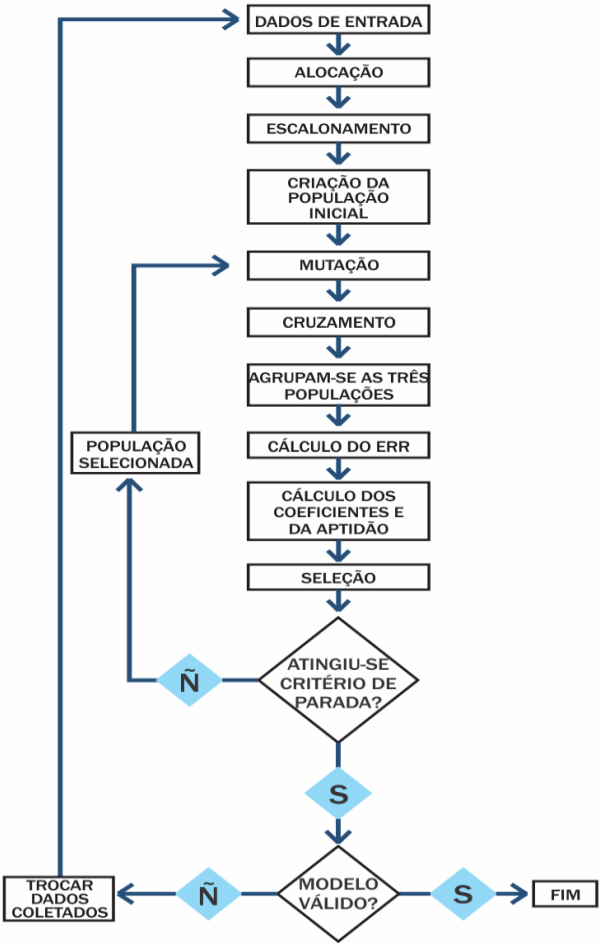

Figura 8: Fluxograma do aplicativo completo.

4.2) Detalhamento das Etapas do Fluxograma

Os detalhes da implementação de cada etapa encontram-se descritos nos subitens seguintes.

4.2.1) Métodos utilizados

Durante a etapa de coleta de resultados mostrou-se necessário um método eficiente, preciso e comprovado para se medir o tempo gasto para execução de determinadas tarefas. Para tanto é utilizada a biblioteca <*chrono.h*> presente no pacote *std* do C++ em sua versão 14;

Para fins de comparação e análise estatística optou-se pelo uso da média e do desvio padrão.

Dentro da estatística o significado de média refere-se ao ponto de equilíbrio de determinada frequência ou a concentração de dados de uma distribuição. O desvio padrão tem como objetivo demonstrar a regularidade referente a um conjunto de dados, de modo a apontar o grau de oscilação destes em comparação com a média dos valores do conjunto (MORETTIN; BUSSAB, 2010).

4.2.2) Alocação e preparação dos dados

Inicialmente os dados experimentais coletados são armazenados em um arquivo de texto simples (*.txt) com cabeçalho, então o aplicativo lê tal arquivo e o aloca em formato matricial sem alterar sua estrutura, onde a coluna *Time* representa os tempos de cada coleta e as colunas subsequentes representam cada variável coletada do sistema. As linhas representam os atrasos de amostragem de forma decrescente, ou seja, quanto maior o índice da linha menor o atraso. A estrutura do arquivo pode ser melhor visualizada na figura 9.

Time	y	u1	u2
1.00000000	0.00000000	1.33809714	1.10721411
2.00000000	0.00000000	1.33809714	1.10721411
3.00000000	0.00000000	1.33809714	1.10721411
4.00000000	0.33733908	1.33809714	1.10721411
5.00000000	0.35379408	1.33809714	1.10721411
6.00000000	0.35530428	1.33809714	1.10721411
7.00000000	0.35564118	1.33809714	1.10721411
8.00000000	0.35563874	1.33809714	1.10721411
9.00000000	0.35562496	1.33809714	1.10721411
10.00000000	0.35555220	1.33809714	1.10721411
11.00000000	0.35556091	2.71129831	0.63650889
12.00000000	0.25182413	2.71129831	0.63650889
13.00000000	0.25575490	2.71129831	0.63650889
14.00000000	0.27640484	2.71129831	0.63650889
15.00000000	0.27727250	2.71129831	0.63650889
16.00000000	0.27707211	2.71129831	0.63650889
17.00000000	0.27718801	2.71129831	0.63650889
18.00000000	0.27712819	2.71129831	0.63650889
19.00000000	0.27720140	2.71129831	0.63650889
20.00000000	0.27719792	2.71129831	0.63650889
21.00000000	0.27712104	2.47288661	1.23129434
22.00000000	0.29682389	2.47288661	1.23129434
23.00000000	0.28370349	2.47288661	1.23129434

Figura 9: Exemplo de formato dos dados no arquivo de entrada do aplicativo, variáveis coletadas do sistema.

Assim que são alocados na matriz os dados são escalonados entre 0.01 e 0.99, para melhor funcionamento do algoritmo DE, através da equação (3.1).

$$0.98 \times \left(\frac{Valor - Vmenor}{Vmaior - Vmenor}\right) + 0.01 \qquad (3.1)$$

Este escalonamento mostra-se necessário para que o algoritmo DE implementado neste trabalho funcione corretamente, desta forma todos os valores são escalonados para se manterem no intervalo entre 0.01 e 0.99, evitando assim que números negativos entrem no algoritmo, pois o algoritmo faz contas com expoentes menores que 1, e um número negativo com expoente menor que um encontra-se no domínio dos números imaginários e o domínio dos números imaginários não é contemplado neste trabalho.

4.2.3) Início do algoritmo DE

Nesta etapa é iniciado o algoritmo DE. É definido o número de indivíduos N da população inicial por saída, ou seja, cada saída gera uma população de N possíveis equações solução, já que o algoritmo é capaz de identificar um sistema MIMO. São definidas as variáveis de saída da matriz de dados, coletados na etapa um. E o atraso inicial máximo aceitável para as equações.

Cada equação é representada por uma estrutura de estruturas de dados. Tais estruturas e suas funções são descritas a seguir.

A estrutura básica é o termo, que é composto por três campos, um que armazena o valor do atraso, outro que armazena a coluna da variável e outro que armazena o valor do expoente.

A estrutura subsequente é o regressor, que contém um vetor de termos, um campo que armazena o coeficiente, um que armazena se o regressor está no denominador ou no numerador e ainda um último campo que armazena o ID (Número de Identificação).

Por fim, tem-se a estrutura da equação composta por: um vetor de regressores, um vetor para armazenar os valores da taxa de redução do erro, seis campos para armazenar o maior atraso da equação, a aptidão, o erro quadrático, a probabilidade da roleta e o acúmulo da probabilidade da roleta e um para dizer se a equação é válida ou não.

A estrutura da população pode ser vista na figura 10.

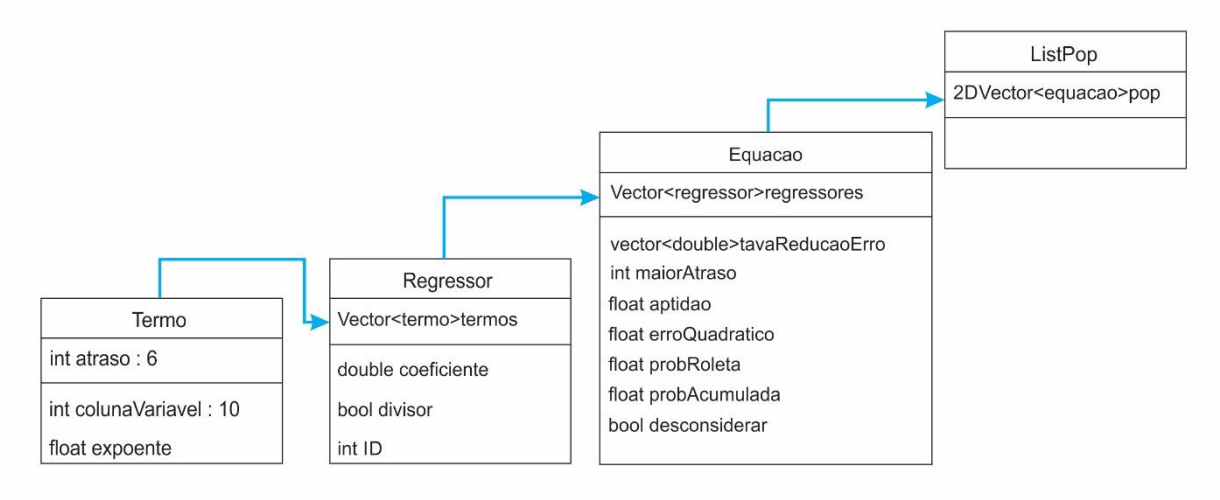

Figura 10: Estrutura da equação/cromossomo completa.

Para criar a população inicial são gerados entre 1 e 4 regressores de forma aleatória, que podem ser numeradores ou divisores, com ID aleatório entre 0 e 50. Cada um contém entre 1 e 4 termos aleatórios, com atrasos entre 1 e o atraso inicial máximo definido anteriormente, a coluna da variável aleatória, dentre as possíveis e o expoente inteiro entre 1 e 5. Então cada um desses regressores é colocado na equação gerada, a operação se repete até formar as N equações que formam a população inicial.

Para geração de números aleatórios contidos neste trabalho utilizou-se o algoritmo de *Mersenne Twister* padrão do C++ em sua versão 14, o MT19937, que utiliza distribuições uniformes padrões de números inteiros e reais, também presentes do padrão do C++ em sua versão 14(MATSUMOTO; NISHIMURA, 1998).

O MT19937 consiste de um gerador de números pseudo aleatórios baseados em uma distribuição qualquer. Desenvolvido em 1997 por Makoto Matsumoto e Nishimura Takuji (MATSUMOTO; NISHIMURA, 1998), fornece uma geração rápida e de qualidade de números pseudo aleatórios, tendo sido projetado especificamente para corrigir muitas falhas encontradas em algoritmos anteriores.

O MT19937 possui duas variantes, uma com 32 bits de comprimento de palavra, utilizada por esse trabalho, e outra com 64 bits de comprimento de palavra denominada MT19937 - 64 que gera uma sequência diferente.

4.2.4) Geração da população mutante

A operação de mutação consiste na aplicação do próprio operador sobre um conjunto de equações, a fim de se gerar uma nova equação baseada nas anteriores.

O operador de mutação inicialmente seleciona três equações da população: a melhor equação e duas aleatórias escolhidas pelo método de roleta.

Neste método cada indivíduo da população é representado na roleta proporcionalmente ao seu índice de aptidão. Assim, aos indivíduos com alta aptidão é dada uma porção maior da roleta, enquanto que aos de aptidão mais baixa é dada uma porção menor da roleta, desta forma o indivíduo 1 tem de 0 à X%, o indivíduo 2 tem de X% a Y% e assim sucessivamente. Em seguida gera-se um número aleatório entre 0 e 100%, e o indivíduo selecionado é aquele que contempla esse número. Cada indivíduo tem uma parte desses 100%, o tamanho dessa parte depende de sua aptidão. Somando-se todas as partes de todos os indivíduos obtêm-se 100% da roleta ilustrada na figura 11.

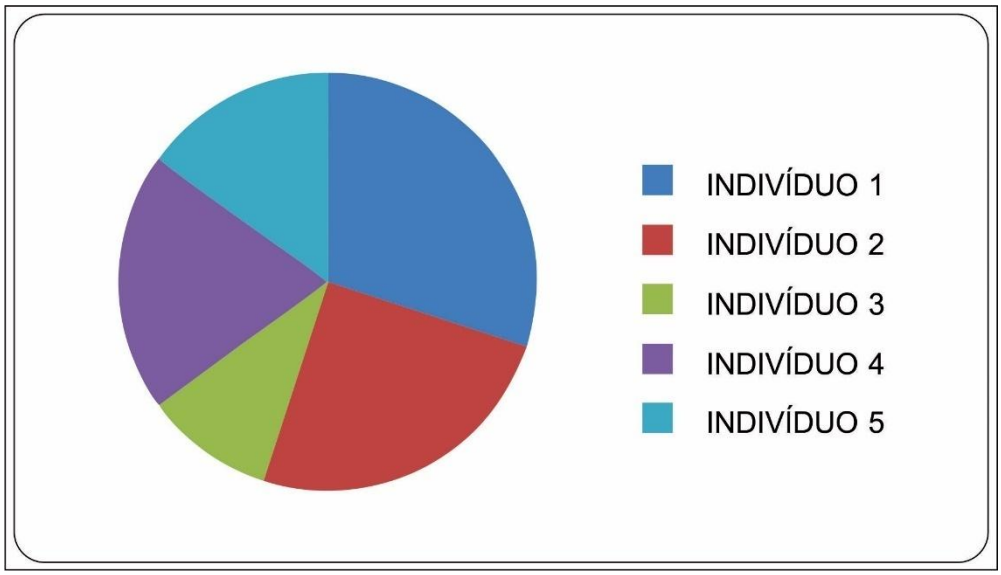

Figura 11: Exemplo de distribuição de indivíduos na roleta.

Após a seleção por roleta realiza-se a operação de mutação RAND-TO-BEST-MODIFICADO (MORAIS, 2013) presente na equação (3.2).

$$V_{i,G} = X_{best,G} + \mu\,(X_{new,G} - X_{best,G}) + F\,(X_{r_2,G} - X_{r_3,G}) \qquad (3.2)$$

Onde X representa as equações selecionadas eμ e F podem assumir valores aleatórios entre -2 e 2.As operações ocorrem nos expoentes dos termos de cada regressor, apenas caso os regressores tenham a mesma ID. Caso contrário a operação ocorre no regressor como um todo, operando regressores de uma equação em outra.

O RAND-TO-BEST é utilizado em problemas de otimização, porém no presente trabalho ele é empregado para resolver problemas de seleção de estrutura, o que consiste em um problema de combinação, desta forma sendo modificado para se adequar melhor (MORAIS, 2013).

A modificação se dá através da adição de uma nova equação denominada $X_{new,G}$, que é gerada aleatoriamente no momento da mutação. Desse modo, caso um

regressor necessário para representar o sistema em questão não esteja presente na população inicial, ele ainda pode surgir na equação $X_{new,G}$. Evitando assim que o algoritmo fique travado em um mínimo/máximo local. Já no RAND-TO-BEST tradicional, caso um regressor necessário não tenha sido criado na população inicial não aparecerá no resultado final.

A etapa de mutação pode ser visualizada na figura 12

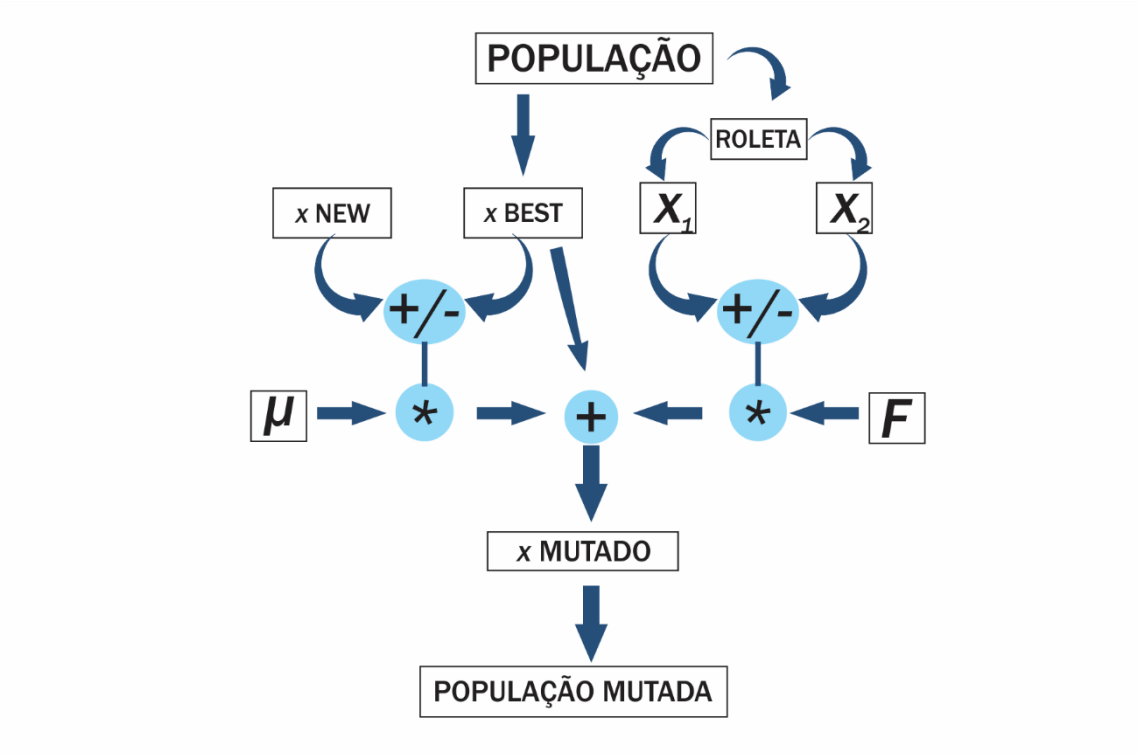

Figura 12: Fluxograma da etapa de mutação.

Conforme demonstrado pela figura 12, a diferença entre o fluxograma apresentado e o do DE tradicional é exatamente o algoritmo RAND-TO-BEST-MODIFICADO. A mutação consiste em selecionar os regressores presentes em $\mathbf{X_{new,G}}$que não estão presentes em $\mathbf{X_{best,G}}$, multiplicados pelo fator$\boldsymbol{\mu}$,e adicioná-los em$\mathbf{X_{best,G}}$, selecionar os regressores presentes em $\mathbf{X_{1,G}}$que não estão presentes em $\mathbf{X_{2,G}}$e adicioná-los em $\mathbf{X_{2,G}}$eposteriormente adicionar$\mathbf{X_{2,G}}$, multiplicado pelo coeficiente F,

em $X_{best,G}$, desta forma a equação $X_{best,G}$ recebe vários regressores diferentes aumentando seu espaço amostral, porém esta operação gera equações longas com muitos regressores, a fim de se contornar este problema é utilizado o algoritmo descrito na seção 3.2.6 - Seleção dos regressores.

4.2.5) Cruzamento

A técnica de mutação pode levar a uma convergência prematura da população, pois a melhor equação é sempre utilizada, mesmo que a cada mutação uma equação inteiramente nova seja introduzida. Para o cruzamento optou-se por trocar uma faixa de elementos de uma equação da população mutante por outra de uma equação da população original, minimizando assim o problema da convergência prematura da população.

O algoritmo inicia decidindo se a troca de elementos ocorrerá entre termos ou entre regressores, ou ainda se não ocorrerá. Caso ocorra, é selecionado um ponto a partir do qual todos os elementos são trocados. A figura 13 ilustra um cruzamento de termos e a figura 14 ilustra um cruzamento de regressores.

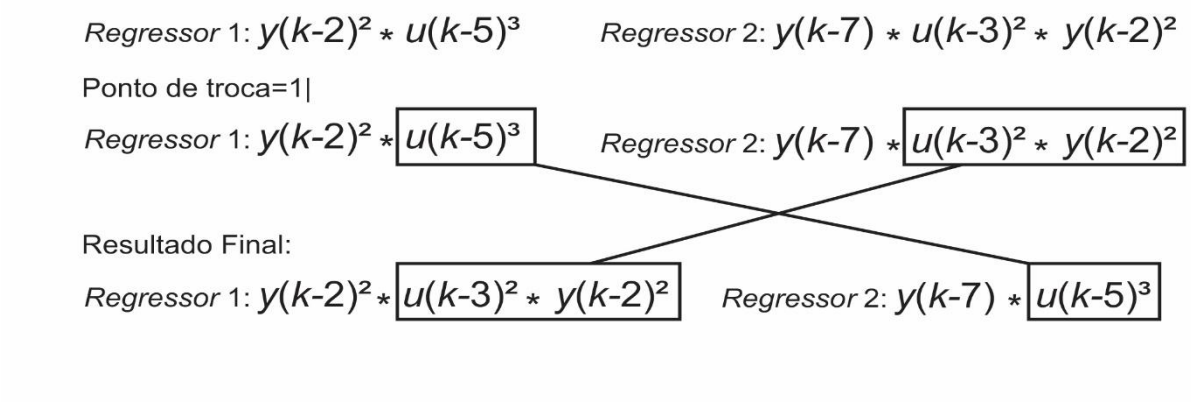

Figura 13: Exemplo de cruzamento de termos.

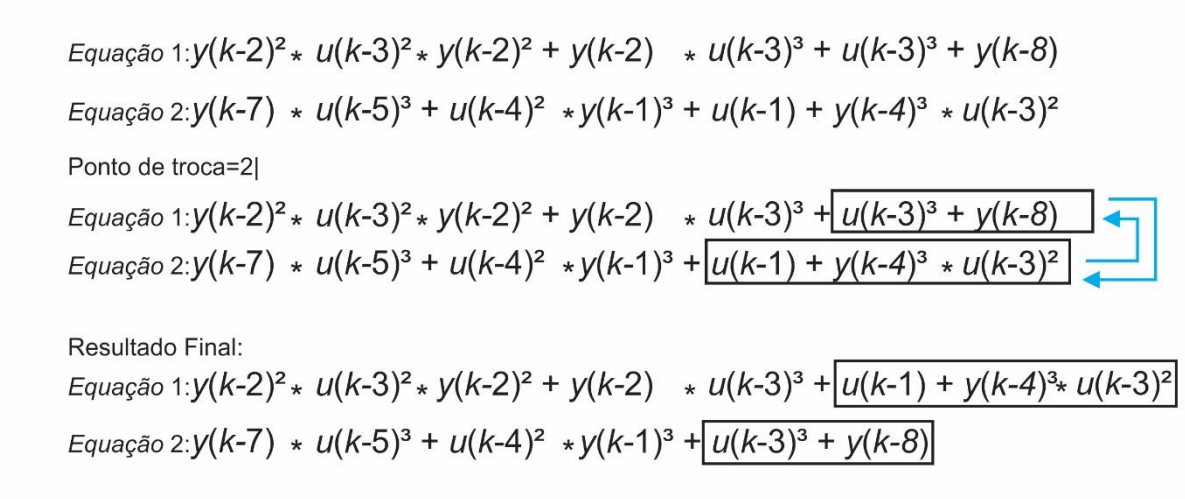

Figura 14: Exemplo de cruzamento de regressores.

4.2.6) Seleção dos regressores

Após as etapas de mutação e cruzamento as novas equações passam pelo processo de seleção dos regressores, onde cada regressor da equação é testado e, caso seja considerado ineficiente, é removido da equação. Esta etapa serve para otimizar as novas equações.

A seleção adequada da estrutura é fundamental para a representação global da dinâmica do sistema, sendo realizada em duas partes:

- Seleção dos grupos de termos que devem pertencer ao modelo;
- Dentro desses grupos quais termos farão parte do modelo.

Deve-se levar em conta na representação do sistema a escolha adequada dos grupos de termos, para evitar efeitos dinâmicos espúrios, e o número de termos, que deve ser o menor possível visando evitar problemas de instabilidade numérica. (AGUIRRE, 1994). A questão é definir quantos e quais termos devem ser selecionados dentre o universo de possibilidades.

Dentre as técnicas para seleção de termos, duas se destacam: a construtiva e a de eliminação(AGUIRRE, 1994)(KREMLIOVSKY, 1999). Como a metodologia se baseia no proposto em (MORAIS, 2013), optou-se pelo uso da técnica de eliminação, onde é escolhido um modelo de teste, são estimados os parâmetros iniciais e a cada passo são descartados os termos menos significativos.

Para combinar de forma eficiente a seleção de estrutura ortogonal e a estimativa de parâmetros utilizou-se o algoritmo ortogonal com regressão direta (MORAIS, 2013), onde a principal técnica utilizada na determinação da estrutura é a taxa de redução de erro "ERR" (BILLINGS, 1989). O ERR consiste em associar um índice na contribuição de cada termo na variância da saída da equação, é obtido através da fatoração ortogonal da matriz.

Com a informação obtida pelo ERR define-se quais termos serão inseridos no modelo. Também é levado em conta o estudo dos efeitos da sobre parametrização, que ocorre quando a equação contém parâmetros em excesso(AGUIRRE; BILLINGS, 1995) e de grupos de termos de pontos fixos de modelos polinomiais (AGUIRRE; MENDES; BILLINGS, 1996) nas estruturas de modelos NARMAX.

Ao final do processo os termos considerados pouco significativos para a equação são descartados e a equação segue menor e otimizada.

4.2.7) Cálculo dos coeficientes e da aptidão

Para o cálculo dos coeficientes utilizou-se o método dos mínimos quadrados estendidos e para o cálculo da aptidão utilizou-se o Critério de Informação Bayesiana (BIC) (MORAIS, 2013).

Nos algoritmos de mínimos quadrados, busca-se reduzir de forma iterativa a correlação do vetor de resíduos através da incorporação de um modelo de ruído na

matriz de regressores, onde os parâmetros dos termos e os parâmetros dos ruídos são estimados até que sua polarização seja reduzida a níveis aceitáveis(MORAIS, 2013).

O método utilizado para se calcular os coeficientes não consegue lidar com regressores que estão no denominador, por isso realizou-se uma manobra matemática, denominada pseudo linearização com objetivo de transferir os regressores do denominador para o numerador a matemática completa pode ser visualizada no Apêndice A.

Para o cálculo do BIC é necessário calcular o Somatório do Erro Quadrático - SSE (*Sum of Squared Error*) demonstrado na equação (3.3) e a função custo básica dos mínimos quadráticos demonstrada pela equação (3.4). A BIC pode ser visualizada na equação (3.5).

$$\mathrm{SSE} = \sum_{t=1}^{N} [y(t) - \hat{y}(t)]^2 \tag{3.3}$$

$$J_n = \frac{1}{N} * \mathrm{SSE} \tag{3.4}$$

$$\mathrm{BIC} = N * \ln[J_n] + p * \ln[N] \tag{3.5}$$

Onde: $y(t)$ é o valor desejado da amostragem, $\hat{y}(t)$ é o valor obtido pelo modelo avaliado, N é o número de medidas e p a quantidade de regressores.

Para auxiliar o cálculo da aptidão, levando em conta o tamanho real da equação, realizou-se um pequeno ajuste no índice p, onde ele passa a representar não só o número de regressores, mas também o número total e termos da equação, além de ganhar um peso maior, desta forma é considerado a tamanho real da equação em sua aptidão, quanto mais termos tem a equação maior será sua aptidão. Conforme equação (3.6)

$$p_{Novo} = 2 \times NTermos \times p \qquad (3.6)$$

4.2.8) Seleção dos Cromossomos

Nesta etapa do processo tem-se três populações, a inicial, a que sofreu mutação e a resultante do cruzamento. Para continuar com o algoritmo a nova população deve conter sempre o mesmo tamanho, logo devemos escolher apenas N elementos das 3*N equações totais, já que cada população contribui com N indivíduos. Para isso são unidas as três populações em uma única matriz em seguida as equações na matriz são ordenadas de forma crescente através do BIC de cada equação, pois quanto menor o valor da aptidão mais apta é a equação. Após esta etapa,são escolhidas as N primeiras equações para seguirem para a próxima etapa do algoritmo DE.

4.2.9) Critério de parada

O algoritmo DE necessita de um critério de parada, caso contrário ele fica em estado de *loop* indefinidamente. O processo finaliza quando atinge a condição de não progredir um determinado valor em um determinado número de iterações. Por padrão se a aptidão do melhor cromossomo/equação não sofrer um progresso em seu BIC em 1000 iterações o processo de identificação é finalizado.

4.2.10) Validação do modelo

Após o algoritmo finalizar, ele apresenta como resultado uma equação que representa o sistema em questão, deve-se então medir o quão bem ela o representa. Para

isso pode-se utilizar dois tipos de validação, validação estatística e a validação dinâmica (BILINGS S.A., 1983)(BILLINGS S.A., 1986)(BILLINGS S.A., 1993).

A validação estatística utiliza-se de testes de auto correlação, correlação entre resíduos de identificação e correlação entre sinal de saída e sinal de entrada para validar o modelo. O mesmo deve apresentar os resíduos de identificação sem correlação entre si e entre os sinais de saída e de entrada. Deve-se então verificar as seguintes identidades na validação de modelos não lineares (BILLINGS S.A., 1993)(BILLINGS S.A., 1986)(BILINGS S.A., 1983):

$$\Phi_{\xi\xi}(\tau_1) = E\{\xi(t)\xi(t-\tau_1)\} = \delta(\tau_1), \tag{3.9}$$

$$\Phi_{\xi u}(\tau_1) = E\{\xi(t)u(t-\tau_1)\} = 0, \forall\tau_1 \tag{3.10}$$

$$\Phi_{u^2\xi}(\tau_1) = E\{(u^2(t) - E\{u^2(t)\})\xi(t-\tau_1)\} = 0, \forall\tau_1 \tag{3.11}$$

$$\Phi_{u^2\xi^2}(\tau_1) = E\{(u^2(t) - E\{u^2(t)\})\xi^2(t-\tau_1)\} = 0, \forall\tau_1 \tag{3.12}$$

$$\Phi_{\xi'\xi'}(\tau_1) = E\{(\xi(t) - E\{\xi(t)\})(\xi(t-\tau_1) - E\{\xi(t) \quad\}) \quad\} = 0, \forall\tau_1 \tag{3.13}$$

$$\Phi_{\xi'\xi^{2'}}(\tau_1) = E\{(\xi(t) - E\{\xi(t)\})(\xi^2(t-\tau_1) - E\{\xi^2(t) \quad\}) \quad\} = 0, \forall\tau_1 \tag{3.14}$$

$$\Phi_{\xi^{2'}\xi^{2'}}(\tau_1) = E\{(\xi^2(t) - E\{\xi^2(t)\})(\xi^2(t-\tau_1) - E\{\xi^2(t) \quad\}) \quad\} = \delta(\tau_1), \forall\tau_1 \tag{3.15}$$

$$\Phi_{(y\xi)'\xi^{2'}}(\tau_1) = E\{(y(t)\xi(t) - E\{y(t)\xi(t)\})(\xi^2(t-\tau_1) - E\{\xi^2(t)\})\} = 0\,, \forall\tau_1 \tag{3.16}$$

$$\Phi_{(y\xi)'u^{2'}}(\tau_1) = E\{(y(t)\xi(t) - E\{y(t)\xi(t)\})(u^2(t-\tau_1) - E\{\xi u^2(t)\})\} = 0\,, \forall\tau_1 \tag{3.17}$$

Com base no comprimento do registro de dados disponíveis, N, é definido um intervalo probabilístico de 95% de confiança, no qual as funções de correlação devem se manter para serem consideradas nulas. Para funções normalizadas os limites do intervalo de confiança de 95% são: ± 1,96 N(BILINGS S.A., 1983).

Através da validação estatística, garante-se que não existem correlações não modeladas nos resíduos do sistema. Porém ela sozinha não assegura que o modelo

validado apresenta o mesmo comportamento dinâmico do sistema original (AGUIRRE, 1994).

Para verificar o comportamento dinâmico é necessária a validação dinâmica, que consiste em aplicar dados coletados diferentes dos usados na identificação do modelo obtido e verificar o erro.

4.3) Comparativo de bibliotecas matemáticas

A necessidade de cálculos matemáticos complexos envolvendo matrizes, implica em uma complexidade de algoritmo e de programação elevada. Como o C++ 14 oferece poucos recursos para tais cálculos, existe uma demanda por bibliotecas que o façam, pois, escrever os algoritmos para fazer tais cálculos demanda tempo e esforço desnecessários.

Existem a disposição várias bibliotecas para realizar cálculos matemáticos, dentre as quais se destacam as gratuitas Open Blas, GSL e Armadillo. Cada uma tem suas particularidades, como por exemplo: a Open Blas é muito complexa e de difícil implementação, enquanto que a GSL apresenta uma interface mais amigável, facilitando a mantenabilidade do código sendo uma das bibliotecas usadas pelo software MatLab® já a Armadillo é dinâmica e de melhor facilidade de uso, se assemelhando a códigos de MatLab®, porém escrita em C++.

O presente trabalho com o objetivo de se comparar as bibliotecas realizou alguns testes de performance, medindo o tempo médio de execução de determinados algoritmos inclusos nas bibliotecas e avaliando a média e o desvio padrão dos testes.

Tais testes encontram-se no capítulo de resultados.

4.4) Linguagem e Ambiente de Programação

A performance do sistema de aprendizado de máquina é algo relevante, uma vez que a maioria dos algoritmos de aprendizagem são complexos e onerosos, do ponto de vista do esforço computacional necessário para realizá-los (MORAIS, 2013). O uso de um ambiente de programação que minimize as perdas de performance sem minimizar a capacidade de programação é algo essencial no desenvolvimento de um sistema de aprendizagem de máquina, como é o caso do DE (LUIGI BIANCO, 2015).

Optou-se pelo uso do framework Qt pelo fato de o mesmo apresentar uma versão gratuita, ser confiável e dinâmico, o mesmo utilizado por (MORAIS, 2013)porém o presente trabalho não utiliza bibliotecas fornecidas pelo Qt a fim de manter o código o mais portável possível, e da linguagem de programação C++ 14, devido a sua performance, simplicidade de programação e ampla gama de bibliotecas gratuitas e consolidadas disponíveis para uso.

O C++ em sua versão 14 oferece uma gama de vantagens em relação ao seu predecessor o C++ em sua versão 98, pois tornou-se mais eficiente, melhorou sua confiança, credibilidade, legibilidade e consequentemente a facilidade de manutenção e performance, além de trazer novos recursos e bibliotecas que se mostraram muito úteis no desenvolvimento do trabalho (ADITYA KUMAR, 2012).

4.5) Detalhamento da classe gerenciadora dos *threads*

Com a tendência da utilização de processadores com múltiplos núcleos torna-se imprescindível o uso de técnicas de programação que extraiam o melhor desempenho de tais processadores(LEI CHAI, 2007). Desta forma o uso e gerenciamento de vários

threads mostra-se promissor no aumento da performance de algoritmos paralelizáveis (ALKALAJ, 1991).

Com base no exposto,o aplicativo utiliza de vários *threads* a fim de melhorar seu desempenho. Para gerenciá-los e controlá-los de forma otimizada foi criada uma classe gerenciadora, com objetivo de que todo *thread* lançado pelo aplicativo seja gerenciado apenas por tal classe.

O algoritmo implementado para tal gerenciamento, consiste em criar o máximo de *threads* suportados pelo processador do sistema ao se iniciar o aplicativo, além de mantê-los em estado de *sleep* até que os mesmos sejam demandados por tarefas.

À medida que o aplicativo necessita executar tarefas ele deve primeiramente criá-las e empacotá-las, logo em seguida colocá-las em uma pilha. Desta forma os *threads* devem apenas verificar a pilha e solicitar para si as tarefas. Assim sendo, enquanto houver tarefas os *threads* se mantêm ocupados. Quando não há tarefas, voltam ao estado de *sleep*. Assim, evita-se criá-los e terminá-los toda vez que uma tarefa precisar ser executada em paralelo.

O processo de gerenciamento pode ser melhor visualizado nas figuras 15 e 16.

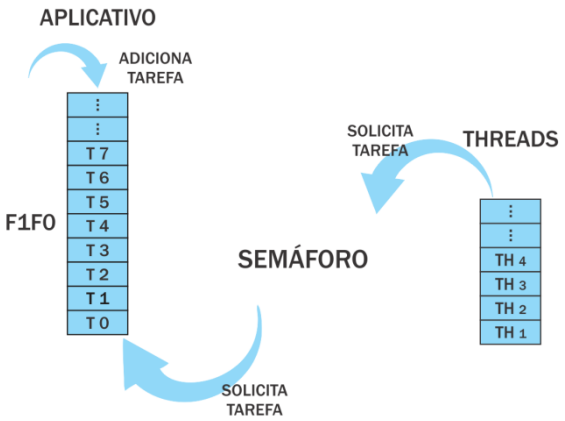

Figura 15: Processo de gerenciamento dos *threads*.

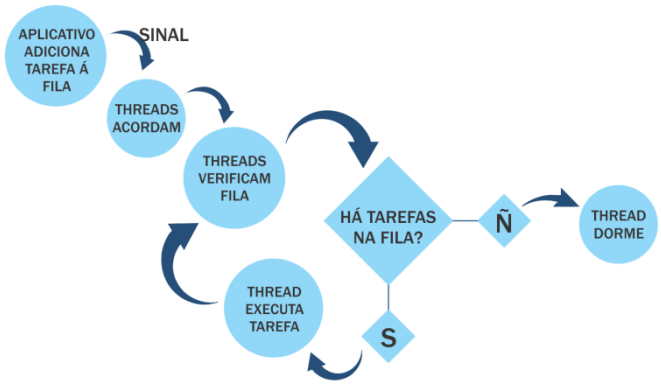

Figura 16: Fluxograma do gerenciamento dos *threads*.

Quando o *thread* solicita uma tarefa para si, a mesma é removida da pilha, daí a necessidade do semáforo para realizar o acesso à pilha, o que evitará que duas *threads* tomem para si a mesma tarefa.

Como pode ser visualizado na figura 16, o ato do processo colocar tarefas empacotadas na fila gera um sinal que acorda os threads, os mesmos então verificam a fila através de uma estrutura de semáforo e solicitam as tarefas para si, removendo-as da pilha. Este processo continua enquanto houver tarefas na pilha, quando as tarefas acabarem os *threads* voltam a estado de *sleep*. Como demonstrado nas figuras o processo é cíclico e depende das duas partes.

Capítulo 5:
Resultados

5.1) Considerações iniciais

Neste capítulo aplicou-se a metodologia proposta em vários algoritmos, a fim de medir o desempenho de diferentes técnicas.

Para análise do aplicativo utilizou-se o sistema SISO e o sistema MIMO propostos no capítulo 4.

Testou-se os algoritmos deste capítulo sempre no mesmo ambiente computacional, com uso de um computador com processador Intel Core I7 - 3612QM a 2.8 Ghz, com 8Gb de memória *Ram* a 1600 Mhz.

Buscou-se executar os testes comparativos sempre em sequência, alternando sua ordem de forma a homogeneizar ao máximo as condições do sistema operacional e do computador.

Os métodos padrões utilizados para efeito de comparação, podem ser encontrados em muitos livros de programação, sendo que o livro utilizado é o (ZIVIANE, 2005)

A análise do desvio padrão depende da média, ou seja, para concluir o quão bom está o desvio padrão deve-se considerar a média.(CORREA, 2003)

5.2) comparativo entre GSL e Armadillo

A fim de se selecionar a biblioteca mais adequada para o projeto foram realizados testes de desempenho para os algoritmos de cálculo da média, da variância e do mínimo e máximo de um vetor, relevantes para o aplicativo, foram analisadas as bibliotecas GSL e Armadillo, a biblioteca OpenBlas não é analisada devido a seu alto

grau de complexidade de uso, o que torna a mantenabilidade do código ineficiente, embora a GSL utilize a OpenBlas internamente.

Os testes foram realizados fora do ambiente do software de identificação de sistemas, a fim de se testar apenas a capacidade das bibliotecas, posteriormente testes foram realizados dentro do ambiente do software de identificação de sistemas desenvolvido por esse trabalho.

5.2.1) Teste de cálculo da média

Testou-se amostras de 10, 50, 100, 500, 1000, 2000, 3000, 4000, 5000 e 10000 elementos.

Tabela 10: Teste Armadillo Vs GSL – Média.

Nº AMOSTRAS	ARMADILLO	GSL	DIFERENÇA %
10	1955 ns	5376 ns	174,98%
50	1955 ns	4399 ns	125,01%
100	1955 ns	7819 ns	299,94%
500	3910 ns	16128 ns	312,48%
1000	3910 ns	25902 ns	562,45%
2000	7820 ns	47896 ns	512,48%
3000	9286 ns	67933 ns	631,56%
4000	11730 ns	91392 ns	679,13%
5000	14173 ns	110452 ns	679,31%
10000	22970 ns	214063 ns	831,92%

O uso da biblioteca Armadillo para cálculo da média mostrou-se mais eficiente em todos os testes realizados, sendo que quando maior a amostra melhor se comporta o Armadillo em relação a GSL. Resultado demonstrado na figura 17.

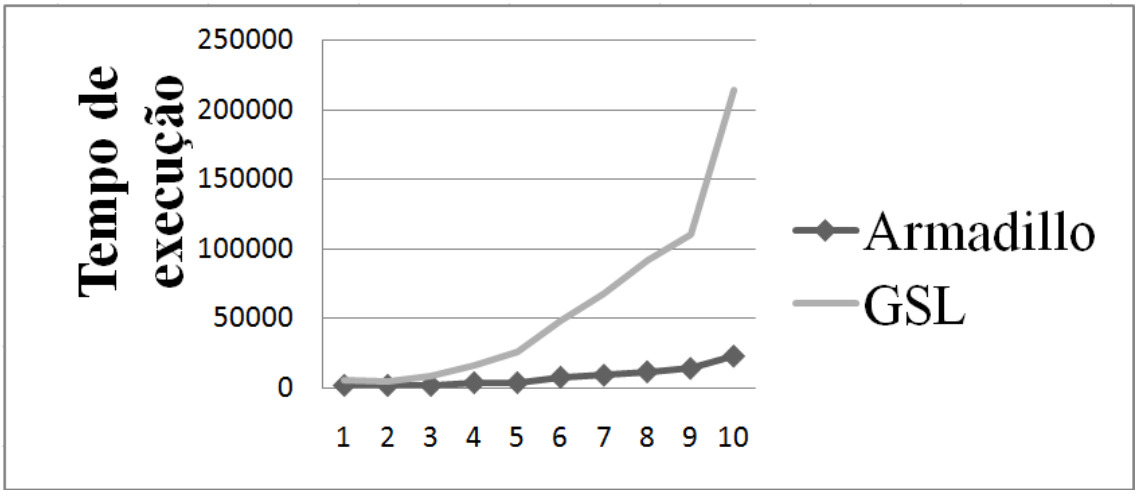

Figura 17: Teste Armadillo Vs GSL – Média.

A figura 17 demonstra que a biblioteca Armadillo apresenta tempos inferiores a biblioteca GSL, e evidencia que quanto maior a quantidade de amostras melhor se comporta o Armadillo em relação a GSL.

5.2.2) Teste de cálculo da Variância

Testou-se amostras de 10, 50, 100, 500, 1000, 2000, 3000, 4000, 5000 e 10000 elementos.

Tabela 11: Teste Armadillo Vs GSL – Variância.

Nº AMOSTRAS	ARMADILLO	GSL	DIFERENÇA %
10	1466 ns	1466 ns	0%
50	4399 ns	2932 ns	-33,34%
100	5376 ns	5376 ns	0%

500	15150 ns	21015 ns	38,71%
1000	18572 ns	41053 ns	121,04%
2000	30301 ns	82595 ns	172,58%
3000	42030 ns	123648 ns	194,18%
4000	56203 ns	162746 ns	189,56%
5000	63046 ns	203310 ns	222,47%
10000	130491 ns	406132 ns	211,23%

O uso da biblioteca Armadillo para cálculo da variância mostrou-se mais eficiente para testes com amostras maiores, e menos eficiente ou com eficiência equivalente para testes com amostras de 10, 50 e 100, sendo que quando maior a amostra melhor se comporta o Armadillo em relação a GSL, a diferença negativa da amostra 50 indica que para esta amostra a biblioteca GSL se saiu melhor. Resultado demonstrado pela figura 18.

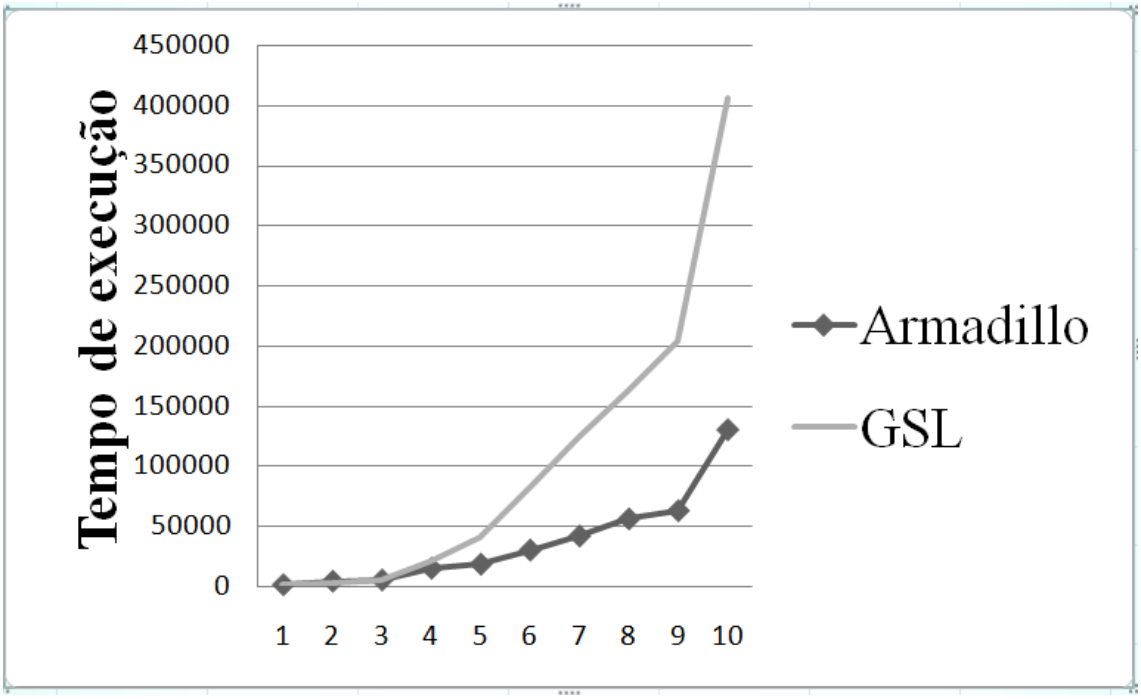

Figura 18: Teste Armadillo Vs GSL – Variância.

A figura 18 demonstra que a biblioteca Armadillo apresenta tempos inferiores a biblioteca GSL para quantidade de amostras superiores a 100, e evidencia que quanto maior a quantidade de amostra melhor se comporta o Armadillo em relação a GSL.

5.2.3) Teste de cálculo do mínimo e do máximo de um vetor

Testou-se amostras de 10, 50, 100, 500, 1000, 2000, 3000, 4000, 5000 e 10000 elementos.

Tabela 12: Teste Armadillo Vs GSL - Máximo e Mínimo de um vetor.

Nº AMOSTRAS	ARMADILLO	GSL	DIFERENÇA %
10	978 ns	5376 ns	449,69%
50	1467 ns	5865 ns	299,79%
100	1954 ns	4398 ns	125,07%
500	3421 ns	7331 ns	114,29%
1000	6842 ns	12707 ns	85,72%
2000	11240 ns	17594 ns	56,53%
3000	16128 ns	24925 ns	54,54%
4000	21504 ns	41053 ns	90,90%
5000	26391 ns	39587 ns	50,00%
10000	50827 ns	77219 ns	51,92%
1.000.000	5173669 ns	7061135 ns	36,48%

O uso da biblioteca Armadillo para cálculo do máximo e do mínimo mostrou-se mais eficiente, sendo que quando maior a amostra mais próximo se comporta o Armadillo em relação a GSL, porém mesmo para amostras na casa de 1.000.000, o

Armadillo ainda é melhor, no caso Armadillo levou 5173669 ns e a GSL levou 7061135 ns, 36,48% mais rápida ou 1,36 vezes. Resultado demonstrado na figura 19.

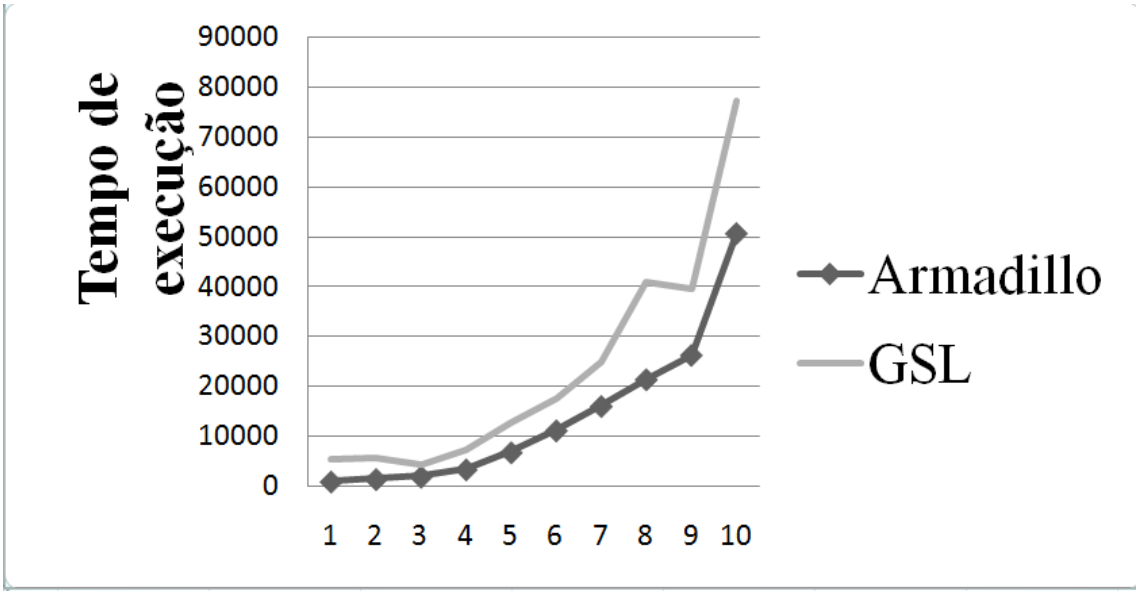

Figura 19: Teste Armadillo Vs GSL - Máximo e Mínimo de um vetor.

A figura 19 demonstra que a biblioteca Armadillo apresenta tempos inferiores a biblioteca GSL em todos os testes realizados para cálculo de máximo e mínimo de um vetor, e evidencia que ela sempre se sai melhor que a GSL, a amostra de 1.000.000 é deixada de fora do gráfico, pois como seu tempo de execução é muito superior as demais amostras a mesma distorce o gráfico.

Além de geralmente ser mais rápida que a GSL, a Armadillo auxilia em muito no desenvolvimento dos algoritmos mais complexos devido a sua facilidade de uso e consequentemente uma melhor mantenabilidade do código.

5.3) Comparativos de métodos utilizados na alocação e preparação dos dados

A leitura do arquivo .txt fez-se de forma sequencial devido à natureza do acesso aos dados guardados em HD. Tendo em vista que o mesmo arquivo não pode ser acessado de forma simultânea.

5.3.1)Algoritmo de escalonamento

O uso de um algoritmo de escalonamento mostrou-se necessário para que o DE trabalhe melhor, também para que em vários cálculos matemáticos não haja divisão por zero, mesmo que sua utilização introduza um pequeno erro no sistema.

5.3.1.1) Média

Para o algoritmo de escalonamento precisou-se calcular a média e os valores máximos e mínimos de cada vetor. Para a média testou-se dois algoritmos, um com uso da biblioteca Armadillo e outro com uso de funções padrões do GSL, ambos listados na figura 20.

Método Armadillo:

```
for(intj=0;j<size;j++)
{
medias(j-1)+=valor;
mediasQuadraticas(j-1)+=valor*valor;
}
medias=medias/linhas;
mediasQuadraticas=mediasQuadraticas/linhas;
```

Método GSL:

```
doubley=0;
for(inti=0;i<linhas;i++)
{
y+=gsl_matrix_get(matriz,i,1)/linhas;
}
```

Figura 20: Algoritmos testados para cálculo da média.

O resultado de cada método pode ser visto na Tabela 13 e na Tabela 14

Tabela 13: Resultado do teste de cálculo da média para 500 amostras de entrada.

ARMADILO	GSL	DIFERENÇA %
3910 ns	4887 ns	24,98%
2443 ns	3910 ns	60,04%
1466 ns	3910 ns	166,71%
978 ns	4398 ns	349,69%
977 ns	3910 ns	300,20%
987 ns	4398 ns	345,59%
2444 ns	5376 ns	119,96%
977 ns	5376 ns	450,25%
1955 ns	4399 ns	125,01%
1466 ns	4399 ns	200,06%
MÉDIAS 1760,3 ns	4496,3 ns	155,42%
DESVIO PADRÃO 954,69	554,76	140,47

Tabela 14: Resultado do teste de cálculo da média para 480.000 amostras de entrada.

ARMADILLO:	GSL	DIFERENÇA %
977 us	3248 us	232,44%
1467 us	3286 us	123,99%
977 us	3256 us	233,26%
978 us	3688 us	277,09%
977 us	3258 us	233,46%

976 us	3292 us	237,29%
978 us	3246 us	231,90%
977 us	3198 us	227,32%
975 us	3248 us	233,12%
977 us	3258 us	233,46%
MÉDIAS 1025,9 us	3297,8 us	221,45%
DESVIO PADRÃO 154,9	139,42	38,63

O método Armadillo é mais rápido em ambos os casos, no primeiro para 500 amostras de entrada ele é, em média, 155,42% mais rápido ou 2,55 vezes, sendo que no segundo caso é, em média, 221% mais rápido ou 2,21 vezes.

O desvio padrão, de 140,47 da diferença, para 500 amostras é elevado devido a quantidade de amostras pequena, pois como o tempo de execução é pequeno as variações causadas pelo ambiente computacional representam uma parcela considerável, causando uma oscilação nos tempos resultantes, o que é reduzido com o aumento da quantidade de amostras e conseqüentemente o aumento do tempo de execução. Já para uma amostra de 480.000 obteve-se um desvio padrão de 38,63.

Mesmo com um desvio padrão elevado os resultados da biblioteca Armadillo são promissores. Optou-se então por utilizar esse método no aplicativo final. Resultados demonstrados pelas figuras 21 e 22.

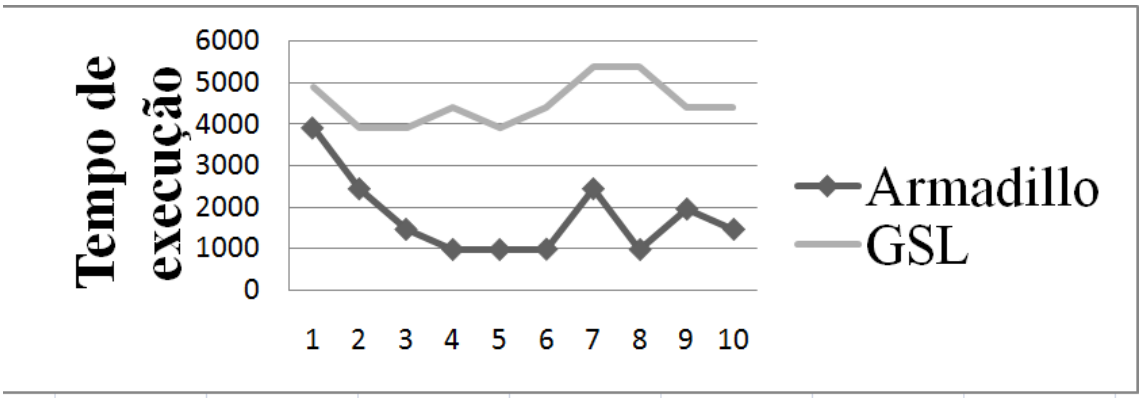

Figura 21: Resultado do teste de cálculo da média para 500 amostras de entrada.

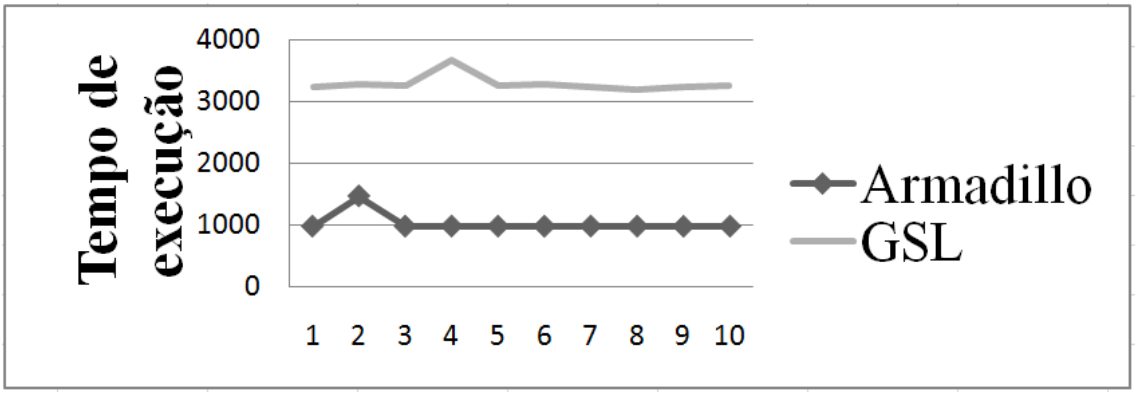

Figura 22: Resultado do teste de cálculo da média para 480.000 amostras de entrada.

As figuras 21 e 22 demonstram que a biblioteca Armadillo apresenta tempos inferiores a biblioteca GSL em todos os testes realizados para o cálculo da média dentro do ambiente do software de identificação de sistemas, e evidencia que ela sempre se sai melhor que a GSL.

5.3.1.2) Máximos e mínimos de um vetor

Para o cálculo dos valores máximos e mínimos de um dado vetor, testou-se dois métodos: um GSL e outro da biblioteca Armadillo. Os métodos podem ser visualizados na figura 23. Os resultados de cada método podem ser vistos nas Tabelas 15 e 16.

Método Armadillo:

```
maximos=max(*matriz,0);
minimos=min(*matriz,0);
```

Método GSL:

```
for(inti=0;i<linhas;i++)
{
for(intj=0;j<colunas;j++)
{
valor=gsl_matrix_get(matriz,i,j);
if(valor>gsl_vector_get(maximos,j-1))
        {gsl_vector_set(maximos,j-1,valor);} }}
```

Figura 23:Algoritmos testados para cálculo do Máximo e do Mínimo de um vetor.

Tabela 15: Resultado do teste do cálculo do máximo e mínimo para 500 amostras de entrada.

ARMADILLO	MÉTODO PADRÃO	DIFERENÇA %
5376 ns	10752 ns	100%
5376 ns	10752 ns	100%
5864 ns	10752 ns	83,35%
5376 ns	23947 ns	345,44%
8308 ns	12218 ns	47,06%
5376 ns	11241 ns	109,09%
5376 ns	12707 ns	136,36%
5865 ns	10752 ns	83,32%
7331 ns	11241 ns	53,33%
5376 ns	12218 ns	127,26%
MÉDIAS 12658 ns	5962,4 ns	112,29%
DESVIO PADRÃO 1025,08	4032,89	84,65

Tabela 16: Resultado do teste do cálculo do máximo e mínimo para 480.000 amostras de entrada.

ARMADILLO	MÉTODO PADRÃO	DIFERENÇA %
7702us	23325 us	202,84%
7453us	23817 us	219,56%
7671us	24690 us	221,86%
7275us	22730 us	212,43%
7114us	23781 us	234,28%
7367us	26033 us	253,37%
7281us	23440 us	221,93%
7144us	23149 us	224,03%
7331us	23275 us	217,48%
7034us	23160 us	229,25%
MÉDIA 23740 us	23325 us	223,55%
DESVIO PADRÃO222,34	963,33	13,52

Em média o método Armadillo é mais rápido em ambos os casos, no primeiro caso, de 500 amostras, ele é 112,29% mais rápido ou 2,12 vezes. No segundo caso, de 480.00 amostras, é 223,55% mais rápido ou 3,23 vezes.

O desvio padrão de 84,65 da diferença para 500 amostras é devido a oscilação de tempo de execução que ocorreu com o método da Armadillo .Já para uma amostra de 480.000 houve um desvio padrão de 13,52 que deve-se a constância de tempo das amostras. Resultados demonstrados pelas figuras 24 e 25.

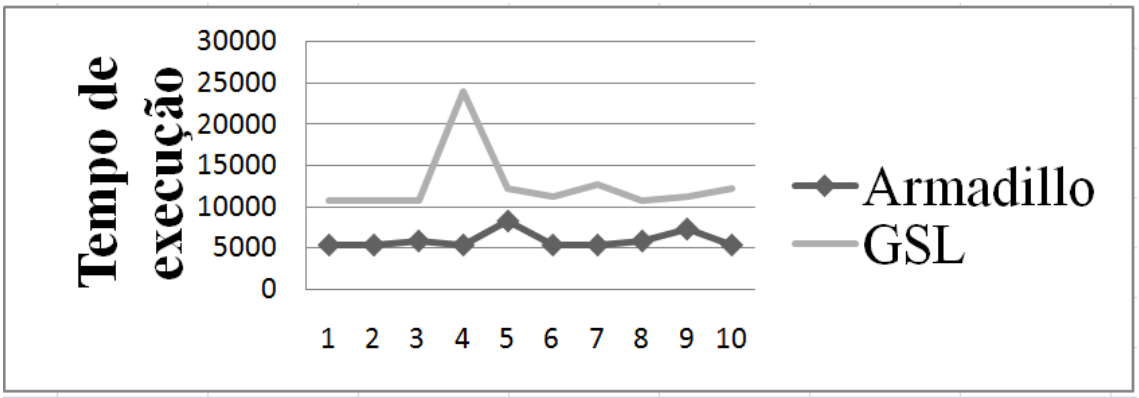

Figura 24: Resultado do teste do cálculo do máximo e mínimo para 500 amostras de entrada.

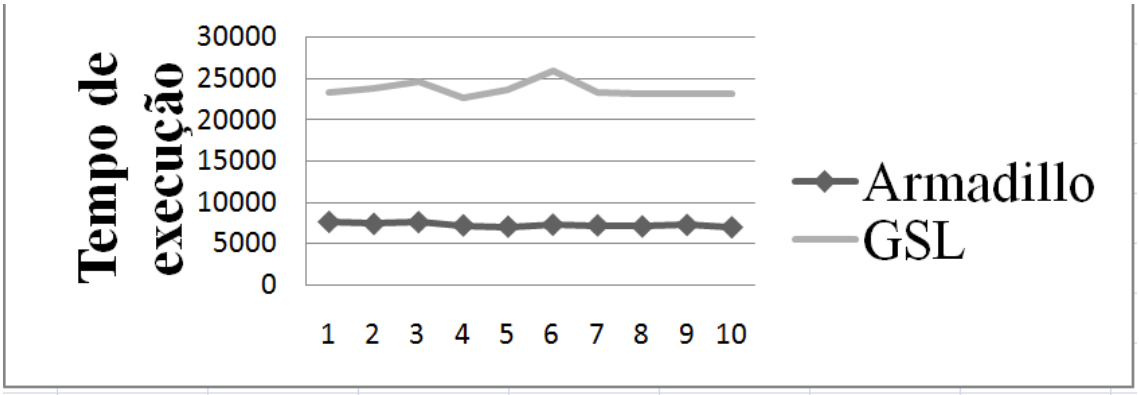

Figura 25: Resultado do teste do cálculo do máximo e mínimo para 480.000 amostras de entrada.

As figuras 24 e 25 demonstram que a biblioteca Armadillo apresenta tempos inferiores a biblioteca GSL em todos os testes realizados para o cálculo da dos valores máximo e mínimo de um vetor, dentro do ambiente do software de identificação de sistemas, e evidencia que ela sempre se sai melhor que a GSL.

5.3.1.3) Análise sequencial e paralela

Testou-se o algoritmo de escalonamento como um todo em suas formas sequencial e paralela. Para 500 e 480.000 amostras de entrada. Os resultados de cada método podem ser vistos nas Tabelas 17 e 18.

Tabela 17: Resultado do teste de cálculo do escalonamento para 500 amostras de entrada.

PARALELO	SEQUENCIAL	DIFERENÇA %
253 us	231 us	-8,69%
290 us	184 us	-36,55%
216 us	231 us	6,94%
189 us	195 us	3,17%
168 us	188 us	11,90%
198 us	309 us	56,06%
162 us	237 us	46,29%
173 us	216 us	24,85%
149 us	211 us	41,61%
177 us	210 us	18,64%
MÉDIAS 197,5 us	221,2 us	12%
DESVIO PADRÃO 44,16	35,86	27,68

Tabela 18: Resultado do teste de cálculo do escalonamento para 480.000 amostras de entrada.

PARALELO	SEQUENCIAL	DIFERENÇA %
49 ms	194 ms	295,91%
53 ms	193 ms	264,15%
50 ms	193 ms	286%
50 ms	194 ms	288%
50 ms	194 ms	288%
52 ms	194 ms	273,07%
50 ms	193 ms	286%
50 ms	193 ms	286%

50 ms	194 ms	288%
52 ms	193 ms	271,15%
MÉDIAS 50,6 ms	193,5 ms	282,41%
DESVIO PADRÃO 1,26	0,52	9,78

Em média o método Paralelo foi mais rápido em ambos os casos, no primeiro caso, de 500 amostras, ele é 12% mais rápido ou 1.12 vezes. No segundo caso, de 480.00 amostras, é 282,41% mais rápido ou 3,82 vezes, as diferenças negativas dizem respeito aos casos em que o processamento sequencial é mais rápido que o paralelo.

O desvio padrão de 27,68 para 500 amostras, juntamente com casos onde o método sequencial levou menos tempo, torna o método paralelo desnecessário em casos de pequenas amostras. Já para uma amostra de 480.000 o desvio padrão de 9,78 deve-se à constância de tempo das amostras, e torna o método paralelo interessante em casos de grandes amostras.Resultados demonstrados pelas figuras 26 e 27.

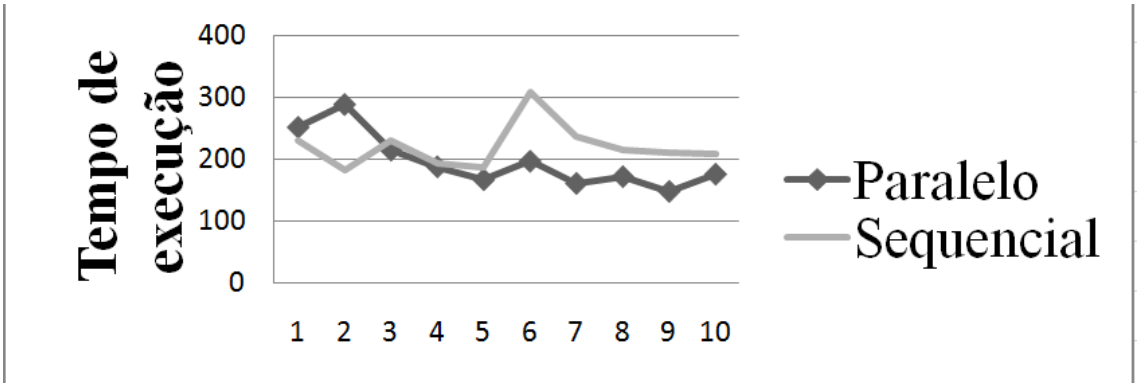

Figura 26: Resultado do teste de cálculo do escalonamento para 500 amostras de entrada.

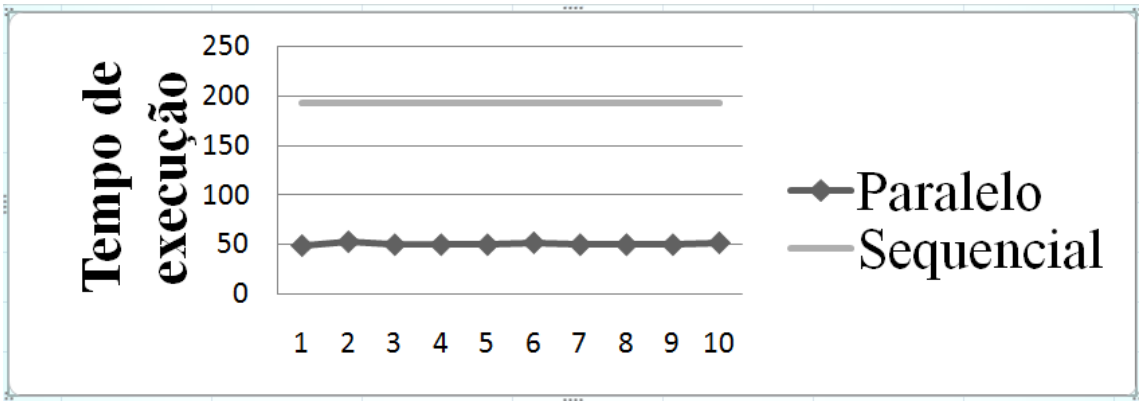

Figura 27: Resultado do teste de cálculo do escalonamento para 480.000 amostras de entrada.

As figuras 26 e 27 demonstram que o método paralelo apresenta tempos inferiores ao método sequencial para amostras na ordem de 480.000 em todos os testes realizados, já para amostras na ordem de 500 o tempo de execução do método sequencial aproxima-se do tempo do método paralelo, porém em oito dos dez testes realizados o método paralelo apresentou tempos inferiores. Os testes foram realizados dentro do ambiente do software de identificação de sistemas, e evidenciam que o método paralelo é promissor.

5.4) Comparativos de métodos utilizados no algoritmo DE

Os testes realizados se baseiam no DE proposto no capítulo 3. Serão analisados os algoritmos principais de cada uma de suas etapas quanto à execução sequencial ou paralela dos mesmos.

5.4.1) Geração da população inicial

A geração da população inicial é testada de quatro formas, algoritmo inicial sequencial, algoritmo inicial paralelo, algoritmo otimizado sequencial e o algoritmo otimizado paralelo. Os resultados podem ser visualizados nas Tabelas 19, 20, 21 e 22.

Tabela 19: Resultado do teste de geração da população inicial de 10 equações.

PARALELO	SEQUENCIAL	DIFERENÇA %
1236 US	707 us	74,8%
1115 US	703 us	58,6%
1207 US	784 us	53,9%
1073 US	774 us	38,6%
1179 US	685 us	72,1%
1176 US	755 us	55,8%
1273 US	714 us	78,3%
1304 US	707 us	84,5%
1203 US	716 us	68,0%
1197 US	724 us	65,3%
MÉDIAS 1196,3 US	726,9us	64,6%
DESVIO PADRÃO68,10	32,79	13,55

Tabela 20: Resultado do teste de geração da população inicial de 60.000 equações.

PARALELO	SEQUENCIAL	DIFERENÇA %
5550MS	3453 ms	60,73%
5467 MS	3627 ms	50,7%
5475 MS	3697 ms	48%
5527MS	3671 ms	50,6%
5352 MS	3653 ms	47%
5364 MS	3730 ms	44%
5500 MS	3667 ms	50%
5492 MS	3545 ms	55%
5410 MS	3723 ms	45,3%
5524 MS	3748 ms	47%
MÉDIAS 5466,1MS	3651,4ms	59,7%
DESVIO PADRÃO 68,79	91,05	4,97

Tabela 21: Resultado do teste de geração da população inicial de 10 equações otimizada.

PARALELO	SEQUENCIAL	DIFERENÇA %
373US	389us	4,3%
353US	351us	-0,5%
315US	333us	5,7%
314US	348us	10,8%
295US	341us	15,6%
345US	359us	4%
345US	351us	1,7%
370US	360us	-2,7%

309US	343us	11%
300US	352us	17,3%
MÉDIAS 352,7 US	331,9 us	6,3%
DESVIO PADRÃO 15,12	28,74	6,72

Tabela 22: Resultado do teste de geração da população inicial de 60.000 equações otimizada.

PARALELO	SEQUENCIAL	DIFERENÇA %
733 MS	2003 ms	173,3%
746 MS	1893 ms	153,8%
717 MS	1927 ms	168,8%
739 MS	1926 ms	160,6%
737 MS	1915 ms	159,8%
729 MS	1960 ms	168,9%
704 MS	1949 ms	176,8%
725 MS	1933 ms	166,6%
750 MS	1899 ms	153,2%
727 MS	1897 ms	160,9%
MÉDIAS 730,7MS	1930,2ms	164,2%
DESVIO PADRÃO 33,77	13,62	7,90

Antes de se otimizar o algoritmo o sequencial apresenta tempos inferiores aos do algoritmo paralelo, isto se deve ao uso de semáforos e estruturas de controle de acesso a dados compartilhados. Com a otimização esses acessos já não precisavam mais de tanto controle o que permitiu ao algoritmo paralelo executar em tempos inferiores. Após as otimizações tanto o sequencial como o paralelo apresentam tempos inferiores. As

diferenças negativas dizem respeito aos casos em que o processamento paralelo é mais rápido que o sequencial.

Nota-se que após a otimização para uma população inicial de 10 indivíduos o algoritmo paralelo é cerca de 6,3% mais rápido ou 1,06 vezes mais rápido, para uma população de 60.000a média obtida é de 164,2% mais rápido ou 2,64 vezes. Optou-se então por utilizar o algoritmo paralelo otimizado no aplicativo final.

O desvio padrão para 10 amostras é de 6,72, o que torna o método paralelo desnecessário em casos de pequenas amostras. Já para uma amostra de 60.000 o desvio padrão é de 7,9 e deve-se a constância de tempo das amostras.resultados demonstrados nas figuras 28 à 31.

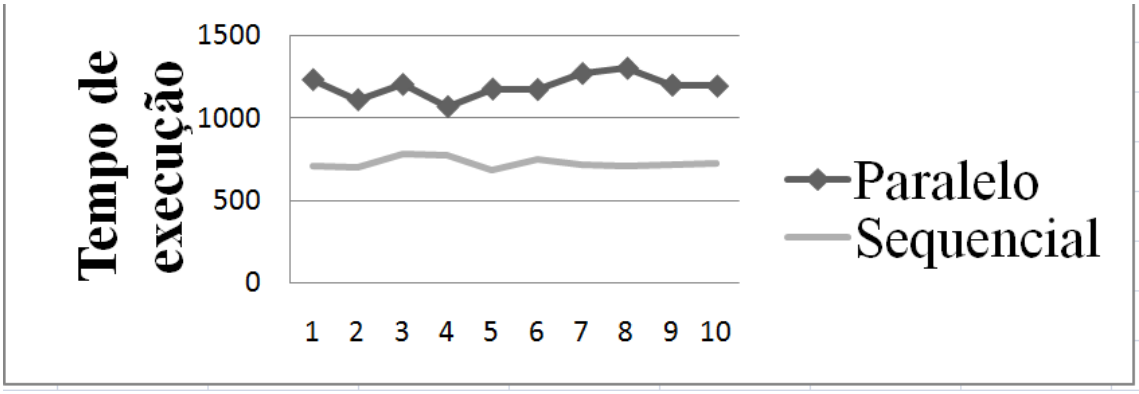

Figura 28: Resultado do teste de geração da população inicial de 10 equações.

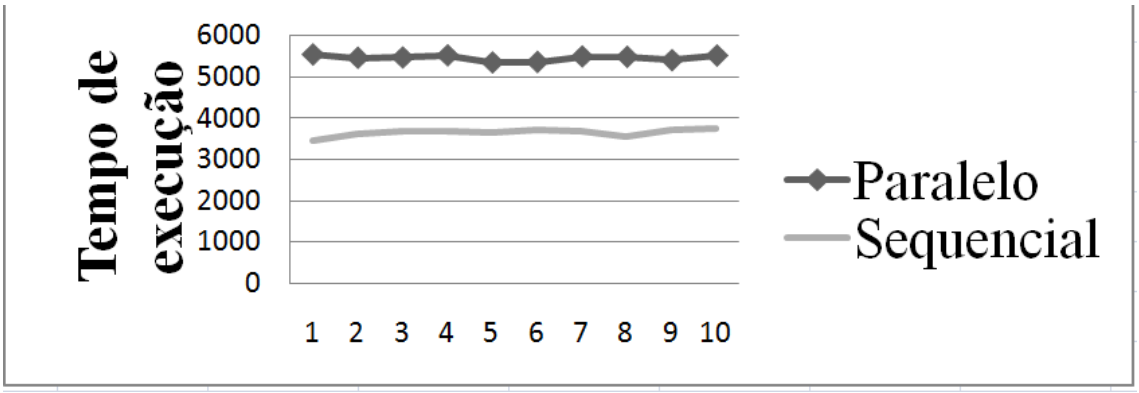

Figura 29: Resultado do teste de geração da população inicial de 60.000 equações.

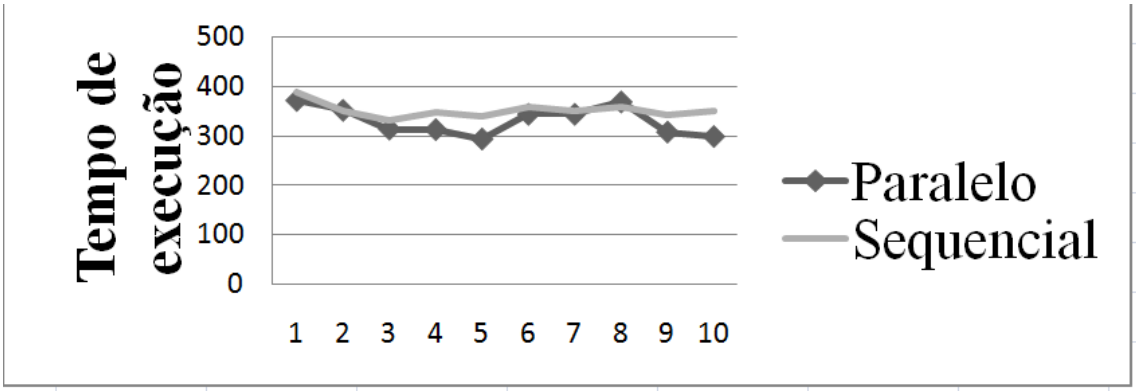

Figura 30: Resultado do teste de geração da população inicial de 10 equações otimizada.

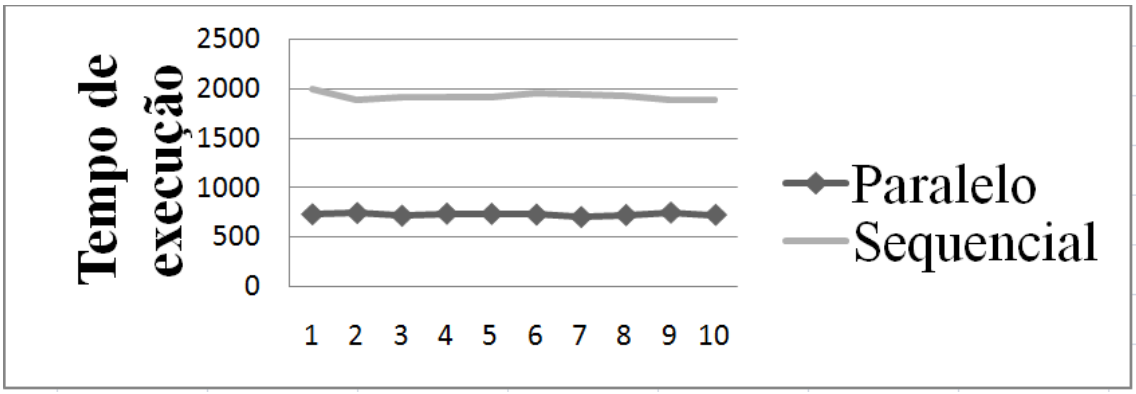

Figura 31: Resultado do teste de geração da população inicial de 60.000 equações otimizada.

As figuras 28 à 31 demonstram que no algoritmo padrão, por apresentar estruturas de controle multinúcleo que demandam tempo de execução. Os tempos de execução do algoritmo sequencial são inferiores ao tempos de execução do algoritmo paralelo em todos os testes realizados, porém no algoritmo otimizado, que possui alterações a fim de se evitar o uso de tais estruturas de controle, os tempos de execução do algoritmo paralelo mostraram-se inferiores aos tempos de execução do algoritmo sequencial. O que evidencia a importância de um código bem estruturado para o paralelismo no desempenho do algoritmo.

5.4.2) Mutação

Nesta etapa o algoritmo de mutação é analisado em sua forma paralela e sequencial. Utilizou-se uma amostra de 500 elementos e testou-se os algoritmos de seleção por roleta e de mutação propriamente dito.

5.4.2.1) Roleta

Para selecionar quais equações seriam usadas em cada processo de mutação utilizou-se o método da roleta. Testou-se dois algoritmos, um padrão retirado de (LINDEN, 2008) e outro otimizado. O padrão segue os seguintes passos:

1) Somar todas as aptidões;

2) Gerar um número aleatório X entre 0 e a soma das aptidões;

3) Checar cada indivíduo, subtrair sua aptidão de X, continuar até essa subtração ser menor que zero, o indivíduo da iteração menor que zero é o escolhido;

4) Repetir o passo 2 e 3 até preencher toda a população.

Esse algoritmo tem uma complexidade de O (n*log (m) + m), onde m é o número de indivíduos da população a ser escolhida, e n o número de indivíduos a escolher.

O algoritmo otimizado pode ser descrito em linguagem de programação conforme figura 32.

```
Doublerandom=Math.random(); //Retorna um valor aleatório entre 0 e 1.
Doublesum=0;
for(inti=0;i<items.length;i++)
{
val=items[i];
sum+=val.getValue();
if(sum>random)
{
selected=val;
break;
}
}
```

Figura 32: Algoritmo de roleta otimizado.

O algoritmo otimizado possui uma complexidade de O (n+m), que é linear, além de não exceder a memória. Os resultados dos testes com ambos os algoritmos podem ser visualizados na Tabela 23 e na Tabela 24.

Tabela 23: Resultado do teste de seleção por roleta para tamanho de população 100 equações.

ALGORITMO OTIMIZADO	ALGORITMO PADRÃO	DIFERENÇA %
179 us	265 us	48,04%
173 us	257 us	48,55%
178 us	221 us	24,15%
174 us	240 us	37,93%
181 us	225 us	24,30%
187 us	248 us	32,62%
173 us	227 us	31,21%
184 us	220 us	19,56%
208 us	235 us	12,98%
186 us	237 us	27,41%
MÉDIAS 182,3 us	237,5 us	30,27%
DESVIO PADRÃO 10,39	15,29	11,58

Tabela 24: Resultado do teste de seleção por roleta para tamanho de população 300 equações.

ALGORITMO OTIMIZADO	ALGORITMO PADRÃO	DIFERENÇA %
996 us	1461 us	46,68%
924 us	1487 us	60,93%
964 us	1409 us	46,16%
978 us	1455 us	48,77%
950 us	1442 us	51,78%
952 us	1439 us	51,15%
989 us	1478 us	49,44%
968 us	1469 us	51,75%
1025 us	1446 us	41,07%
999 us	1427 us	42,84%
MÉDIAS 974,5 us	1451,3 us	48,92%
DESVIO PADRÃO 29,14	23,69	5,54

O algoritmo otimizado da roleta mostrou-se em média 30,27% mais rápido ou 1,3 vezes para 100 amostras e em média 48,93% mais rápido ou 1,49 vezes para 300 amostras.

O desvio padrão é de 11,58 da diferença para uma população de 100 equações, e para uma população de 300 o desvio padrão é de 5,54. O que tornou o método otimizado interessante em ambos os casos.Resultados demonstrados pelas figuras 33 e 34.

O método otimizado da roleta possui melhores tempos de execução, levando em consideração o desvio padrão e a média,optou-se por utilizar esse método no aplicativo final.

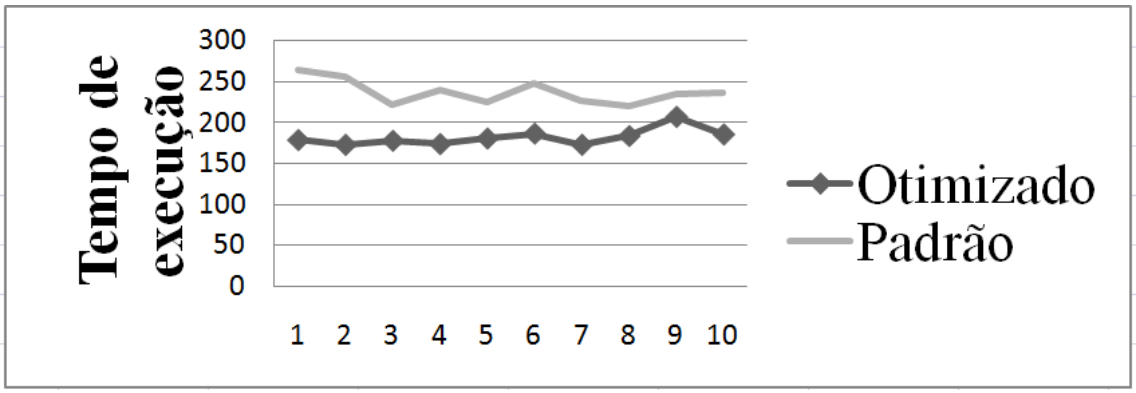

Figura 33: Resultado do teste de seleção por roleta para tamanho de população 100 equações.

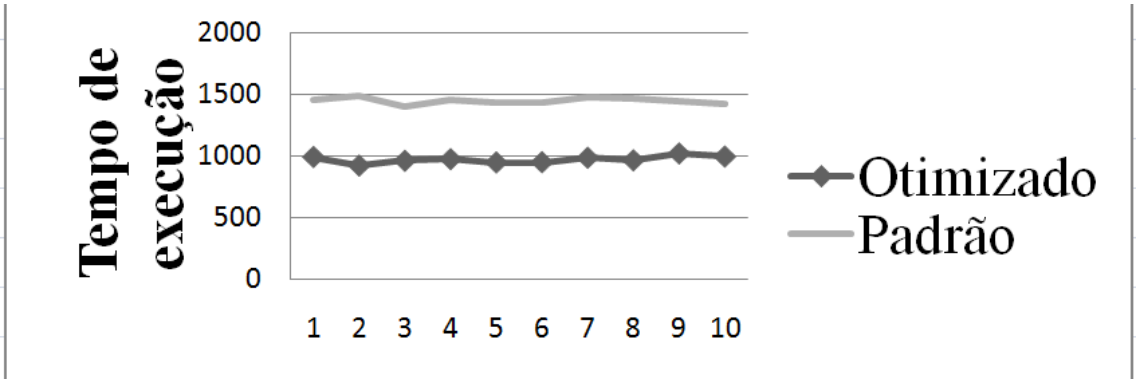

Figura 34: Resultado do teste de seleção por roleta para tamanho de população 300 equações.

As figuras 33 e 34 demonstram que no algoritmo padrão, por apresentar uma complexidade exponencial de O (n*log (m) + m), os tempos de execução do algoritmo são superiores ao tempos de execução do algoritmo otimizado em todos os testes realizados, já no algoritmo otimizado, que possui complexidade linear de O (n+m), os tempos de execução mostraram-se inferiores aos tempos de execução do algoritmo padrão. O que evidencia a importância da análise de software e da escolha de algoritmos de complexidade menor.

5.4.2.2) Algoritmo de mutação

Testou-se o algoritmo de mutação como um todo em suas formas sequencial e paralela. Para uma população de 100 e 1.000 equações amostras. Os resultados de cada método podem ser vistos nas Tabelas 25 e 26.

Tabela 25: Resultado do teste do algoritmo de mutação para uma população de 100 equações.

SEQUENCIAL	PARALELO	DIFERENÇA %
5 us	8 us	60%
5 us	8 us	60%
7 us	10 us	42,85%
7 us	10 us	42,85%
8 us	11 us	37,5%
10 us	11 us	10%
6 us	9 us	50%
5 us	9 us	80%
4 us	8 us	100%
3 us	6 us	100%
MÉDIAS 6 us	9 us	50%
DESVIO PADRÃO 2,05	1,56	28,41

Tabela 26: Resultado do teste do algoritmo de mutação para uma população de 1.000 equações.

SEQUENCIAL	PARALELO	DIFERENÇA %
39 ms	82 ms	110,25%
75 ms	126 ms	68%
74 ms	118 ms	59,45%
82 ms	125 ms	52,43%
98 ms	143 ms	45,91%
83 ms	123 ms	48,19%
77 ms	114 ms	48,05%
74 ms	113 ms	52,70%
68 ms	107 ms	57,35%
74 ms	108 ms	45,94%
MÉDIAS 74,4 ms	115,9 ms	55,77%
DESVIO PADRÃO 14,87	15,90	19,35

Em média o método sequencial é mais rápido em ambos os casos, no primeiro caso, de população com 100 equações, ele é 50% mais rápido ou 1,5 vezes. No segundo caso, de 1.000 equações, é 55,77% mais rápido ou 1,55 vezes.

O desvio padrão é de 28,41 para 100 equações, já para uma população de 1.000 o desvio padrão é de 19,35, o que, juntamente com os resultados da média, tornou o método sequencial promissor em ambos os casos.Resultados demonstrados pelas figuras 35 e 36.

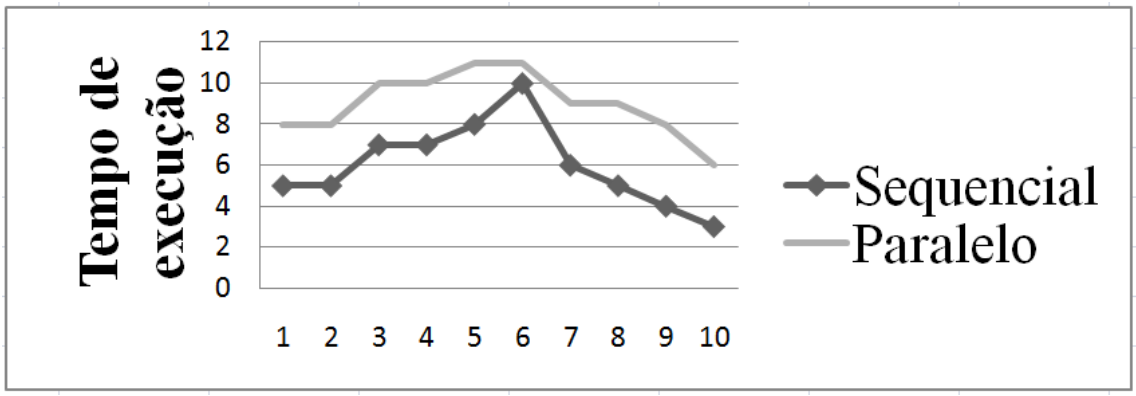

Figura 35: Resultado do teste do algoritmo de mutação para uma população de 100 equações.

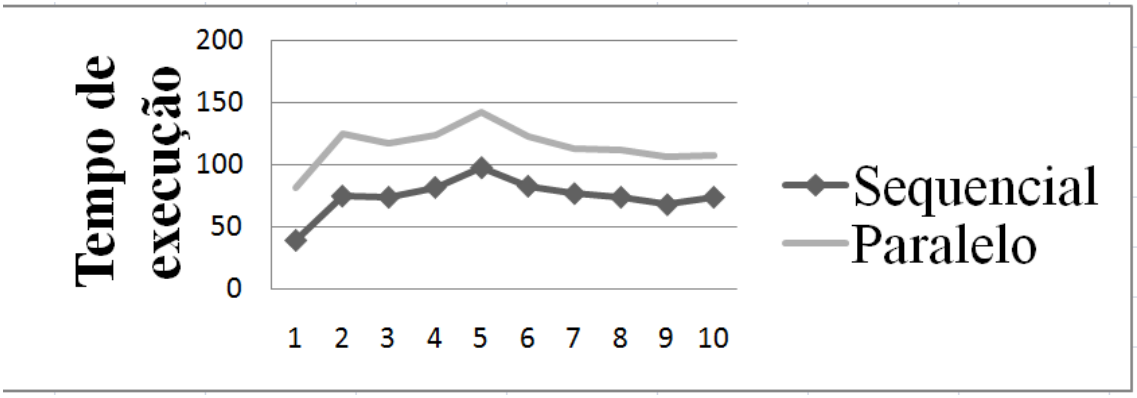

Figura 36: Resultado do teste do algoritmo de mutação para uma população de 1.000 equações.

As figuras 35 e 36 demonstram que no algoritmo sequencial, por apresentar um algoritmo simples, os tempos de execução do algoritmo sequencial são inferiores aos tempos de execução do algoritmo paralelo em todos os testes realizados, isto se deve a simplicidade do algoritmo e ao algoritmo paralelo demandar tempo de execução para gerenciar suas *threads*. O que evidencia que alguns algoritmos apresentam tempos de execução inferiores em suas formas sequenciais.

5.4.3) Cruzamento

Nesta etapa o algoritmo de cruzamento analisou-se a forma paralela e sequencial do mesmo. Utilizou-se uma amostra de 500 elementos e uma população de 100 e de 1.000 equações. Os resultados de cada método podem ser vistos nas Tabelas 27 e 28.

Tabela 27: Resultado do teste do algoritmo de cruzamento para uma população de 100 equações.

SEQUENCIAL	PARALELO	DIFERENÇA %
1239 us	18963 us	1430,50%
359 us	3204 us	792,47%
209 us	2680 us	1182,29%
602 us	11415 us	1796,17%
239 us	2470 us	933,47%
306 us	2892 us	845,09%
236 us	3153 us	1236,01%
212 us	2508 us	1083,01%
265 us	3096 us	1068,30%
262 us	2612 us	896,94%
MÉDIAS 392,9 us	5299,3 us	1248,76
DESVIO PADRÃO 319,13	5513,85	305,86

Tabela 28: Resultado do teste do algoritmo de cruzamento para uma população de 1.000 equações.

SEQUENCIAL	PARALELO	DIFERENÇA %
1907 us	24818 us	1201,41%
3393 us	78384 us	2210,16%
2514 us	28804 us	1045,74%
6130 us	133433 us	2076,72%
2809 us	25019 us	790,67%
3012 us	26700 us	786,45%
2753 us	29344 us	965,89%
3234 us	25226 us	680,02%
3010 us	28100 us	833,55%
2317 us	25187 us	987,05%
MÉDIAS 3107,9 us	42501,5 us	1267,53%
DESVIO PADRÃO 1149,14	35886,39	541,44

Em média o método sequencial é mais rápido em ambos os casos. No primeiro caso, de população com 100 equações, ele é 1248,76% mais rápido ou 13,48 vezes. No segundo caso, de 1.000 equações, é 1267,53% mais rápido ou 13,67 vezes.

O desvio padrão é de 305,86 para 100 equações e para uma população de 1.000 o desvio padrão é de 541,44, o que, considerando a média, tornou o método sequencial promissor em ambos os casos. Resultados são demonstrados nas figuras 37 e 38.

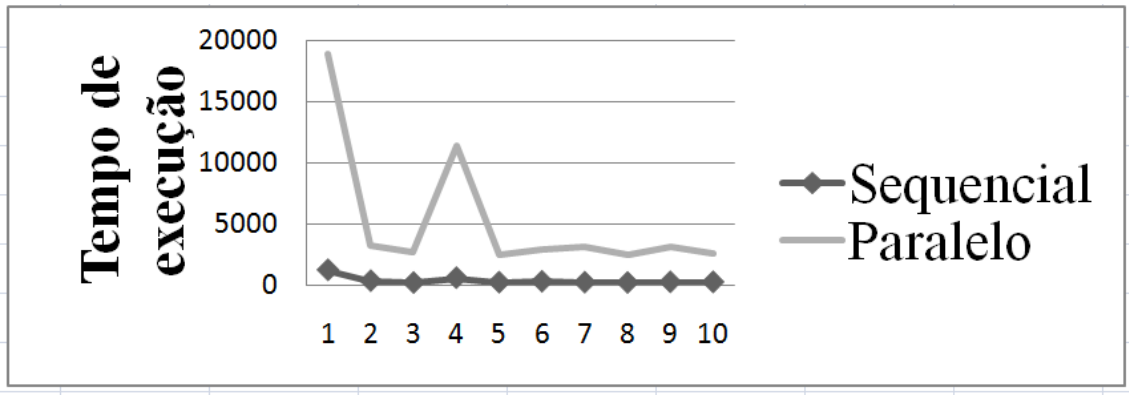

Figura 37: Resultado do teste do algoritmo de cruzamento para uma população de 100 equações.

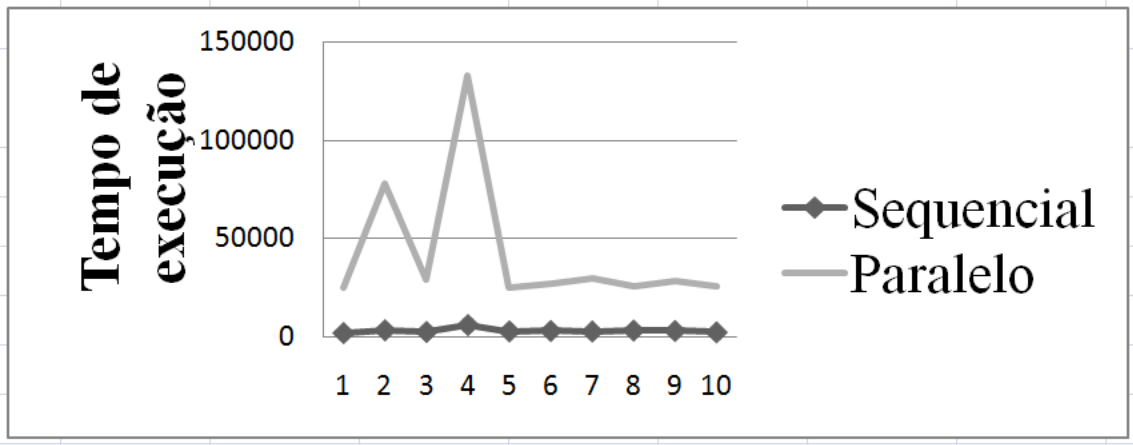

Figura 38: Resultado do teste do algoritmo de cruzamento para uma população de 1.000 equações.

As figuras 37 e 38 demonstram que no algoritmo sequencial, por apresentar um algoritmo simples de troca (*swap*), os tempos de execução do algoritmo sequencial são inferiores ao tempos de execução do algoritmo paralelo em todos os testes realizados, isto se deve a simplicidade do algoritmo e ao algoritmo paralelo demandar tempo de execução para gerenciar suas *threads*. O que evidencia que alguns algoritmos apresentam tempos de execução inferiores em suas formas sequenciais.

5.4.4) Cálculo do ERR, dos coeficientes, do BIC e da aptidão

Nesta etapa o algoritmo de cálculo do ERR, coeficientes, BIC e aptidão foram analisados em sua forma paralela e sequencial. Utilizou-se uma amostra de 500 elementos. E uma população de 100 e 1.000 equações.

Por questão de arquitetura de software juntou-se os cálculos do ERR, coeficientes, BIC e aptidão em uma única função e esta função é analisada em sua forma sequencial e paralela. Os resultados de cada forma podem ser vistos nas Tabelas 29 e 30.

Tabela 29: Resultado do teste do algoritmo ERR, coeficientes, BIC e aptidão para uma população de 100 equações.

SEQUENCIAL	PARALELO	DIFERENÇA %
285 ms	121 ms	135,53%
148 ms	99 ms	49,49%
247 ms	125 ms	97,6%
255 ms	121 ms	110,74%
347 ms	146 ms	137,67%
390 ms	188 ms	107,44%
389 ms	162 ms	140,12%
394 ms	177 ms	122,59%
265 ms	150 ms	76,66%
294 ms	152 ms	93,42%
MÉDIAS 301,4 ms	144,1 ms	109,16%
DESVIO PADRÃO 79,21	27,68	29,07

Tabela 30: Resultado do teste do algoritmo ERR, coeficientes, BIC e aptidão para uma população de 1.000 equações.

SEQUENCIAL	PARALELO	DIFERENÇA %
3667 ms	1107 ms	231,25%
2728 ms	1018 ms	167,97%
2214 ms	1047 ms	111,46%
3491 ms	1182 ms	195,34%
2698 ms	1307 ms	106,42%
3137 ms	1426 ms	119,98%
3286 ms	1378 ms	138,46%
3535 ms	1529 ms	131,19%
3539 ms	1509 ms	134,52%
3233 ms	1475 ms	119,18%
MÉDIAS 3152,8 ms	1297,8 ms	142,93%
DESVIO PADRÃO 467,50	195,33	40,47

Em média o método Paralelo é mais rápido em ambos os casos. No primeiro caso, de 100 equações, ele é 109,16% mais rápido ou 2,09 vezes e no segundo caso, de 1.000 equações, é 142,93% mais rápido ou 2,42 vezes.

O desvio padrão é de 29,07 para 100 equações e para uma população de 1.000 equações o desvio padrão é de 40,47, considerando a média e o desvio padrão o método paralelo mostrou-se promissor em ambos os casos. Resultado demonstrado pelas figuras 39 e 40.

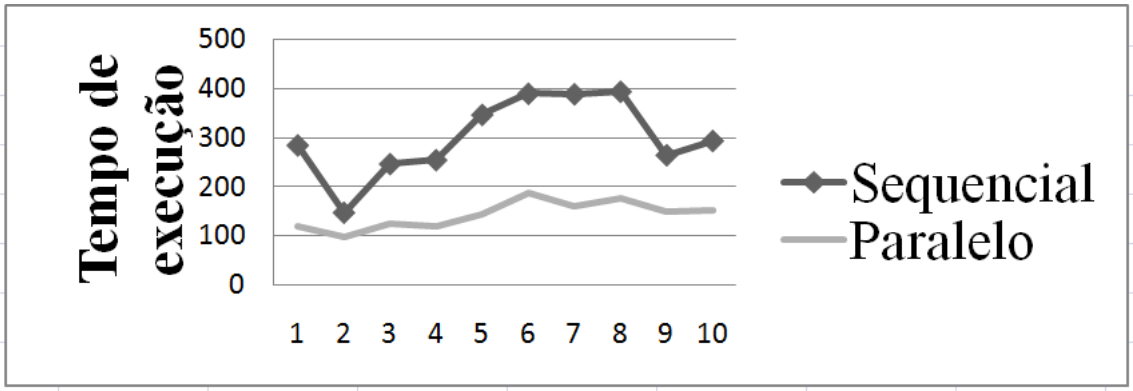

Figura 39: Resultado do teste do algoritmo ERR, coeficientes, BIC e aptidão para uma população de 100 equações.

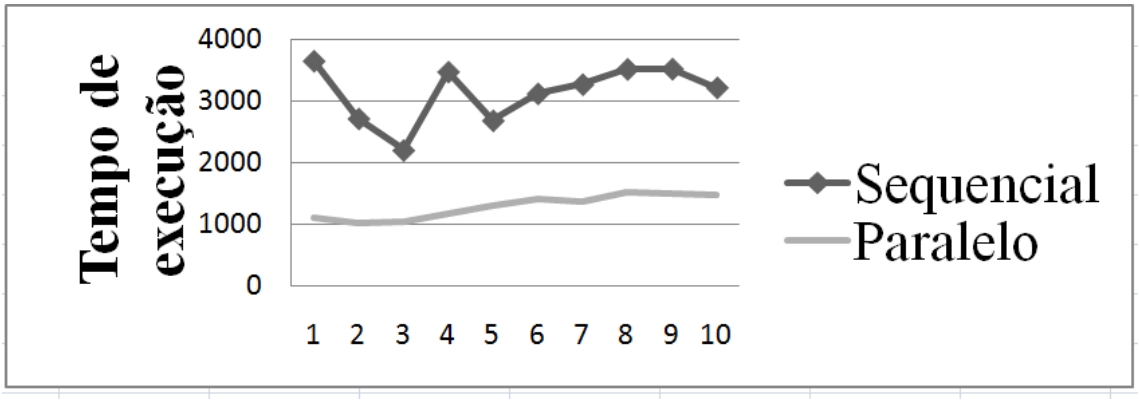

Figura 40: Resultado do teste do algoritmo ERR, coeficientes, BIC e aptidão para uma população de 1.000 equações.

As figuras 39 e 40 demonstram que no algoritmo paralelo, por apresentar um algoritmo longo e complexo, os tempos de execução do mesmo são inferiores aos tempos de execução do algoritmo sequencial em todos os testes realizados, isto se deve a complexidade do algoritmo. O que evidencia que alguns algoritmos apresentam tempos de execução inferiores em suas formas paralelas.

5.5) Considerações Finais

Neste capítulo são apresentados os resultados obtidos em vários testes realizados para a construção do aplicativo final com uso da metodologia proposta pelo trabalho.

Os resultados obtidos demonstraram que nem sempre uma abordagem paralela é mais rápida que uma sequencial. Também pode ser visualizado que um algoritmo mal estruturado pode resultar em perdas no paralelismo do mesmo, e que algumas técnicas, muitas vezes simples, podem fazer grandes diferenças quando se trata de otimizar um algoritmo.

Ressalta-se também que muitos problemas e dificuldades foram encontradas no desenvolvimento e teste de cada algoritmo descrito nesse capítulo, e que muitos métodos e técnicas promissoras foram estudadas, porém optou-se por mantê-las como trabalhos futuros a fim de não escapar do escopo deste trabalho.

Capítulo 6:
Validação do Modelo

6.1) Considerações iniciais

O objetivo deste capítulo é apresentar a validação do algoritmo de seleção das estruturas, ou seja, validar se o aplicativo é capaz de identificar um sistema pré-determinado.

Para isso, utilizou-se de modelos artificiais com características especificas, geradoras de problemas para a seleção da estrutura, a fim de avaliar a capacidade do aplicativo em superar dificuldades.

Serão apresentadas as técnicas de validação de modelos e seus resultados.

Devido a recursividade do processo de identificação por algoritmo DE, a definição de um critério de parada é necessário. Neste projeto decidiu-se por executar o DE até que não haja mais nenhuma melhora na aptidão da melhor equação em mil ciclos consecutivos, ou seja, se ele não encontrar uma equação melhor em mil ciclos consecutivos, o DE é finalizado.

6.2) Validação com modelagem de um sistema não linear racional SISO

O modelo avaliado nesta seção foi extraído de (CORRÊA, 1997) e consiste em um modelo artificial com características específicas que gerem problemas para a seleção da estrutura assim auxilie na avaliação da capacidade do algoritmo em superar estas dificuldades, os testes de geração de sinais de entrada e de ruído são os mesmos utilizados por (MORAIS, 2013). O modelo pode ser visualizado na equação (5.1).

$$y(k) = \frac{0.2y(k-1) + 0.1y(k-1)u(k-1) + u(k-1)}{1 + y(k-1)^2 + y(k-2)^2} + e(k) \qquad (5.1)$$

Onde: u(k) é uniformemente distribuído em [-1,1], e(k) é um ruído branco com média zero, variância $8{,}34x10^{-6}$, amplitude variando de $\pm 5{,}0x10^{-3}$ com uma relação sinal/ruído (SNR) de 73,38 dB.

Através de dados de entrada gerados pelo modelo descrito na equação (5.1), testou-se o programa, para avaliar se o mesmo é capaz de encontrar tal modelo.

O Algoritmo é capaz de encontrar o modelo da equação (5.1) o que levou 110 gerações e cerca de 9 segundos. A comparação gráfica entre o sistema encontrado e o original pode ser visualizada na figura 41.

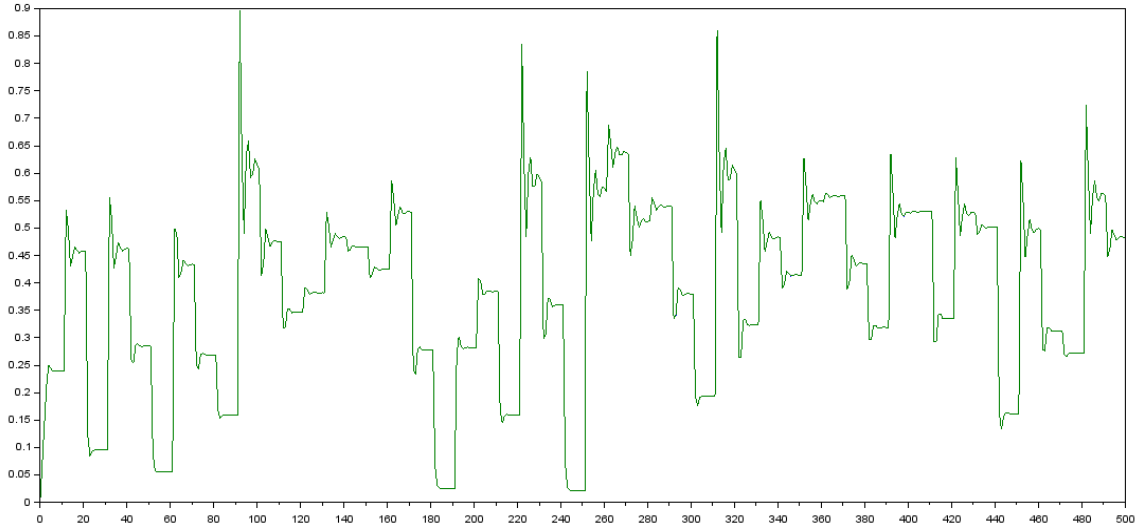

Figura 41: Saída esperada (Verde) e saída calculada (Azul) para o modelo (5.1).

O gráfico do modelo original encontra-se na cor verde e o do modelo encontrado na cor azul, porém como o algoritmo encontrou exatamente a mesma equação os dois estão sobrepostos. Ressalta-se que para encontrar o sistema foram utilizadas apenas as primeiras 250 amostras, e que da amostra 251 à amostra 500 são dados utilizados para validação.

Como o modelo encontrado é o mesmo, não há necessidade de se realizar a análise estatística. Isso se deve ao conhecimento prévio do sistema que está sendo modelado, caso não se tenha tal conhecimento ou caso o modelo seja diferente deve-se realizar a análise estatística.

Optou-se então por utilizar o método de escalonamento dos dados de entrada, a fim de analisar a capacidade do modelo em trabalhar com dados escalonados. Os dados utilizados para este teste foram os mesmos gerados pelo modelo (5.1), porém foram escalonados seguindo a equação (3.2) do capítulo 3.

O algoritmo é capaz de encontrar um modelo que representa o conjunto de dados escalonados, o modelo encontrado pode ser visualizado através da equação (5.2).

$$y(k) = \frac{0.017u(k-1)^{0.012} + 0.21y(k-1) + 1.13u(k-1)}{1 + 0.82y(k-2)^{2.03} + 0.75y(k-1)^{2} + 0.04y(k-1)^{-0.10}y(k-2)^{-0.01}u(k-1)^{-0.04}} \quad (5.2)$$

O Algoritmo é capaz de encontrar o modelo da equação (5.2) o que levou em 163 gerações e cerca de 27 segundos. A comparação gráfica entre o sistema encontrado e o original pode ser visualizada na figura 42.

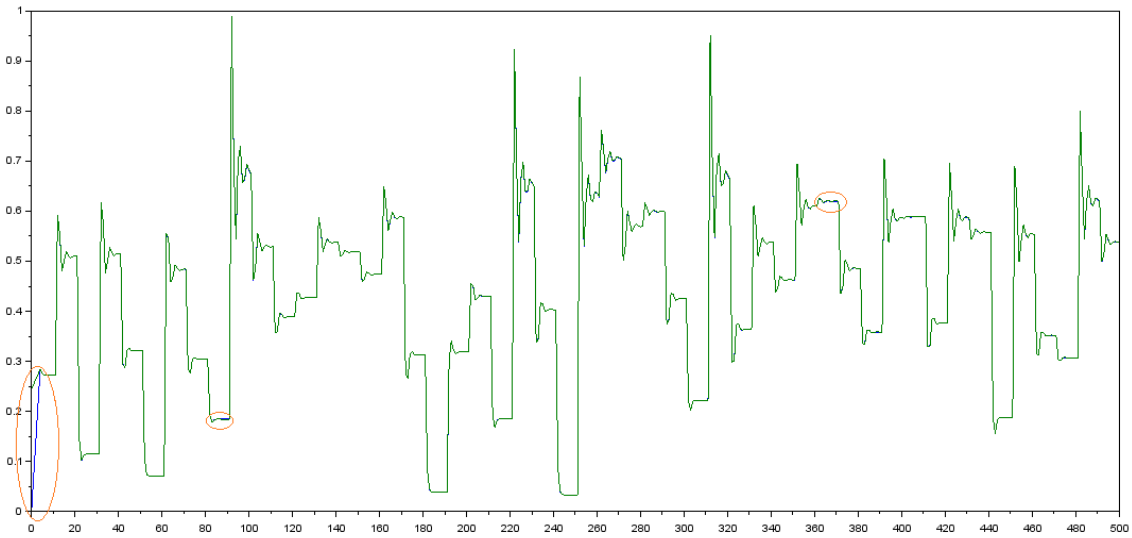

Figura 42: Saída esperada (Verde) e saída calculada (Azul) para o modelo (5.2).

O gráfico do modelo original encontra-se na cor verde e o do modelo encontrado na cor azul, porém como o algoritmo encontrou um modelo com erro quadrático de1.29e-7, os dois aparentam estar sobrepostos, pode-se notar a diferença no começo do gráfico e nos pontos assinalados por elipses em vermelho. Ressalta-se que para encontrar o sistema foram utilizadas apenas as primeiras 250 amostras, e que da amostra 251 à amostra 500 são dados utilizados para validação.

Como o modelo encontrado não possui conhecimento prévio, deve-se utilizar análise estatística a fim de validar o mesmo. A análise foi realizada e encontra-se nas figuras 43 à 49.

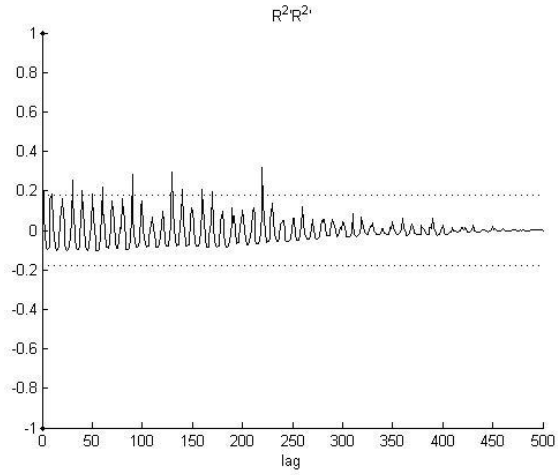

Figura 43: Análise estatística do modelo calculado (5.2) R^2R^2.

Alguns pontos da figura 43 saem do intervalo de aceitação, o que demonstra que provavelmente ainda há informação relevante não linear nos resíduos.

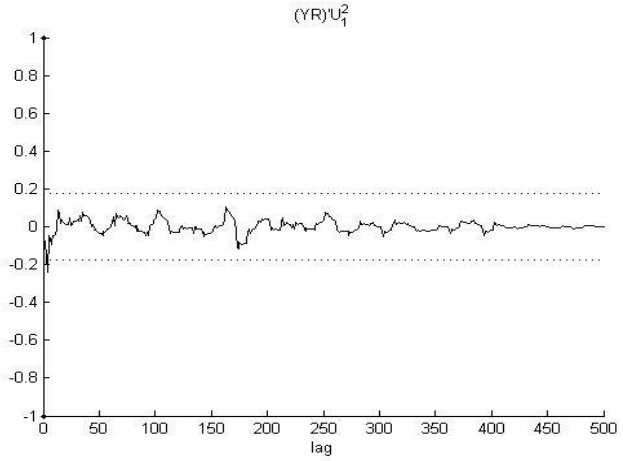

Figura 44: Análise estatística do modelo calculado (5.2) (YR)U2

Todos os pontos da figura 44 então dentro do intervalo de aceitação, o que demonstra que provavelmente não há informação relevante não linear da entrada com a saída.

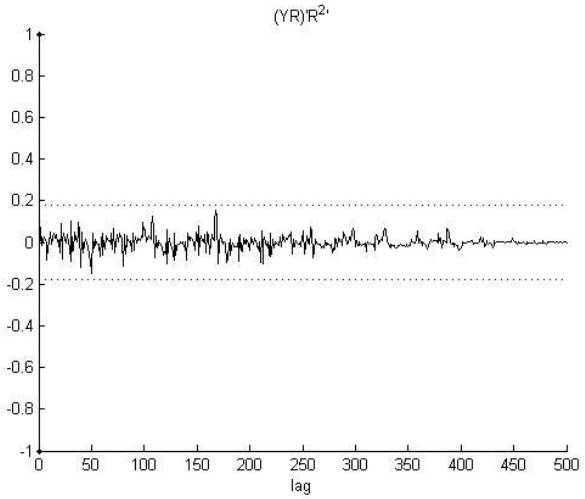

Figura 45: Análise estatística do modelo calculado (5.2) (YR)R^2.

Todos os pontos da figura 45 então dentro do intervalo de aceitação, o que demonstra que provavelmente não há informação relevante não linear dos resíduos com a sida.

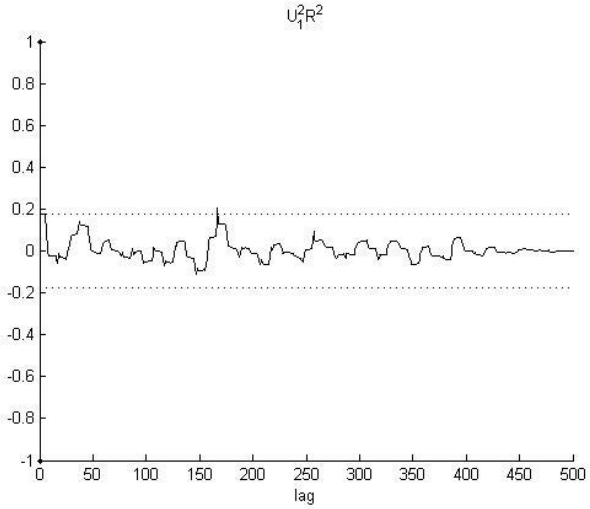

Figura 46: Análise estatística do modelo calculado (5.2) U^2 R^2.

Alguns pontos em torno da amostra 180 saem da especificação na figura 46o restante está dentro do intervalo de aceitação, o que demonstra que provavelmente não há informação relevante não linear suficiente contida na saída.

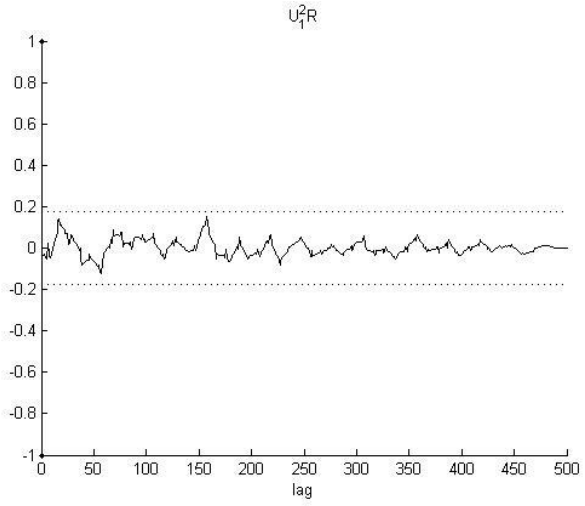

Figura 47: Análise estatística do modelo calculado (5.2) U^2R.

Novamente todos os pontos da figura 47 então dentro do intervalo de aceitação, o que demonstra que provavelmente não há informação não linear contida na entrada.

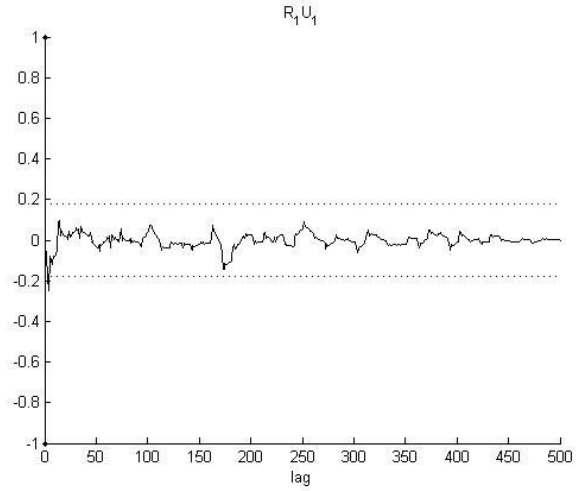

Figura 48: Análise estatística do modelo calculado (5.2) RU.

Os pontos da figura 48 então dentro do intervalo de aceitação, o que demonstra que provavelmente não há informação contida na entrada.

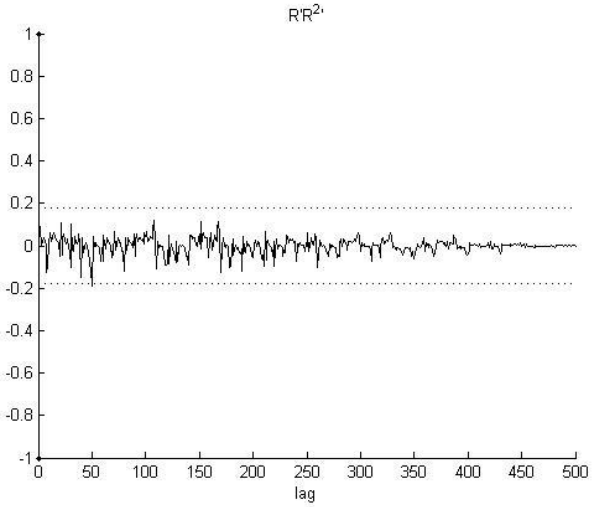

Figura 49: Análise estatística do modelo calculado (5.2) RR^2.

Todos os pontos da figura 49 então dentro do intervalo de aceitação, mostrando que provavelmente não há informação contida nos resíduos para esta análise.

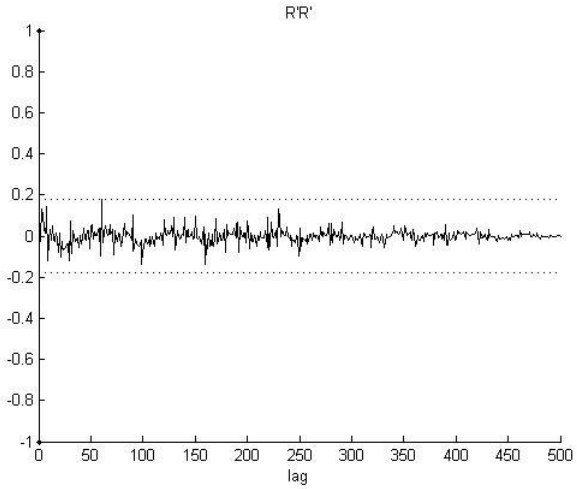

Figura 50: Análise estatística do modelo calculado(5.2) RR.

Finalmente na figura 50 todos os pontos encontram-se dentro do intervalo de aceitação, o que demonstra que provavelmente não há informação contida nesta análise.

No geral o modelo apresentou resultados satisfatórios, porém como pode-se visualizar pelo gráfico que envolvem os resíduos R quadráticos, alguns pontos saem das especificações. O que indica que provavelmente ainda há informação quadrática contida nos resíduos, porém o algoritmo não conseguiu encontrar um modelo melhor para representar os dados.

6.3) Validação com modelagem de um sistema não linear racional MIMO

O modelo avaliado nesta seção foi extraído de (MORAIS, 2013) e os testes de geração de sinais de entrada e de ruído são os mesmos utilizados por ele, consiste em um modelo MIMO e visa avalizar o desempenho do algoritmo na identificação de sistemas com múltiplas entradas e múltiplas saídas. O modelo pode ser visualizado nas equações (5.3) e (5.4).

$$y_1(k) = \frac{u_1(k-1)^2 + u_2(k-3) + y_1(k-1) + y_2(k-2)}{1 + u_1(k-1) + y_1(k-1)^2} + e(k) \qquad (5.3)$$

$$y_2(k) = \frac{u_1(k-1)^2 + u_2(k-3) + y_2(k-1) + y_1(k-2)}{1 + u_1(k-1) + y_2(k-1)^2} + e(k) \qquad (5.4)$$

Onde: u(k) é uniformemente distribuído em [-1,1], e(k) é um ruído branco com média zero, variância $8{,}34\text{x}10^{-6}$, amplitude variando de $\pm 5{,}0\text{x}10^{-3}$ com uma relação sinal/ruído (SNR) de 73,38 dB.

Através de dados de entrada gerados pelo modelo descrito na equação (5.3) e (5.4), testou-se o programa, para verificar se o mesmo é capaz de encontrar tal modelo.

O Algoritmo é capaz de encontrar o modelo da equação (5.3) o que levou 796 gerações e cerca de 818 segundos, ou aproximadamente 14 minutos. O modelo da equação (5.4) foi encontrado em 464 gerações o que levou cerca de 377 segundos, ou aproximadamente 6 minutos e 17 segundos.

A comparação gráfica entre o sistema encontrado e o original pode ser visualizada nas figuras 51 e 52.

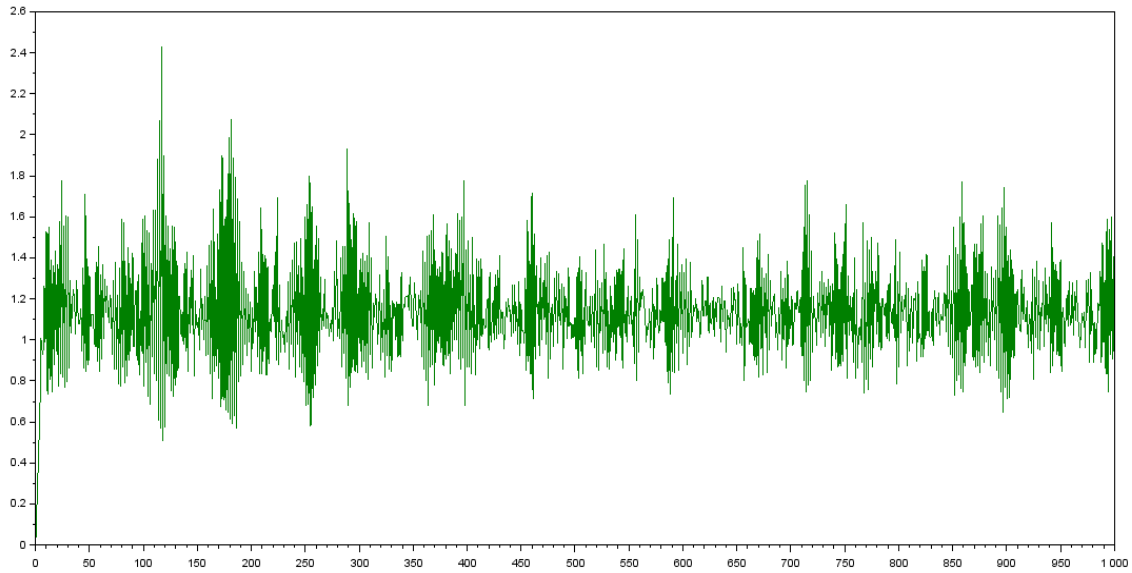

Figura 51: Saída esperada (Verde) e saída calculada (Azul) para o modelo (5.3).

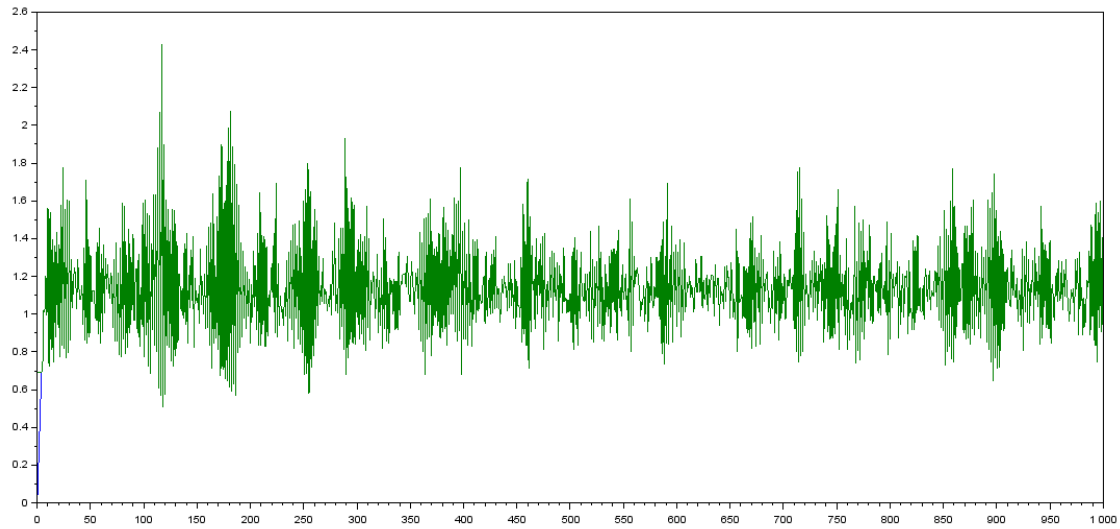

Figura 52: Saída esperada (Verde) e saída calculada (Azul) para o modelo (5.4).

Os gráficos dos modelos originais encontram-se na cor verde e os dos modelos encontrados na cor azul, porém como o algoritmo encontrou exatamente a mesma equação os dois estão sobrepostos. Ressalta-se que para encontrar o sistema foram utilizadas apenas as primeiras 500 amostras, e que da amostra 501 à amostra 1000 são dados utilizados para validação.

Como os modelos encontrados foram os mesmos não há necessidade de se realizar a análise estatística. Isso se deve ao conhecimento prévio do sistema que está

sendo modelado, caso não se tenha tal conhecimento ou caso o modelo seja diferente deve-se realizar a análise estatística.

Novamente optou-se por utilizar o método de escalonamento dos dados de entrada, a fim de analisar a capacidade do modelo de trabalhar com dados escalonados. Os dados utilizados para este teste foram os mesmos gerados pelos modelos (5.3) e (5.4).

O algoritmo é capaz de encontrar um modelo que representa o conjunto de dados escalonados, e o modelo pode ser visualizado através da equação (5.5) e (5.6).

$$x) = \frac{0.36u_1(k-1)^2 + 0.38u_2(k-3) + 0.73y_2(k-1) + 0.95y_1(k-2) + 0.02y_1(k-2)^5}{1 + 0.93u_1(k-1) + 6.35y_2(k-1)^2 - 0.96y_2(k-1)^2} \quad (5.5)$$

$$k) = \frac{0.02y_1(k-2)^5 + 0.4u_2(k-3) + 0.95y_1(k-2) + 0.73y_2(k-1) + 0.36u_1(k-1)^2}{1 + 0.9u_1(k-1) + 0.63y_2(k-1)^2 - 0.96y_2(k-1)^2} \quad (5.6)$$

O Algoritmo é capaz de encontrar os modelos da equação (5.5) o que levou 14 gerações e 7 segundos. Da equação (5.6) levou 87 gerações e 64 segundos. A comparação gráfica entre o sistema encontrado e o original pode ser visualizada nas figuras 53 e 54.

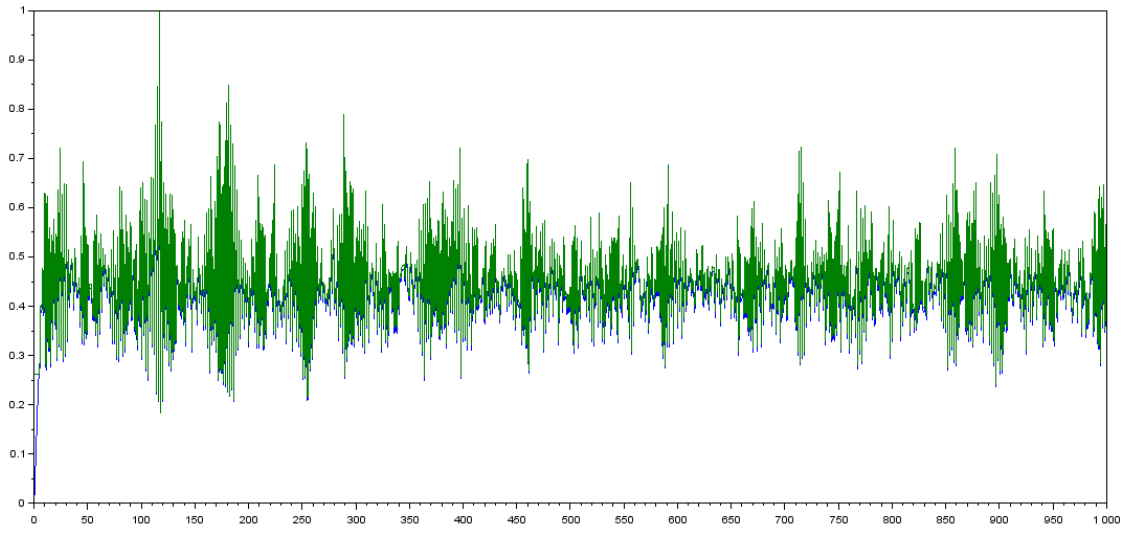

Figura 53: Saída esperada (Verde) e saída calculada (Azul) para o modelo (5.5).

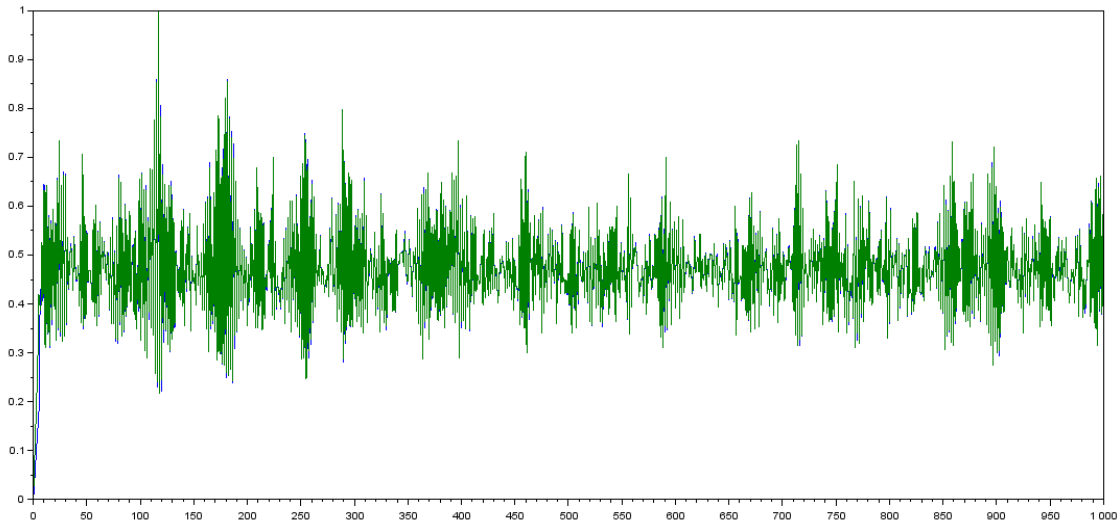

Figura 54: Saída esperada (Verde) e saída calculada (Azul) para o modelo (5.6).

O gráfico do modelo original encontra-se na cor verde e o do modelo encontrado na cor azul, porém como o algoritmo encontrou um modelo com erro quadrático de3.18407e-005 para o modelo (5.5), e 2.46923e-007 para o modelo (5.6), os dois aparentam estar sobrepostos, pode-se notar melhor a diferença no gráfico da figura 53, porém de forma sutil. Ressalta-se que para encontrar o sistema foram utilizadas apenas as primeiras 500 amostras, e que da amostra 501 à amostra 1000 são dados utilizados para validação.

Como o modelo encontrado não possui conhecimento prévio, deve-se utilizar análise estatística a fim de validar o mesmo. A análise foi realizada e encontra-se nas figuras 55 a 58.

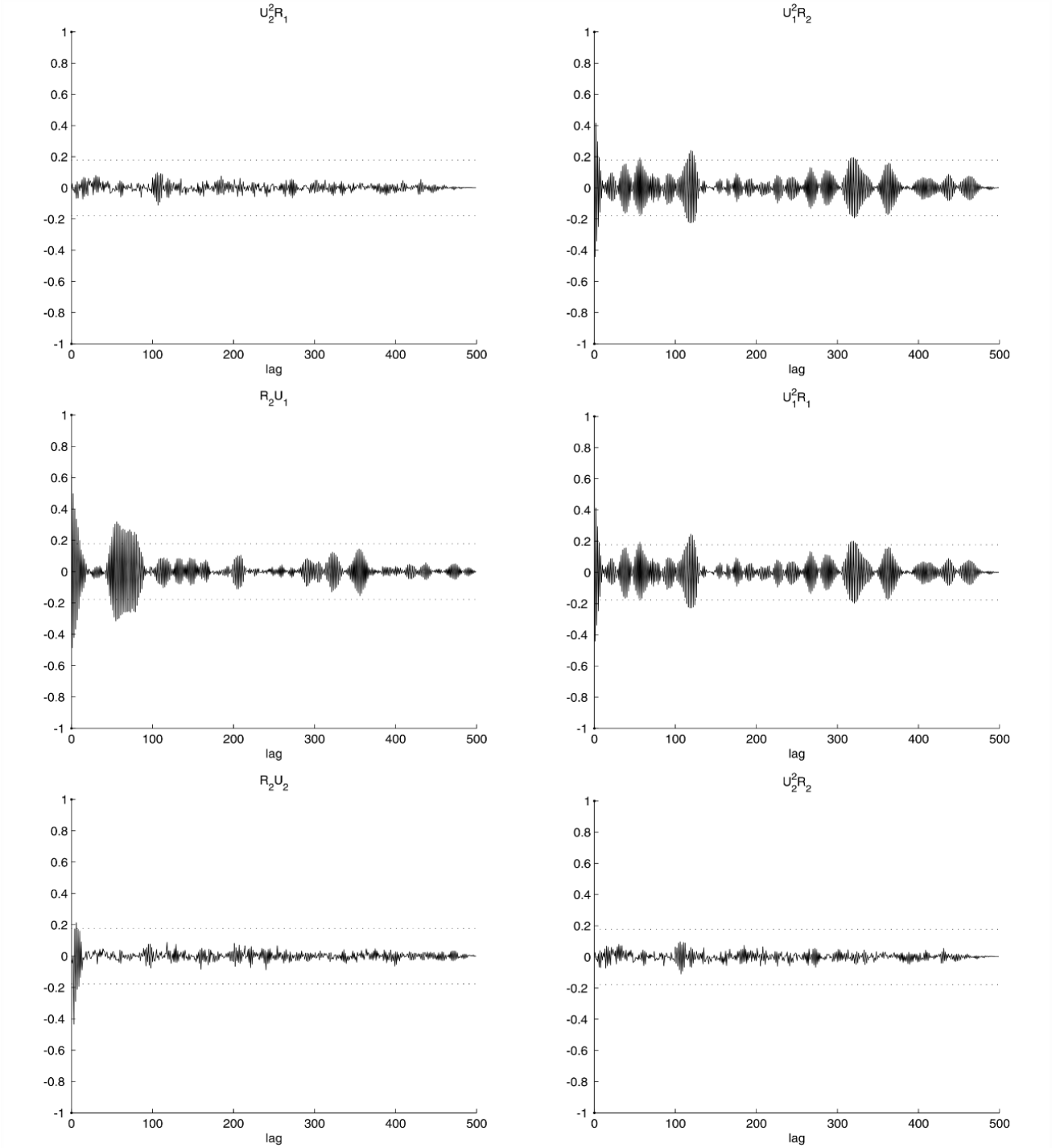

Figura 55: Análise estatística dos modelos calculados (5.5) e (5.6).

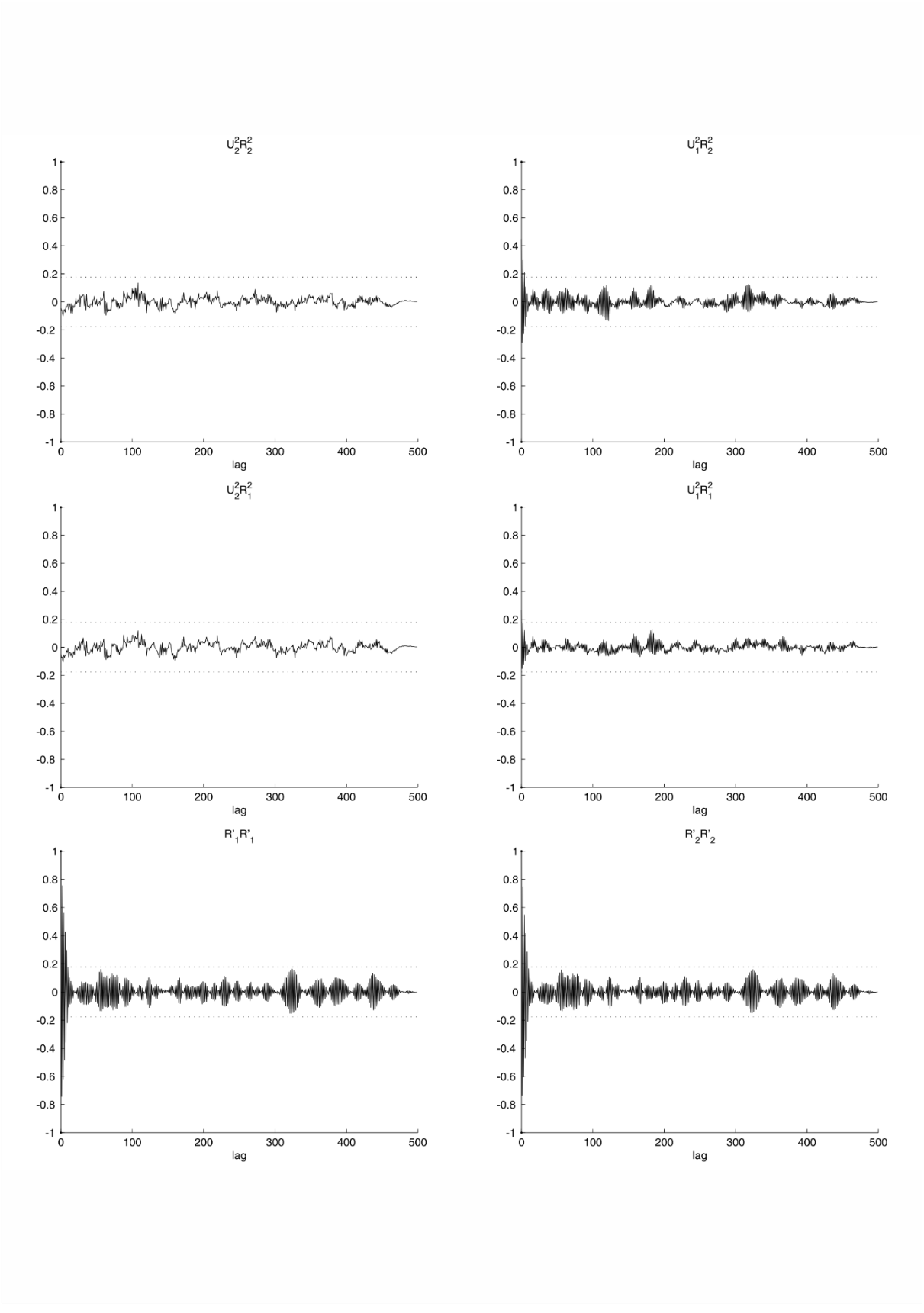

Figura 56: Análise estatística dos modelos calculados (5.5) e (5.6).

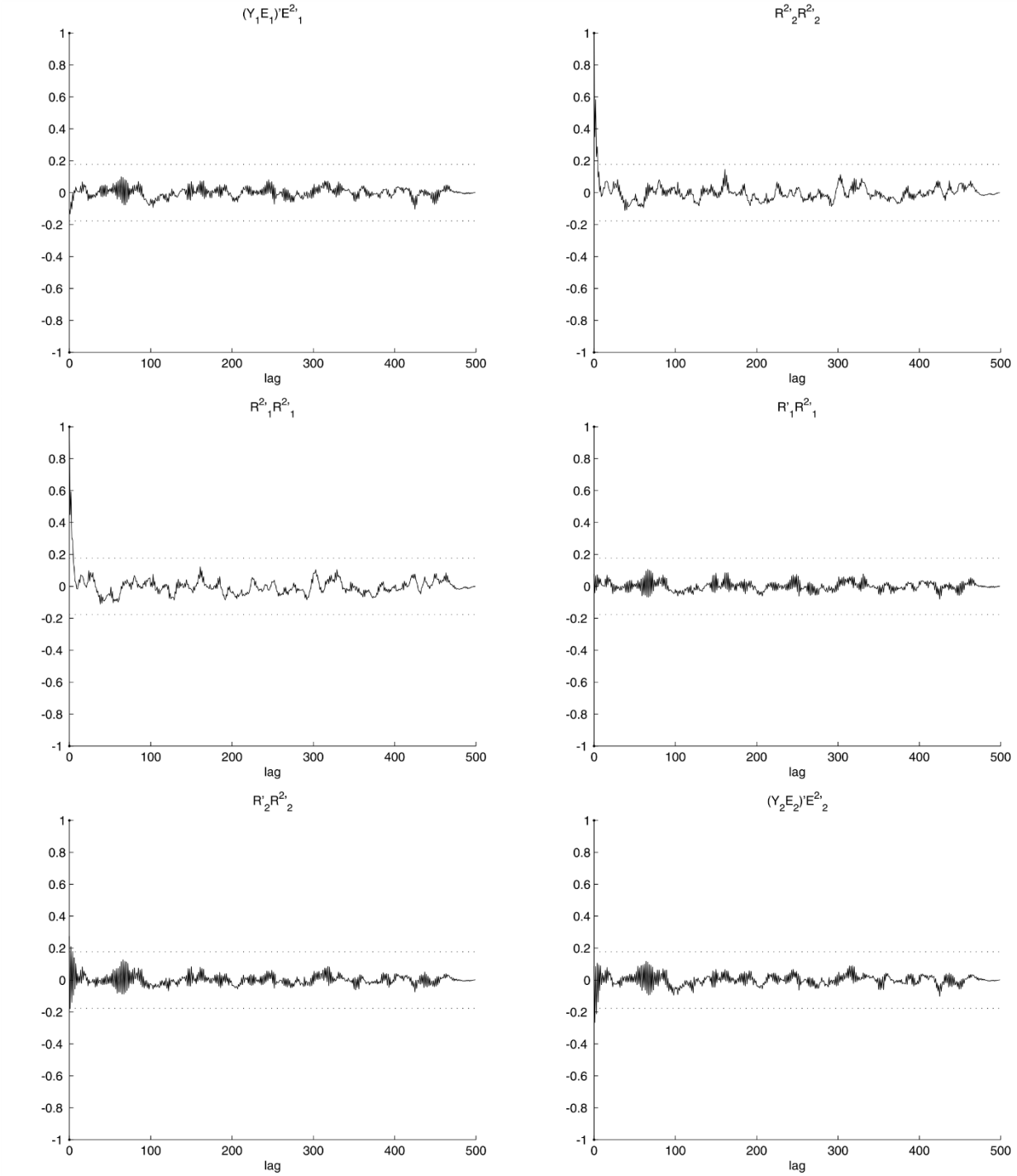

Figura 57: Análise estatística dos modelos calculados (5.5) e (5.6).

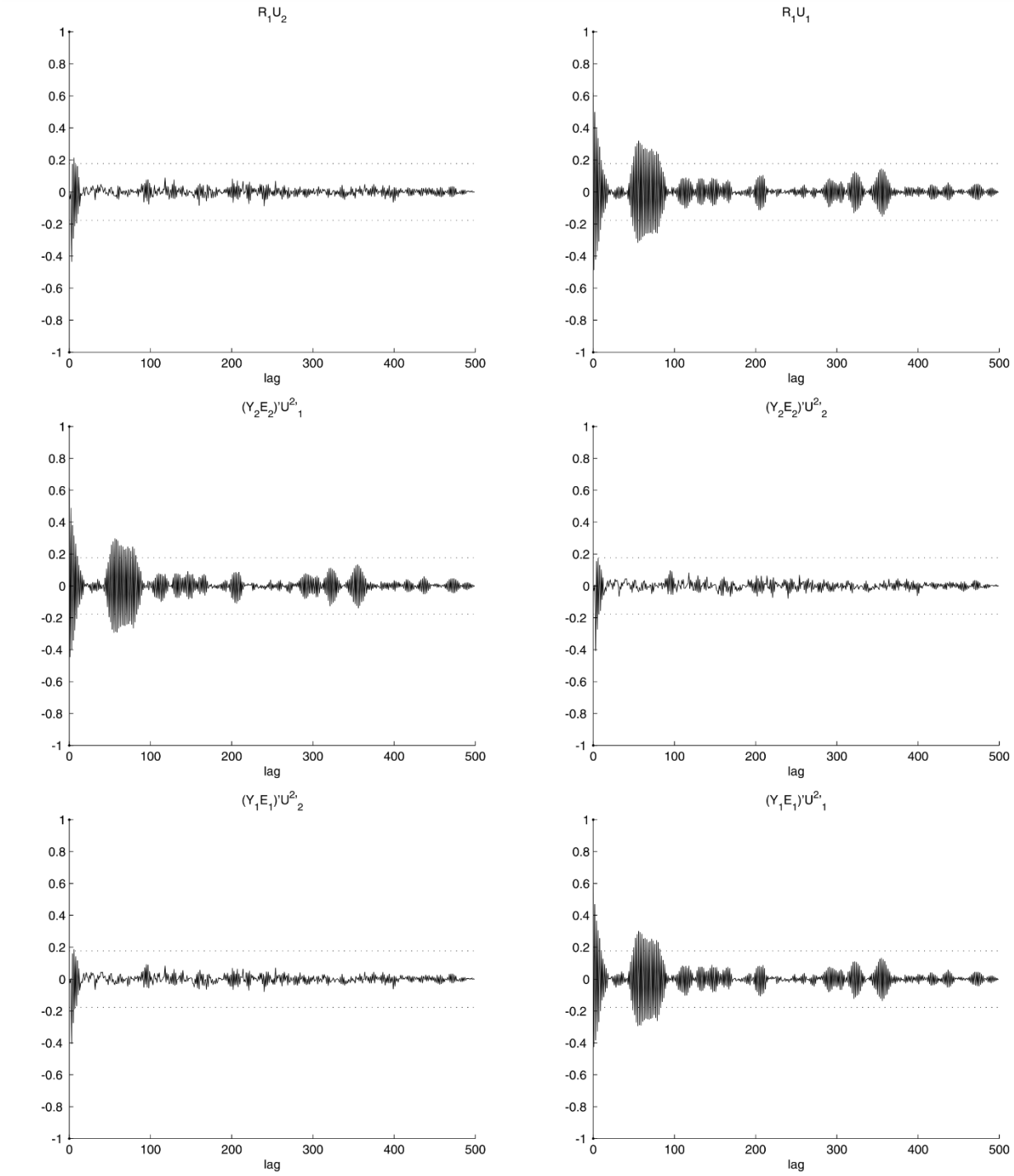

Figura 58: Análise estatística dos modelos calculados (5.5) e (5.6).

No geral o modelo apresentou resultados satisfatórios, porém como pode-se visualizar pelos gráficos que envolvem a entrada U_1, alguns pontos saem das especificações. O que indica que provavelmente ainda há informação contida na entrada U_1 que não foram modelados, porém o algoritmo não conseguiu encontrar um modelo melhor para representar os dados. Isto ocorre também no começo de alguns gráficos. Considerando que são necessárias algumas amostras iniciais para que o cálculo se estabeleça (atraso da equação) é aceitável que alguns pontos fujam um pouco dos limites no começo do gráfico.

6.4) Comentários Finais

Neste capítulo o algoritmo é validado através do uso de sistemas do tipo não lineares SISO e MIMO.Tal validação se deu para garantir a capacidade de modelar sistemas já conhecidos,a fim de validar a capacidade do algoritmo em encontrar tais modelos. A metodologia proposta apresentou resultados satisfatórios, mesmo para casos que apresentam certas dificuldades(MORAIS, 2013).

Quando necessário, testes estocásticos e dinâmicos foram aplicados aos modelos obtidos e nenhum dos testes foram insatisfatórios.

A metodologia com uso do DE para gerar regressores com expoentes reais apresenta muito potencial para futuros estudos.

Capítulo 7:
Conclusão e Trabalhos Futuros

Neste capitulo é feita a análise final do trabalho, sua viabilidade, os prós e os contras e toda a perspectiva futura gerada com base nos resultados obtidos.

Para isso, são analisados os resultados e as validações geradas pelo aplicativo desenvolvido. Considerando as comparações em diferentes metodologias e técnicas utilizadas nos algoritmos do mesmo, priorizando o paralelismo sempre que vantajoso.

E assim propiciando um sistema que minimize o esforço computacional e maximiza a utilização dos recursos computacionais no processo de identificação em sistemas NARMAX Fracionário.

7.1) Conclusões

Para o desenvolvimento deste trabalho realizou-se uma pesquisa de revisão bibliográfica relevante e atual nas áreas de identificação de sistemas, algoritmos genéticos, algoritmos computacionais e performance computacional, que demonstraram uma certa carência de trabalhos que visem melhorar a performance de algoritmos de evolução diferencial em identificação de sistemas.

O trabalho em um primeiro momento constata que a escolha de bibliotecas externas a serem utilizadas deve ser estudada com cautela, pois tal escolha pode impactar de forma considerável no desempenho final do aplicativo bem como em seu desenvolvimento, como apresenta a seção 5.2.

Em outra análise verifica-se que vários algoritmos obtêm resultados satisfatórios quando aplicados em forma paralela, quando em comparação à sua forma sequencial, porém não são todos os algoritmos que tem performance melhor em sua forma paralela do que na forma sequencial, conforme demonstrado no capítulo 5.4.1, 5.4.2.2 e 5.4.3.

Por fim destaca-se a importância do uso de algoritmos otimizados para solucionar problemas comuns à programação, como demonstrado pelo capítulo 5.4.2, onde o uso de um algoritmo otimizado mostrou-se promissor.

Os algoritmos retirados de (MORAIS, 2013) foram implementados, validados e comprovados, e demonstram grande potencial na identificação de modelos não lineares com expoentes reais, tanto modelos SISO quando MIMO. Os testes realizados nas seções 6.2 e 6.3 comprovam a validade dos algoritmos bem como sua capacidade em encontrar exatamente o modelo gerador dos dados, mesmo quando os dados estão contaminados com um erro.

Desta forma, esse trabalho contribuiu com técnicas de paralelismo aplicadas em Algoritmos evolutivos e em Identificação de Sistemas ao ponto de propor e validar uma abordagem diferente para o processo da roleta e para o processo de paralelismo, e estes podem ser utilizados em outros trabalhos científicos envolvidos com estes temas.

Os testes realizados demonstram a factibilidade do trabalho, destacou-se a importância de um código bem arquitetado, com boa legibilidade e bem estruturado para um bom desempenho e uma boa mantenabilidade do algoritmo. Além de demonstrar que em alguns casos uma abordagem seqüencial pode ser mais satisfatória que uma paralela.

7.2) Trabalhos Futuros

Durante o desenvolvimento do presente trabalho notou-se que importantes estudos poderiam ser feitos, porém optou-se por mantê-los como trabalhos futuros.

São eles:

- Avaliar métodos que se utilizem de técnicas para sistemas heterogêneos, a fim de melhorar o desempenho do sistema, como por exemplo o uso de bibliotecas como a Open CV.

- Avaliar o uso de tecnologias e plataformas de processamento paralelo que se beneficiam do poderio computacional da GPU,como por exemplo Nvidia CUDA ou AMD Stream.

- O desenvolvimento de uma interface gráfica que visa a disponibilização do aplicativo desenvolvido para a comunidade científica.

- Aplicar a metodologia final do aplicativo, através da utilização de um conjunto de dados obtidos de sistemas dinâmicos reais diferentes dos utilizados por este trabalho.

- Implementar um módulo computacional de decimação que seja eficaz e que introduza uma quantidade mínima de erro, que visa eliminar dados espúrios de sistemas reais.

- Implementar métodos e técnicas, geralmente aplicadas em sistemas embarcados, a fim de se evitar realocação de recursos computacionais e otimização de códigos e objetos, além de otimizar ao máximo o código visando melhor desempenho.

Referências Bibliográficas

ADITYA KUMAR, A. S. S. **Rejuvenating C++ Programs through Demacrofication**. [S.l.]: IEEE, 2012.

AGUIRRE, L. A. **Digital Simulation And Discrete Modelling Of A Chaotic System**. [S.l.]: Journal of Systems Engineering, v. 4, 1994.

AGUIRRE, L. A.; BILLINGS, S. A. **Nonlinear Chaotic Systems:** Approaches And Implications For Science And Engineering. [S.l.]: EURASIP Journal on Applied Signal Processing., 1995.

AGUIRRE, L. A.; MENDES, E. M. A. M.; BILLINGS, S. A. **Smoothing Data With Local Instabilities For The Identification Of Chaotic Systems.** [S.l.]: International Journal of Control, 1996.

ALKALAJ, L. **Performance of multi-threaded execution in a shared-memory multiprocessor**. Dallas, TX: IEEE, 1991. ISBN 4372731.

AMD. **Shared Level-1 instruction-cache performance on AMD family 15h CPU's**. [S.l.]: Advanced Micro Devices, 2011.

BARRON, A. R. **Statistical properteis of artificial neural networks.** [S.l.]: IEEE, 1989.

BELL, G. Scalable, Parallel Computers: Alternatives, Issues, and challenges. **International Journal of Parallel Programming**, 22, 1994.

BIDYADHAR SUBUDHI, S. M. I. A. D. J. **A Combined Differential Evolution and Neural Network Approach to Nonlinear System Identification**. [S.l.]: IEEE, 2008.

BIDYADHAR SUBUDHI, S. M. I. A. D. J. A Combined Differential Evolution and Neural Network Approach to Nonlinear System Identification, 2008.

BILINGS S.A., V. W. S. F. **Structure detection and model validity tests in the identification of nonlinear systems**. [S.l.]: IEEE, 1983.

BILLINGS S.A., V. W. S. F. **Correlation based model validity tests for nonlinear models**. [S.l.]: [s.n.], 1986.

BILLINGS S.A., Z. Q. M. **Nonlinear model validation Using Correlation Tests**. [S.l.]: [s.n.], 1993.

BILLINGS, S. A. **Identification of nonlinear systems - a survey**. [S.l.]: IEEE, 1980.

BILLINGS, S. A. **The identification of linear and non-linear models of a turbocharged automotive diesel engine. Mechanical System and Signal Processing**. [S.l.]: [s.n.], 1989.

BOHLIN, T. **Practical Grey-box Process Identification**. 1. ed. [S.l.]: Springer-Verlag London, 2006.

BURGER, T. W. **Intel Multi-Core Processors:** Quick Reference Guide. [S.l.]: [s.n.], 2005. Disponivel em: <http://www.researchgate.net/publication/228386317_Intel_Multi-Core_Processors_Quick_Reference_Guide>. Acesso em: 10 Março 2015.

CHENG XIAO, W. Q. **Modified Parallel Differential Evolution Algorithm with Local Spectral Feature to Solve Data Registration Problems**. [S.l.]: IEEE, 2011.

CHIHA IBTISSEM, L. N. **A Hybrid Method Based on Conjugate Gradient Trained Neural Network and Differential Evolution for Non Linear systems Identification**. [S.l.]: IEEE, 2013.

CORRÊA, M. V. **Identificação de Sistemas Dinâmicos Não-Lineares Utilizando Modelos NARMAX Racionais - Aplicação a Sistemas Reais.** Belo Horizonte: [s.n.], 1997.

CORREA, S. B. B. **Probabilidade e estatística**. 2. ed. Belo Horizonte: PUC Minas Virtual, 2003.

D.K. TASOULIS, N. G. P. V. P. P. N. V. **Parallel Differential Evolution**. [S.l.]: IEEE, 2004.

DIG, D. **A Practical Tutorial on Refactoring for Parallelism**. Timisoara: IEEE, 2010.

DIG, D. A Practical Tutorial on Refactoring for Parallelism, Timisoara, n. 26, 2010.

DING, F. **System identification, Part A:** Introduction tothe identification. [S.l.]: Journal of Nanjing University of Information Science and Technology: Natural Science Edition, v. 3, 2011.

EVANS, D. **Introduction to Computing:** Explorations in Language, Logic, and Machines. Virginia : [s.n.], 2011.

FILHO, C. F. **História da Computação:** O Caminho Do Pensamento E Da Tecnologia. Porto Alegre: EdiPucRS, 2007. 205 p. ISBN ISBN 978-85-7430-691-9. Disponivel em: <http://www.pucrs.br/edipucrs/online/historiadacomputacao.pdf>. Acesso em: 12 Julho 2015.

FLYNN, M. Some Computer Organizations And Their Effectiveness. **Computers, IEEE Transactions on**, C-21, n. 9, 1972.

FRITZSON, P. **Object-oriented Modeling and Simulation with MODELICA 2.1**. Piscataway: IEEE, 2004.

GARCIA, C. **Modelagem e simulação de processos industriais e de sistemas eletromecânicos**. São Paulo: EDUSP, 1997.

GUIMARÃES, F. G. **Algoritmos de evolução diferencial para otimização e aprendizado de máquina**. Ouro Preto: Congresso Brasileiro de Redes Neurais, 2009.

HELON VICENTE HULTMANN AYALA, L. F. D. C. R. Z. F. L. D. S. C. **Cascaded Free Search Differential Evolution Applied to Nonlinear System Identification Based on Correlation Functions and Neural Networks**. [S.l.]: IEEE, 2014.

HENRI E. BAL, A. S. T. **Distributed Programming With Shared Data**. Great Britain: Pergamon Press, v. 16, 1991.

HESHAM EL-REWINI, M. A.-E.-B. **Advanced Computer Architecture and Parallel Processing**. New Jersey: JOHNWILEY&SONS, 2005.

HISTORY, C. http://www.computinghistory.org.uk. **http:** //www.computinghistory.org.uk, 2015. Disponivel em: <http://www.computinghistory.org.uk>. Acesso em: 10 Junho 2015.

HONGTAO ZHONG, S. A. L. S. A. M. **Extending Multicore Architectures to Exploit Hybrid Parallelism in Single-thread Applications**. [S.l.]: IEEE, 2007.

HWANG. K., X. Z. **Scalable Parallel Computing:** Technology, Architecture Programming. San Francisco: McGraw-Hill, 1998. 802 p.

INTEL. **Intel® 64 and IA-32 Architectures Software Developer's Manual**. [S.l.]: Intel, 2015. ISBN 253665-056US.

IWASE, M. et al. An Identification Method for Continuous-Time Transfer Functions Based on Nonlinear Optmization. **Proceedings of the IEEE Conference of the 28th Annual Industrial Electronics Society**, Tokyo, 3, 2002. 1978-1983.

J. SJÖBERG, Q. Z. . L. A. B. B. D. . P. Y. G. H. H. A. A. J. **Nonlinear blackbox modeling in system identification, A unified overview.** [S.l.]: Automatica, 1995.

J.A.K. SUYKENS, T. G. J. D. B. B. D. M. A. J. V. **Least Squares Support Vector Machines. World Scientific. Singapore and often non-parametric models such as neural networks are used in place of intricate mathematics**. [S.l.]: [s.n.], 2002.

JÁJÀ, J. **An Introduction to Parallel Algorithms**. [S.l.]: Addison-Wesley Pubishing Company , 1992.

JOEL H. VAN SICKEL, K. Y. L. J. S. H. **Differential Evolution and its Applications to Power Plant Control**. Niigata: IEEE, 2007.

KE WANG, X. W. J. W. M. J. G. L. G. F. X. X. **Solving Parameter Identification Problem of Nonlinear Systems Using Differential Evolution Algorithm**. Shanghai: IEEE, 2008.

KE WANG, X. W. J. W. M. J. G. L. G. F. X. X. Solving Parameter Identification Problem of Nonlinear Systems Using Differential Evolution Algorithm, Shanghai, 2008.

KREMLIOVSKY,. K. A. M. **Estimating dynamical models using generalized moment functions**. [S.l.]: [s.n.], 1999.

L. LJUNG, Q. Z. P. L. A. J. A. R. S. **An integrated system identification toolbox for linear and non-linear models**. Newcastle: 14th IFAC Symposium on System Identification., 2006.

LEI CHAI, Q. G. D. K. P. **Understanding the Impact of Multi-Core Architecture in Cluster Computing:** A Case Study with Intel Dual-Core System. Rio De Janeiro: IEEE, 2007. ISBN 10286223.

LEONTARITIS, I. A. **Input-output parametric models for non-linear systems part I:** deterministic non-linear systems. [S.l.]: [s.n.], 1985.

LI FU, P. L. **The Research Survey of System Identification Method**. [S.l.]: Fifth International Conference on Intelligent Human-Machine Systems and Cybernetics, 2013.

LINDEN, R. **Algoritmos Genéticos**. Rio de Janeiro: Brasport, 2008. ISBN 2ª ed.

LUIGI BIANCO, L. F. **Railway maintenance purpose and improvement**. Aachen: IEEE, 2015.

MARKUS KUSANO, C. W. **CCmutator:** A Mutation Generator for Concurrency Constructs in Multithreaded C/C++ Applications. Palo Alto, USA: IEEE, 2013.

MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. **ACM Transactions on Modeling and Computer Simulation**, v. 8, n. 1, p. 3-30, 1998. ISSN doi:10.1145/272991.272995.

MORAIS, J. S. D. **ALGORITMO DIFERENCIAL EVOLUTIVO APLICADO À IDENTIFICAÇÃO DE SISTEMAS DINAMICOS NÃO-LINEARES**. Uberlândia: [s.n.], 2013.

MORETTIN, P. A.; BUSSAB, W. O. **Estatística Básica**. São Paulo: Saraiva, 2010.

NOSHADI, A. **Genetic Algorithm-based System Identification of Active Magnetic Bearing System:** A Frequency-domain Approach. Taichung, Taiwan: IEEE, 2014.

OLIVEIRA, F. **STAGE:** an Integrated Environment for Statistical Test ScriptGeneration Test and Fault Tolerance Workshop. Gramado,: [s.n.], 2004.

PAIVA, R. P. **Identificação Neuro-Difusa - Aspectos de Interoperabilidade**. Universidade de Coimbra: [s.n.], 1999.

PAVEL KROMER, J. P. V. S. **Parallel Differential Evolution in Unified Parallel C**. [S.l.]: IEEE, 2013.

PAVEL KROMER, J. P. V. S. A. A. **A Comparison of Many-threaded Differential Evolution and Genetic Algorithms on CUDA**. [S.l.]: IEEE, 2011.

PEARSON, R. K. Selecting Nonlinear Model Structures for Computer Control. **Journal of Process Control**, v. 13, p. 1-26, 2003.

PEVIANI, C. R. T. **Algoritmos Paralelos Realísticos Para a Maior Subsequência Comum**. Mato Grosso do Sul: [s.n.], 2009.

RAINER STORN, K. P. **Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces**. [S.l.]: [s.n.], 1995.

RAINER STORN, K. P. **Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces**. [S.l.]: Journal of Global Optimization, 1996.

RANCE RODRIGUES, A. A. I. K. K. **Scalable Thread Scheduling in Asymmetric Multicores for Power Efficiency**. New York: IEEE, v. 24th International Symposium on Computer Architecture and High Performance Computing, 2012.

SANTOS, S. C. **Identificação Multivariável Aplicada aos Processos Químicos:** Estudo de Caso. São Carlos: Universidade Federal de São Carlos, 2000.

SAUTER, M. **Beyond 3G – Bringing Networks, Terminals and the Web Together**. Nortel, Germany: Wiley, 2009. ISBN 978-0-470-75188-6.

SILVA, J. D. C. **Infra-estrutura de Componemtes Paralelos para Aplicações de Computação de Alto Desempenho**. Fortaleza: [s.n.], 2008.

SWATI SWAYAMSIDDHA, S. B. H. P. T. **Blind Identification of Nonlinear MIMO System Using Differential Evolution Techniques and Performance Analysis of Its Variants**. [S.l.]: IEEE, 2015.

SWATI SWAYAMSIDDHA, S. M. H. P. T. **IDENTIFICATION OF NONLINEAR DYNAMIC SYSTEMS USING DIFFERENTIAL EVOLUTION BASED UPDATE ALGORITHMS AND CHEBYSHEV FUNCTIONAL LINK ARTIFICIAL NEURAL NETWORK**. Mumbai: IEEE, 2013.

T. HASTIE, R. T. A. J. F. **The Elements of Statistical Learning**. [S.l.]: [s.n.], 2001.

TOP500, S. C. http://www.top500.org. **Top500**, 2015. Disponivel em: <http://www.top500.org>. Acesso em: 28 Julho 2015.

VAPNIK, V. **Statistical Learning Theory**. Wiley: [s.n.], 1998.

WEN-HSIEN HO, S.-H. C. J.-H. C.-K. L. M.-D. J. C.-M. L. **Application of Improved Differential Evolution Approach on Parameter Identification of Chen and Lü Chaotic Systems**. [S.l.]: IEEE, 2009.

XIAOCEN XUE, J. L. W. X. **Nonlinear System Identification with Modified Differential Evolution and RBF Networks**. Nanjing: IEEE, 2012.

XIAOCEN XUE, J. L. W. X. Nonlinear System Identification with Modified Differential Evolution and RBF Networks, Nanjing, 2012.

XIAOCEN XUE, Z. D. W. X. J. L. **A New Neuro-Fuzzy Approach for Nonlinear System Identification Based on Differential Evolution**. [S.l.]: IEEE, 2012.

YOUYUN AO, H. C. **Experimental Study on Differential Evolution Strategies**. Xiamen: IEEE, 2009.

ZHEN LI, A. J. . A. F. W. **Discovery of Potential Parallelism in Sequential Programs**. Lyon: IEEE, 2013.

ZIVIANE, N. **Projeto de algoritmos**. São Paulo: Thomson Pioneira, 2005.

Apêndice A

Estimador de Mínimos Quadrados Para o Modelo Racional

Estimação dos Parâmetros

O desenvolvimento de uma rotina de estimação de parâmetros utilizando mínimos quadrados (Billings e Zhu, 1991) é a razão primordial para se transformar o modelo racional em uma expressão linear nos parâmetros. Geralmente esta rotina é desenvolvida tendo sido considerada que a estrutura do modelo é conhecida a priori.

Estimador usando método dos mínimos quadrados

Aplicando diretamente a equação dos mínimos quadrados tem-se:

$$\hat{\Theta} = \left[\phi^T \phi\right]^{-1} \phi^T Y \qquad \text{(AN.1)}$$

Onde

$$\phi^T = \left[P(1)^T \quad \cdots \quad P(N)^T\right] =$$

$$= \begin{bmatrix} p_{n1}(1) & \cdots & p_{n1}(N) \\ \vdots & & \vdots \\ p_{nnum}(1) & \cdots & p_{nnum}(N) \\ -p_{d2}(1)\left(\dfrac{a(1)}{b(1)} + e(1)\right) & \cdots & p_{d2}(N)\left(\dfrac{a(N)}{b(N)} + e(N)\right) \\ \vdots & & \vdots \\ -p_{dden}(1)\left(\dfrac{a(1)}{b(1)} + e(1)\right) & \cdots & p_{dden}(N)\left(\dfrac{a(N)}{b(N)} + e(N)\right) \end{bmatrix} \qquad \text{(AN.2)}$$

E

$$Y = \left[Y(1) \quad \cdots \quad Y(N)\right]^T \qquad \text{(AN.3)}$$

Onde N indica o número de amostras e conforme (BN.1) ϕ pode incluir termos de ruído.

Polarização de $\hat{\Theta}$

Pode-se estimar os parâmetros de um modelo linear nos parâmetros do tipo $y(t) = p^T\Theta + e(t)$ a partir das equações (AN.4).

(AN.4)

$$y = P\Theta + e$$
$$\hat{\Theta} = Ay$$

Onde P é a matriz dos regressores e A é uma matriz cujos elementos dependem dos regressores.

$$\begin{aligned} E[Ay] - \Theta = 0, & \\ &= E\left[A(P\Theta + e)\right] - \Theta \\ &= E[AP - I]\Theta + E[Ae] \\ &= \left(E[AP] - I\right)\Theta + E[Ae] \end{aligned} \quad \text{(AN.5)}$$

Na equação (AN.5) a matriz Θ é considerada determinística, nesta equação verifica-se que a polarização será nula se:

- $E[AP] = I$;
- Não houver correlação entre o ruído e os elementos de A;
- O ruído possuir média nula.

Analisando a equação para estimação de mínimos quadrados (CN.1) e a propriedade do limite da probabilidade (Wilks, 1962) para duas matrizes A e B, ambas funções das mesmas variáveis aleatórias, tem-se:

$$P\lim(A \quad B) = P\lim A \; P\lim B \quad \text{(AN.6)}$$

Considerando que a equação (AN.7)existe e aplicando a propriedade do limite da probabilidade obtém-se a equação (AN.8).

$$\begin{aligned} A &\equiv P\lim\left[\frac{1}{N}\phi^T\phi\right]e \\ B &\equiv P\lim\left[\frac{1}{N}\phi^T Y\right] \end{aligned} \quad \text{(AN.7)}$$

$$P\lim\left[\hat{\Theta}\right] = P\lim\left[\left[\phi^T\phi\right]^{-1}\phi^T Y\right] = P\lim\left[\phi^T\phi\right]^{-1} P\lim\left[\phi^T Y\right] \quad \text{(AN.8)}$$

Supondo-se que as sequências de entrada e de saída são fixas e que o comprimento dos dados N é suficientemente grande, tem-se:

$$P\lim\left[\frac{1}{N}\phi^T\phi\right] \approx \frac{1}{N}\phi^T\phi$$
$$P\lim\left[\frac{1}{N}\phi^T Y\right] \approx \frac{1}{N}\phi^T Y \quad \text{(AN.9)}$$

Onde:

$$\phi^T\phi = \begin{bmatrix} \sum_{k=1}^{N} p_{n1}^2(t) & \cdots & \sum_{k=1}^{N} p_{n1}(t)\,p_{nnum}(t) & \cdots & -\sum_{k=1}^{N} p_{n1}(t)\,p_{d2}(t)\frac{a(t)}{b(t)} & \cdots & -\sum_{k=1}^{N} p_{n1}(t)\,p_{dden}(t)\frac{a(t)}{b(t)} \\ \vdots & & \vdots & & \vdots & & \vdots \\ \sum_{k=1}^{N} p_{nnum}(t)\,p_{n1}(t) & \cdots & \sum_{k=1}^{N} p_{nnum}^2(t) & \cdots & -\sum_{k=1}^{N} p_{nnum}(t)\,p_{d2}(t)\frac{a(t)}{b(t)} & \cdots & -\sum_{k=1}^{N} p_{nnum}(t)\,p_{dden}(t)\frac{a(t)}{b(t)} \\ -\sum_{k=1}^{N} p_{d2}(t)\,p_{n1}(t)\frac{a(t)}{b(t)} & \cdots & -\sum_{k=1}^{N} p_{d2}(t)\,p_{nnum}(t)\frac{a(t)}{b(t)} & \cdots & -\sum_{k=1}^{N} p_{d2}^2(t)\left(\frac{a(t)}{b(t)}\right)^2 & \cdots & -\sum_{k=1}^{N} p_{d2}(t)\,p_{dden}(t)\left(\frac{a(t)}{b(t)}\right)^2 \\ \vdots & & \vdots & & \vdots & & \vdots \\ -\sum_{k=1}^{N} p_{dden}(t)\,p_{n1}(t)\frac{a(t)}{b(t)} & \cdots & -\sum_{k=1}^{N} p_{dden}(t)\,p_{nnum}(t)\frac{a(t)}{b(t)} & \cdots & -\sum_{k=1}^{N} p_{dden}(t)\,p_{d2}(t)\left(\frac{a(t)}{b(t)}\right)^2 & \cdots & -\sum_{k=1}^{N} p_{dden}^2(t)\left(\frac{a(t)}{b(t)}\right)^2 \end{bmatrix} \quad \text{(AN.10)}$$

$$+\begin{bmatrix} 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \cdots & \sigma_e^2\sum_{t=1}^{N} p_{d2}^2(t) & \cdots & \sigma_e^2\sum_{t=1}^{N} p_{d2}(t)\,p_{den}(t) \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & \sigma_e^2\sum_{t=1}^{N} p_{den}(t)\,p_{d2}(t) & \cdots & \sigma_e^2\sum_{t=1}^{N} p_{dden}^2(t) \end{bmatrix}$$

e

$$\phi^T Y = \begin{bmatrix} \sum_{t=1}^{N} p_{n1}(t)\, p_{d1}(t)\left(\frac{a(t)}{b(t)} + e(t)\right) \\ \vdots \\ \sum_{t=1}^{N} p_{nnum}(t)\, p_{d1}(t)\left(\frac{a(t)}{b(t)} + e(t)\right) \\ \sum_{t=1}^{N} p_{d2}(t)\, p_{d1}(t)\left(\frac{a(t)}{b(t)} + e(t)\right)^2 \\ \vdots \\ \sum_{t=1}^{N} p_{dden}(t)\, p_{d1}(t)\left(\frac{a(t)}{b(t)} + e(t)\right)^2 \end{bmatrix} = \begin{bmatrix} \sum_{t=1}^{N} p_{n1}(t)\, p_{d1}(t)\left(\frac{a(t)}{b(t)}\right) \\ \vdots \\ \sum_{t=1}^{N} p_{nnum}(t)\, p_{d1}(t)\left(\frac{a(t)}{b(t)}\right) \\ \sum_{t=1}^{N} p_{d2}(t)\, p_{d1}(t)\left(\frac{a(t)}{b(t)}\right)^2 \\ \vdots \\ \sum_{t=1}^{N} p_{dden}(t)\, p_{d1}(t)\left(\frac{a(t)}{b(t)}\right)^2 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -\sigma_e^2 \sum_{t=1}^{N} p_{d2}(t)\, p_{d1}(t) \\ \vdots \\ -\sigma_e^2 \sum_{t=1}^{N} p_{dden}(t)\, p_{d1}(t) \end{bmatrix} \quad \text{(AN.11)}$$

Para $N \to \infty$ a estimativa de $\hat{\Theta}$ converge em probabilidade para $\left[\phi^T \phi\right]^{-1} \phi^T Y$.

Reescrevendo a equação (AN.11) usando notação matricial, tem-se:

$$\begin{aligned} \phi^T \phi &= \left[\phi^T \phi\right]_{t-1} + \sigma_e^2 \Psi \\ \phi^T Y &= \left[\phi^T Y\right]_{t-1} + \sigma_e^2\, \psi \end{aligned} \quad \text{(AN.12)}$$

Onde:

$$\Psi = \begin{bmatrix} 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \cdots & \sum_{t=1}^{N} p_{d2}^2(t) & \cdots & \sum_{t=1}^{N} p_{d2}(t)\, p_{den}(t) \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & \sum_{t=1}^{N} p_{den}(t)\, p_{d2}(t) & \cdots & \sum_{t=1}^{N} p_{dden}^2(t) \end{bmatrix} \quad \text{(AN.13)}$$

$$\psi = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -\sum_{t=1}^{N} p_{d2}(t)\, p_{d1}(t) \\ \vdots \\ -\sum_{t=1}^{N} p_{dden}(t)\, p_{d1}(t) \end{bmatrix} \quad \text{(AN.14)}$$

Todos os termos envolvendo $e(t)$ aparecem em $\sigma_e^2\Psi$ e $\sigma_e^2\,_{\psi}$, que são os termos de ruído. O subscrito $(t-1)$ indica que apenas termos de ruído até $(t-1)$ estão presentes (desde que $e(t-j)$, $j \le 1$) (Billings e Zhu, 1991).

O estimador de parâmetros dado em (CN.1) pode ser reescrito como:

$$\begin{aligned}\hat{\Theta} &= \left[\phi^T\phi\right]^{-1}\phi^T Y \\ &= \left[\left[\phi^T\phi\right]_{t-1}^{-1} + \sigma_e^2\Psi\right]^{-1}\left[\left[\phi^T Y\right]_{t-1} + \sigma_e^2\psi\right]\end{aligned} \qquad \text{(AN.15)}$$

Comparando a equação (AN.15) com a (AN.5) verifica-se que $\left[\phi^T\phi\right]^{-1}\phi^T$ não pode estar correlacionado com o ruído para se evitar a polarização do estimador.

A equação (AN.15) mostra que linearização os parâmetros do modelo racional cria dois termos adicionais $\sigma_e^2\Psi$ e $\sigma_e^2\,_{\psi}$, provando a correlação entre $\left[\phi^T\phi\right]^{-1}\phi^T$ e o ruído. Infelizmente essa correlação causa polarização na estimação dos parâmetros, mesmo para uma sequência $e(t)$ branca. O problema surge porque os termos $y(t)p_{dj}(t)$ em (BN.4) contêm implicitamente $e(t)$ (Billings e Zhu, 1991).

Estimador de Mínimos Quadrados para o Modelo Racional – RME

O algoritmo de mínimos quadrados estendido para modelos com equações lineares diferenciais tem sido exaustivamente usado ao longo dos anos (Goodwin e Payne 1977; Norton 1986; Ljung 1987). Ponto comum em todos os algoritmos é que a correlação do vetor de resíduos é reduzida pela incorporação do modelo de ruído na matriz dos regressores.

Este procedimento é um método iterativo onde os parâmetros dos termos de processo e dos termos de ruído são estimados até a polarização ser reduzida a níveis

aceitáveis. Na Figura AN.1 tem-se o algoritmo sugerido em (Billings e Zhu, 1991), para implementação do método acima.

1º passo. Usar um algoritmo de mínimos quadrados ordinários (OLS) para computar:

$$\hat{\Theta} = \left[\phi^T \phi\right]^{-1} \phi^T Y$$

Esta estimação fornece os valores iniciais dos parâmetros para os cálculos seguintes.

2º passo. Calcular a sequência de resíduos

$$\xi(t) = y(t) - \frac{a(\ldots, \hat{\Theta}(i-1))}{b\left(\ldots, \hat{\Theta}(i-1)\right)}$$

Estimar a variância dos resíduos $\hat{\sigma}_\xi^2$ como

$$\hat{\sigma}_\xi^2(i) = \frac{1}{N - md} \sum_{t=md+1}^{N} \left(y(t) - \frac{a(\ldots, \hat{\Theta}(i-1))}{b\left(\ldots, \hat{\Theta}(i-1)\right)} \right)^2$$

Onde i é o número da iteração, N o número de amostras e *md* é o máximo atraso presente nos termos.

3º passo. Utilizando as equações (AN.10), (AN.11), (AN13) e (AN14), atualize as matrizes $\phi^T\phi$ e Ψ, vetores $\phi^T Y$ e ψ usando a sequência de resíduos calculada no 2º passo. Note que em tais matrizes os resíduos podem aparecer dentro dos polinômios $a(t)$ e $b(t)$.

4º passo. Calcule os novos parâmetros como

$$\hat{\Theta}(i) = \left[\phi^T \phi - \sigma_\xi^2(i)\Psi\right]^{-1} \left[\phi^T Y - \sigma_\xi^2(i)\psi\right]$$

5ª passo. Retorne ao 2º passo e repita até os parâmetros estimados e $\hat{\sigma}_\xi^2$ convergir em um valor constante.

Figura AN.1 - Algoritmo de Mínimos Quadrados Estendido proposto em (Billings e Zhu, 1991).