

**UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA
DE ELETRICIDADE**

Determinação de Aproximação Linear por Partes de Funções
Não Lineares para Sistemas Embarcados Utilizando Algoritmos
Genéticos

Juan Moises Mauricio Villanueva

São Luís -MA
2005

Determinação de Aproximação Linear por Partes de Funções Não Lineares para Sistemas Embarcados Utilizando Algoritmos Genéticos

Dissertação de Mestrado submetido à Coordenação do curso de Pós-Graduação em Engenharia de Eletricidade da UFMA como parte dos requisitos para obtenção do título de mestre em Engenharia Elétrica.

Por

Juan Moises Mauricio Villanueva

03 de Março de 2005

Villanueva, Juan Moises Mauricio

Determinação de aproximação linear por partes de funções não lineares para sistemas embarcados utilizando algoritmos genéticos / Juan Moises Mauricio Villanueva. – São Luis, 2005

99f.: il.

Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal do Maranhão, 2005

1. Algoritmos Genéticos. 2. Propagação de Erros e Incertezas. 3. Sistemas Embarcados. 4, Aproximação de Funções. I. Título.

CDU 621.3: 519.688

Determinação de Aproximação Linear Por Partes de Funções Não Lineares para Sistemas Embarcados Utilizando Algoritmos Genéticos

JUAN MOISES MAURICIO VILLANUEVA

Dissertação aprovada em 03 de Março de 2005

PROF. DR. SEBASTIAN YURI CAVALCANTI CATUNDA
(ORIENTADOR)

PROF. DR. OSVALDO RONALD SAAVEDRA MENDEZ
(ORIENTADOR)

PROF. DR. MARCO AURELIO CAVALCANTI PACHECO
(MEMBRO DA BANCA EXAMINADORA)

PROF. DR. JOÃO VIANA DA FONSECA NETO
(MEMBRO DA BANCA EXAMINADORA)

Determinação de Aproximação Linear por Partes de Funções Não Lineares para Sistemas Embarcados Utilizando Algoritmos Genéticos

MESTRADO

Área de Concentração: AUTOMAÇÃO E CONTROLE

JUAN MOISES MAURICIO VILLANUEVA

Orientador: Dr. Sebastian Yuri Cavalcanti Catunda

Orientador: Dr. Osvaldo Ronald Saavedra Mendez

Programa de Pós-Graduação

Em Engenharia de Eletricidade da

Universidade Federal do Maranhão

Dedicatória

A Deus: sobre todas as coisas.

À minha mãe: Ruth Villanueva,

À meus irmãos Milagros, Moises e Carlos.

Agradecimentos

Agradeço a todos aqueles que, direta ou indiretamente, contribuíram para a elaboração desta dissertação e, de modo especial:

Aos Professores Sebastian Yuri Cavalcanti Catunda e Osvaldo Ronald Saavedra Mendez pela sua paciência e segura orientação, pela confiança em mim depositada e oportunidade de poder realizar este trabalho e, sobretudo pela amizade.

À coordenação do curso de pós-graduação em Engenharia de Eletricidade da UFMA e em especial, à professora Maria da Guia e a Alcides, pelo grande apoio e incentivo.

Aos professores do departamento de pós-graduação: João Viana, Allan Kardec, Leonardo Paucar, José Pessanha, Raimundo Freire; pelo apoio e esclarecimento.

Aos colegas do curso de pós-graduação do Departamento de Engenharia Elétrica Miriam, Yuri, Carlos, Sergio, Jorge, Marcos, Alex, Manfred, Clodomiro. Aos colegas do Laboratório de Instrumentação Eletrônica e Automação (LIEA), que estiveram comigo ao longo deste período de modo especial: Rycardo Bruno, Will Almeida, Jaderson Oliveira, Mauro Sergio, Michel Valença, Igor, Giselia Andrea, Danielle Brito e Silvangela Lima.

A Deus por não me deixar desistir, diante das dificuldades.

Resumo

Em diversas aplicações em eletrônica existe o problema de gerar valores de funções não lineares utilizando-se sistemas embarcados de baixo custo. Essas funções não lineares não podem ser implementadas diretamente devido às restrições de cálculo em ponto fixo e resolução limitada, características de arquitetura do processador empregado.

Nesta dissertação, apresenta-se um procedimento para a determinação de aproximação linear por partes de funções não lineares para sistemas embarcados de baixo custo. Para resolver este problema, desenvolveu-se um algoritmo hierárquico evolutivo que determinará a posição e número mínimo de pontos de quebra e tamanho mínimo da tabela de equivalência para armazenar esses pontos de quebra, para gerar os valores da função aproximada. A função não linear pode então ser aproximada por funções lineares a partir dos valores dos pontos de quebra encontrados. O algoritmo desenvolvido é testado para o caso de aproximação da função seno no primeiro quadrante, e os resultados obtidos são apresentados para diversas resoluções de entrada e de geração dos valores de saída.

Abstract

In several applications in electronics, the generation of nonlinear function values using low-cost embedded systems is a problem. The nonlinear functions cannot be directly implemented due to restrictions of fixed-point calculations and limited resolution that are characteristics of the architecture of the processor employed.

In this work, a procedure for determining piecewise linear approximation of nonlinear functions for a low-cost embedded system is presented. In order to solve this problem, a hierarchical evolutionary algorithm has been developed for determining the position and the minimal number of breakpoints and the minimal size of the look-up table for storing these breakpoints, for generating the approximated function values. The nonlinear function can be approximated using piecewise linear functions from the obtained breakpoints. The developed algorithm is tested using the case of approximating the first quadrant of a sine function, and the obtained results are presented for different resolutions for the input and output values generation.

Sumário

Lista de Tabelas	x
Lista de Figuras	xi
Lista de Símbolos	xiv
Capítulo 1 Introdução	1
1.1. Aproximação de Funções Não Lineares	2
1.2. Algoritmos Genéticos	3
1.3. Organização da Dissertação	4
Capítulo 2 Aproximação de Funções	6
2.1. Introdução	6
2.2. Formulação do Problema de Aproximação de Funções	7
2.3. Quantização, representação das variáveis e representação das variáveis em escalas diferentes em um sistema digital embarcado	9
2.3.1. Quantização das variáveis	9
2.3.2. Representação de variáveis	10
2.3.3. Representação das variáveis em outras escalas	10
2.4. Análise da propagação da incerteza e erro de aproximação em um sistema digital embarcado	11
2.4.1. Análise do erro de aproximação considerando a variável independente x como exata	12
2.4.2. Análise do erro de aproximação considerando que a variável independente x tem associada uma incerteza devido à conversão digital	13
2.5. Normalização da função aproximada	15
2.6. Aproximação de Funções Não Lineares utilizando Funções Lineares por Partes....	15
2.7. Dimensionamento da tabela de equivalência (LUT - Look-up Table)	16
2.8. Estudo de caso: Aproximação da Função Seno	17
2.8.1 Análise do erro de aproximação da função seno considerando a variável independente x como exata ($N = N_Y$)	18
2.8.2 Análise do erro de aproximação da função seno considerando a variável independente x associada a uma incerteza ($N \neq N_Y$)	19
2.9. Algoritmos de Busca	22
2.9.1 Algoritmo de Busca Enumerativo	23

Capítulo 3 Computação Evolutiva	25
3.1. Introdução	25
3.2. Princípios Básicos da Computação Evolutiva	26
3.3. Algoritmos Genéticos	30
3.3.1. Representação de Indivíduos (Codificação)	31
3.3.2. Definição da População Inicial	32
3.3.3. Seleção	32
3.3.4. Operadores Genéticos	35
3.4. Algoritmos Híbridos	38
Capítulo 4 Algoritmo Hierárquico Evolutivo para Determinação de Aproximação de Funções	39
4.1. Introdução	39
4.2. Formulação para determinar aproximação de funções através de algoritmos genéticos ..	41
4.2.1. Aproximação de funções não lineares por funções lineares por partes	41
4.2.2. Normalização das Soluções	41
4.3. Procedimento de Solução	42
4.3.1. Algoritmo Hierárquico	42
4.3.2. Algoritmo Genético Padrão	43
4.3.3. Algoritmo Híbrido	53
Capítulo 5 Resultados.....	56
5.1. Estudo de caso: Aproximação da função seno no primeiro quadrante.....	56
5.2. Procedimento para solução	57
5.3. Resultados utilizando o Algoritmo Genético Padrão	58
5.4. Resultados utilizando o Algoritmo Híbrido	60
5.5. Comparação Computacional entre o Algoritmo Genético Padrão e o Algoritmo Híbrido.....	61
5.6. Análise exploratória dos operadores genéticos e o algoritmo de programação matemática durante o processo evolutivo.....	63
5.7. Estratégia de variação dos parâmetros: P_c e P_m	67
Capítulo 6 Conclusões e Sugestões	70
6.1. Conclusões.....	70
6.2. Sugestões	71
Referências Bibliográficas	72
Apêndices.....	75
Apêndice A – Soluções da Aproximação da Função Seno	76
A1 Gráfico da aproximação da função seno com parâmetros $N = 8$, $N_Y = 8$, $N_T = 10$ e $m = 9$	76
A2 Gráfico da aproximação da função seno com parâmetros $N = 9$, $N_Y = 9$, $N_T = 11$ e $m = 13$	77
A3 Gráfico da aproximação da função seno com parâmetros $N = 10$, $N_Y = 10$, $N_T = 12$ e $m = 17$	78
A4 Gráfico da aproximação da função seno com parâmetros $N = 11$, $N_Y = 11$, $N_T = 14$ e $m = 22$	79
Apêndice B – Variação dos Parâmetros do Algoritmo Genético	80
Variação da probabilidade de cruzamento e mutação	80

Lista de Tabelas

Tabela 2.1. Valores de x e ângulo equivalente com duas casas decimais.	17
Tabela 4.2. Tamanho do cromossomo em bits	52
Tabela 5.1. Parâmetros e resultados da aproximação utilizando o algoritmo genético.....	58
Tabela 5.2. Pontos de quebra com parâmetros $N = 8$, $N_Y = 8$, $N_T = 10$ e $m = 9$	59
Tabela 5.3. Parâmetros e resultados da aproximação utilizando o algoritmo híbrido.....	60
Tabela 5.4. Pontos de quebra para $N = 8$, $N_Y = 8$ e $N_T = 10$	61
Tabela 5.5. Comparação de taxa de convergência do algoritmo genético padrão e o algoritmo híbrido.	62

Lista de Figuras

Figura 2.1. Aproximação linear por partes de uma função não linear.....	8
Figura 2.2. Valores aproximados de y e limites de aproximação.....	18
Figura 2.3. Erro de aproximação S e limites de erro de aproximação E_{Amin} e E_{Amax} normalizados.....	19
Figura 2.4. Limites de incerteza propagada e do erro de aproximação.....	20
Figura 2.5. Função ideal, limites de geração dos valores de y e limites da função aproximada.....	21
Figura 2.6. Erro de aproximação para $m = 4$ e $N_T = 10$, com coeficientes calculados utilizando pontos sobre a curva ideal.....	21
Figura 2.7. Erro de aproximação para $m = 4$ e $N_T = 10$, com coeficientes calculados utilizando pontos fora da curva ideal.....	22
Figura 3.1. Estrutura de um programa evolutivo.....	27
Figura 3.2. Seleção por roleta em algoritmos genéticos.....	33
Figura 3.3 Assinação do <i>fitness</i> para o ranking linear e não linear.....	35
Figura 4.1. Diagrama do procedimento hierárquico de busca.....	43
Figura 4.2. Definição do tamanho da LUT com solução para $N_T = 10$ e $m = 9$	43
Figura 4.3. Representação do cromossomo.....	45
Figura 4.4. Estrutura da população total de indivíduos.....	45

Figure 4.5. Comportamento da função de aptidão com pontos de quebra sobre a função quantizada, com parâmetros $N = N_Y = 4$, $N_T = 6$ e $m = 4$.	47
Figura 4.6. Função seno no primeiro quadrante aproximada por funções lineares por partes.	47
Figura 4.7. Erro de aproximação S e limites de erros de aproximação E_{Amin} e E_{Amax} normalizados.	48
Figura 4.8. Comportamento da função de aptidão com parâmetros $N = N_Y = 4$, $N_T = 6$ e $m = 4$.	49
Figura 4.9. Função seno no primeiro quadrante aproximada por funções lineares por partes.	49
Figura 4.10. Erro de aproximação S e limites de erros de aproximação E_{Amin} e E_{Amax} normalizados.	50
Figura 4.11. Probabilidade de seleção baseada na classificação não linear	51
Figura 4.12. Representação da Operação de Mutação.	53
Figura 5.1. Algoritmo hierárquico evolutivo: A) Composto por um algoritmo genético padrão; B) Composto por um algoritmo híbrido.	58
Figura 5.2. Erro de aproximação para a função seno com $N = 8$, $N_Y = 8$ e $N_T = 10$.	59
Figura 5.3. Aproximação para a função seno com $N = 8$, $N_Y = 8$ e $N_T = 10$.	61
Figura 5.4. Simulação do Algoritmo Genético Padrão e Algoritmo Híbrido com parâmetros $N = 8$, $N_T = 10$ e $m = 9$.	62
Figura 5.5. Simulação do Algoritmo Genético Padrão e Algoritmo Híbrido com parâmetros $N = 9$, $N_T = 11$ e $m = 13$.	63
Figura 5.6. Medida da diversidade genética (gdm)	64
Figura 5.7. Pressão de seleção dos indivíduos p_x e p_y	65
Figura 5.8. Número de cruzamentos em cada época.	66
Figura 5.9. Número de mutações em cada geração para p_x e p_y .	66
Figura 5.10. Número de novos indivíduos introduzidos pelo algoritmo de programação por época.	67
Figura 5.11. Variação das probabilidades de cruzamento e mutação (P_c , P_m).	68
Figura 5.12. Resposta da curva de probabilidade de cruzamento para um total de 1100 simulações.	69

Figura 5.13. Resposta da curva de probabilidade de mutação para 1100 simulações..... 69

Lista de Símbolos

x	Variável independente na escala real;
x_{min}	Mínimo valor da variável independente;
x_{max}	Máximo valor da variável independente;
y	Variável dependente na escala real;
y_{min}	Mínimo valor da variável dependente;
y_{max}	Máximo valor da variável dependente;
X	Variável independente em outra escala;
Y	Variável dependente em outra escala;
\sim	Variáveis com sobrescrito representam aproximações;
q	Passo de quantização;
y_q	Valores da função quantizada;
f	Função não linear;
s	Função aproximada;
S	Função aproximada em outra escala;
m	Número de pontos de quebra;
p_x	Valores dos pontos de quebra no eixo x ;
p_y	Valores dos pontos de quebra no eixo y ;
N	Resolução da variável independente;
N_Y	Resolução de geração dos valores da variável dependente;
N_T	Resolução de geração dos valores da variável independente em outra escala;
ε_x	Valor da incerteza associada à variável x ;

ε_A	Erro de aproximação;
ε_{Amin}	Erro mínimo de aproximação;
ε_{Amax}	Erro máximo de aproximação;
E_{Amin}	Erro mínimo de aproximação em outra escala;
E_{Amax}	Erro máximo de aproximação em outra escala;
$\widehat{\varepsilon}_y$	Valor máximo da incerteza associada à variável y ;
ε_p	Erro de propagação da incerteza ε_x ;
ε_{pmin}	Erro mínimo de propagação da incerteza ε_x ;
ε_{pmax}	Erro máximo de propagação da incerteza ε_x ;
ε_s	Valor da incerteza de geração dos valores da função aproximada;
t	Época do algoritmo evolutivo;
$P(t)$	População de indivíduos na época t ;
$Prob$	Probabilidade de um indivíduo de ser selecionado;
SP	Pressão de seleção;
P_c	Probabilidade de cruzamento;
P_m	Probabilidade de mutação;
ε_{AN}	Erro de aproximação normalizado;
fit	Função de aptidão (<i>Fitness</i>);
gdm	Medida da diversidade genética;

Capítulo 1

Introdução

Os sistemas embarcados são concebidos para aplicações específicas, contendo em sua estrutura componentes de hardware e software. O hardware é utilizado para interagir com o ambiente, composto por sensores, atuadores e funções de controle. O software controla as ações do hardware para realizar as funcionalidades requeridas. Os sistemas embarcados são incorporados para formar parte de sistemas complexos como por exemplo: controle digital, sistemas de comunicação, comando e controle, processamento de sinais, rastreadores, aplicações multimídias, etc [1, 2].

Em diversos exemplos de sistemas embarcados (telefones celulares, PDAs-*Personal Digital Assistant*, fornos de microondas, secretárias eletrônicas, controle de injeção de combustível, freios de automóveis, etc), são encontradas características comuns, tais como:

a) Funcionalidade específica: Um sistema embarcado usualmente executa somente um programa repetidamente.

b) Restrições de Projeto: Restrições impostas pela necessidade de portabilidade, baixo consumo, dimensões reduzidas, desempenho e produção em larga escala (custo baixo); além de restrição do ambiente de operação com capacidade de processamento em tempo real.

c) Comportamento Reativo e em Tempo Real: Muitos sistemas embarcados devem reagir continuamente a mudanças no ambiente do sistema e devem realizar certos cálculos em tempo real com restrições de tempo e sem retardo [3].

Os sistemas embarcados, também chamados de embutidos são compostos por

hardware e software com processamento dedicado para executar suas tarefas programadas. Conseqüentemente, diversas soluções de sistemas embarcados são implementadas em microprocessadores ou microcontroladores com certo poder computacional, no qual os cálculos podem ser realizados em ponto flutuante (*float-point*). Entretanto, diversas aplicações embarcadas requerem de uso de hardware em ponto fixo (*fixed point*). Isto pode ser justificado por motivos tais como redução de custos, complexidade e aumento da velocidade de processamento. Assim os sistemas embarcados baseados em hardware de baixo custo podem ser compostos de: microcontroladores de oito bits, FPGA ou ASIC em que geralmente os cálculos são realizados em ponto fixo e com resolução limitada [4].

1.1. Aproximação de Funções Não Lineares

Em diversas aplicações em eletrônica existe o problema de aproximar ou gerar valores de funções não lineares utilizando sistemas embarcados. Exemplos destes sistemas são descritos a seguir: Na análise de circuitos eletrônicos precisa-se modelar os dispositivos eletrônicos por funções não lineares [5]; em aplicações de processamento de sinais, processamento de imagens, reconhecimento de padrões é necessário gerar tabelas a partir de funções [6]; na análise e controle de sistemas não lineares usam-se funções não lineares [7]; em sistemas digitais de posicionamento, geralmente necessita-se converter a informação de ângulo em posição e vice-versa, utilizando funções trigonométricas [8]; em instrumentação, diversas aplicações necessitam gerar funções não lineares para reconstrução dos valores de medição, que podem ser utilizadas para compensação das não linearidades do sensor ou para compensação da influência de grandezas que causem interferência [4].

Essas funções não lineares podem ser implementadas diretamente em sistemas embarcados com capacidade de processamento de cálculo em ponto flutuante. Entretanto, em diversas aplicações se faz necessário a utilização de sistemas embarcados em que a implementação destas funções não lineares não é direta devido às restrições de cálculo em ponto fixo e resolução limitada, características da arquitetura do processador empregado.

O problema de aproximação de funções tem recebido bastante atenção, como pode ser observado nas referências [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. Os métodos para realizar essas aproximações incluem, por exemplo, redes neural artificial (RNA) e *splines* cúbica. Estas soluções, no entanto, apresentam inconvenientes de implementação, como por exemplo para: uma RNA que tenha uma função de ativação não linear dos neurônios e que os pesos das conexões devam ser armazenados em valores reais; *splines* que também usam coeficientes

reais e necessitam de um grande número de multiplicações nas operações de avaliação. Ainda, nas diversas abordagens apresentadas, a variável independente é sempre considerada como exata, o que não ocorre geralmente na prática.

Em [4], apresenta-se um procedimento para definir as restrições e limites de aproximação de uma função não linear em sistemas embarcados que utilizem cálculo em ponto fixo e com resolução limitada. Nesta abordagem leva-se em consideração a incerteza associada à variável independente e seu efeito na geração da variável dependente. São considerados também, aspectos como saturação em relação às representações máximas e mínimas de números e efeitos de quantização devido à resolução de armazenamento das variáveis.

Uma opção interessante para implementar aproximação de funções não lineares em sistemas embarcados com cálculo em ponto fixo e com resolução limitada é a utilização de tabelas de equivalência (*LUT-Look-Up Table*), com interpolação linear, também chamada de interpolação poligonal [15]. Neste caso, definem-se os pontos de quebra da função aproximada que são armazenados em uma tabela de equivalência (LUT). Os valores de interesse da função são gerados utilizando a função aproximada definida pelos pontos de quebra imediatamente inferior e superior ao valor de entrada [4].

A utilização de LUTs para aproximação de funções não lineares geralmente aborda o problema de minimização do erro de aproximação sem levar em consideração o número de células necessárias para a construção da LUT, sendo um inconveniente para sistemas embarcados com limitações de espaço de memória. Uma alternativa de solução para implementar a função não linear com interpolação linear é utilizar os valores dos pontos de quebra e as restrições de limites de aproximação para encontrar o número de células da LUT, de forma a minimizar o tamanho de memória utilizada da LUT, garantindo a exatidão desejada na geração dos valores estabelecidos na fase do projeto. Isto é útil em sistemas embarcados de baixo custo, com capacidade de cálculo em ponto fixo, resolução limitada e com limitações de espaço de memória devido à restrição da arquitetura do projeto [15].

1.2. Algoritmos Genéticos

Algoritmos genéticos (AGs) têm sido aplicados com sucesso a diversos problemas de otimização [16]. Os AGs são métodos de busca estocásticos que imitam a evolução biológica natural. Os AGs estão compostos por uma seqüência de passos até a solução, sendo que estes passos são os mesmos para uma ampla gama de problemas, fornecendo robustez e

flexibilidade.

Os algoritmos genéticos realizam um processo de busca multi-direcional, mantendo uma população de soluções potenciais trocando informações entre si. A população é processada mediante uma evolução simulada cujo objetivo é: a cada geração reproduzir soluções relativamente “boas”, enquanto são eliminadas soluções relativamente “ruins”. Para distinguir entre diferentes soluções utiliza-se uma função de aptidão (*fitness*) que simula o papel da pressão exercida pelo ambiente sobre o indivíduo [17].

A utilização dos métodos para aproximação de funções para sistemas embarcados descritos anteriormente torna uma solução do problema de aproximação de funções muito complexa o que viabiliza a utilização de algoritmos baseados em computação evolutiva (algoritmos genéticos, programação evolutiva e estratégias evolutivas), para aproximar funções relacionais (pares ordenados da forma entrada – saída). [18]. Para este tipo de problemas, algoritmos genéticos podem encontrar soluções próximas do ótimo.

1.3. Organização da Dissertação

Os objetivos do trabalho dessa dissertação são os seguintes:

- Definir as restrições e limites de aproximação de funções não lineares por funções lineares por partes, considerando-se as incertezas associadas as variáveis.
- Estudo e aplicação de algoritmos genéticos para determinar os pontos de quebra da função aproximada linear por partes para sua implementação em um sistema embarcado com capacidade de processamento em ponto fixo.
- Apresentar aplicações práticas do procedimento proposto para validar a metodologia empregada na determinação das funções não lineares.

Esta dissertação está dividida em seis capítulos, referências bibliográficas e dois apêndices:

- No capítulo 2 é apresentada uma revisão dos procedimentos para aproximar funções não lineares e uma formulação básica sobre este tipo de problema. Analisam-se restrições e limites para aproximação de funções não lineares em sistemas embarcados. Apresenta-se um procedimento para dimensionar uma tabela de equivalência para armazenar os parâmetros da função aproximada em um sistema embarcado.
- No capítulo 3 é apresentada uma revisão sobre computação evolutiva:

algoritmos genéticos, estratégias evolutivas e programação evolutiva.

- No capítulo 4 é apresentada uma metodologia para determinar funções de aproximação utilizando algoritmos genéticos.
- No capítulo 5 são apresentadas aplicações praticas para aproximar funções não lineares em sistemas embarcados.
- No capítulo 6 são apresentadas as conclusões de esta dissertação e são propostas linhas para trabalhos futuros.

Capítulo 2

Aproximação de Funções

Neste capítulo apresenta-se o problema de aproximar uma função não linear para sua implementação em um sistema digital embarcado com restrições de resolução e capacidade de cálculo em ponto fixo. Estudam-se a quantização e a representação das variáveis em outras escalas em um sistema digital. Faz-se uma análise dos efeitos do erro de quantização e como estes são propagados através da função aproximada, em um sistema digital. Apresenta-se um procedimento para determinar os limites de erros de aproximação de forma a garantir a exatidão desejada na geração dos valores da função aproximada. Apresenta-se também um método para aproximar funções mediante funções lineares por partes. De forma a ilustrar o procedimento proposto, mostra-se um estudo de caso da aproximação de uma função seno. Finalmente, analisam-se os diversos métodos de busca para determinar as soluções para aproximar funções não lineares.

2.1. Introdução

Os sistemas de controle são geralmente implementados com processadores digitais de sinais, como por exemplo: microcontroladores, microprocessadores, processadores de propósito geral, processador digital de sinais e hardware personalizado.

Quando em um sistema se usa hardware digital, os números processados podem ser representados em ponto fixo ou em ponto flutuante. Estes dois tipos de dados apresentam tamanhos de palavras fixos com um número determinado de bits. Entretanto, o limite

dinâmico dos valores representados em ponto fixo é muito menor que os valores representados em ponto flutuante com tamanho de palavra (número de bits) equivalente. Conseqüentemente, para evitar *overflow* ou erros de quantização, os valores representados em ponto fixo devem ser representados em escalas diferentes [19].

Se uma representação efetiva dos números do mundo real pode ser feita utilizando hardware em ponto flutuante, então porque utilizar hardware em ponto fixo? quais são as principais vantagens que determinam que um projeto seja realizado utilizando hardware em ponto fixo? estas questões são respondidas a seguir [19].

- *Tamanho e consumo de potência do circuito*; Os sistemas digitais que usam ponto fixo têm menor complexidade que o hardware que usa ponto flutuante. Conseqüentemente, necessita-se de uma pastilha de circuito integrado menor para realizar um hardware com ponto fixo, com um consumo menor de potência comparado com o hardware necessário para implementar o sistema com ponto flutuante.
- *Memória utilizada e velocidade*; em geral, os cálculos realizados em ponto fixo requerem memória e tempo de processamento menores.
- *Custo*; utilizar hardware com processamento com ponto fixo é mais conveniente quando se tem restrição de preço do produto, pois, como a área de semicondutor usada é menor para o sistema que usa ponto fixo, o custo também é menor.

Considerando-se essas questões, propõe-se nessa dissertação abordar os sistemas digitais embarcados de baixo custo, que utilizem, por exemplo, microcontroladores, FPGA, ASIC, entre outras, com capacidade de processamento em ponto fixo e resolução limitada.

Para estes sistemas, muitas vezes, é necessário gerar valores aproximados de funções que não são disponíveis ou não podem ser gerados diretamente. Tendo em vista a utilização de sistemas embarcados de baixo custo, com cálculo de processamento em ponto fixo e velocidade de cálculo relativamente baixa, a utilização de tabelas de equivalência (LUT – *Look-up Table*), com interpolação linear por partes é uma opção interessante para esse tipo de problema, comparada com outros tipos de interpolação, tais como *splines* e polinomial, que exigem maior poder computacional.

2.2. Formulação do Problema de Aproximação de Funções

Considere-se um sistema digital embarcado em que se necessita gerar valores de uma função não linear, utilizando aproximação de funções lineares por partes. Utiliza-se um

sistema embarcado de baixo custo, em que os cálculos sejam realizados em ponto fixo (*fixed-point*), com resolução limitada e com limitações de espaço de memória devido às restrições da arquitetura do projeto.

A função não linear é definida por $y = f(x)$, $x \in [x_{min}, x_{max}]$ e $y \in [y_{min}, y_{max}]$, em que cada elemento do conjunto de valores de entrada x é quantizado com uma resolução equivalente a N bits e os elementos do conjunto de valores de y são gerados com uma exatidão equivalente a uma resolução de N_y bits. Supõe-se que os valores de x e y são positivos e seus valores mínimos são zeros, $x_{min} = 0$ e $y_{min} = 0$, que podem ser obtidos adicionando-se constantes à função f . Essa constante pode ser subtraída após a geração dos valores de y [4].

A função não linear é aproximada por uma função linear por partes considerando que os cálculos podem ser realizados com uma precisão equivalente a N_T bits de resolução, com $N_T \geq N_y$. Os valores dos pontos de quebra para reproduzir a função aproximada são armazenados em uma tabela de equivalência (LUT - *Look-up Table*) embutido em um sistema embarcado, como é mostrado na figura 2.1.

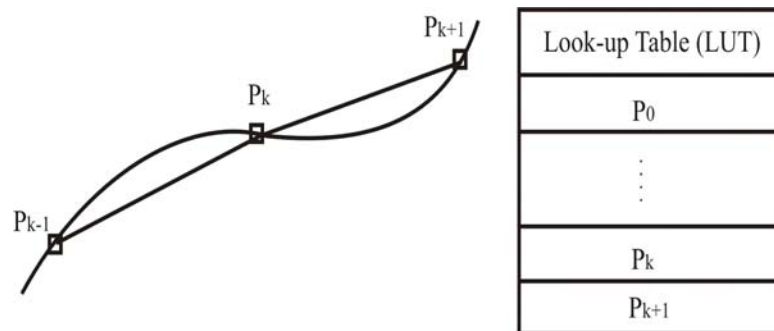


Figura 2.1. Aproximação linear por partes de uma função não linear.

Na utilização de LUTs para aproximação de funções não lineares, aborda-se geralmente o problema de minimização do erro de aproximação, sem levar em consideração o número de células necessárias para a construção da LUT, sendo esse um inconveniente para sistemas embarcados com limitações de espaço de memória. Uma alternativa é desenvolver um algoritmo que encontre o número mínimo de pontos de quebra e resolução mínima de armazenamento na LUT para aproximação linear por partes da função não linear, considerando as restrições de limites de aproximação e garantindo a exatidão desejada na geração dos valores da função aproximada estabelecidas na fase do projeto, de forma a minimizar o tamanho de memória utilizada para a LUT [15].

2.3. Quantização, representação das variáveis e representação das variáveis em escalas diferentes em um sistema digital embarcado

Em um sistema digital embarcado com capacidade de cálculo em ponto fixo, e resolução limitada, as variáveis numéricas podem ser representadas assumindo valores inteiros, assim também a quantização dos números depende da resolução do conversor analógico digital. As variáveis numéricas também podem ser representadas em outras escalas de forma a se obter o máximo de resolução de representação, considerando o erro máximo devido à quantização.

Nesta dissertação foi adotada uma notação para representar as variáveis em escala real, variáveis em outra escala e para as aproximações:

- Variáveis em minúsculo representam valores na escala real, exemplo: x, y ;
- Variáveis em maiúsculo representam valores em outras escalas, por exemplo: X, Y ;
- Variáveis com sobrescrito ‘ \sim ’ representam aproximações;

2.3.1. Quantização das variáveis

O processo de quantização usa uma função não linear em que uma variável de valores contínuos é representada por um conjunto de valores discretos. Cada elemento do segundo conjunto representa um intervalo de variação do primeiro. Na quantização uniforme, para uma quantização de n elementos, a faixa total de variação da variável contínua é dividida em n intervalos iguais e cada intervalo é representado por um único número. Para uma resolução de N bits, tem-se $n = 2^N$ intervalos de quantização. A quantização da variável y em N bits é definida por:

$$\tilde{y} = Q_N(y) \quad (2.1)$$

O intervalo de quantização pode ser calculado por:

$$\Delta = \frac{y_{\max} - y_{\min}}{2^N} \quad (2.2)$$

Quando se usa o valor de saída no meio da faixa dos valores de entrada, o erro de quantização pode ter valores positivos e negativos e tem os menores valores absolutos máximos. Ou seja, nesse caso, cada intervalo da faixa total de variação da variável contínua é representado por um valor posicionado no meio do intervalo. Dessa forma, os valores de quantização de y , podem ser calculados por:

$$\tilde{y} = \Delta i + \frac{\Delta}{2}; i = 0, \dots, (2^N - 1) \quad (2.3)$$

A quantização no meio da faixa em N bits de y pode ser calculada por:

$$\tilde{y} = Q_N(y) = \tilde{y}_i \mid \Delta i \leq y < \Delta(i+1) \quad (2.4)$$

2.3.2. Representação de variáveis

A variável $x \in [x_{min}, x_{max}]$ pode ser considerada com representação inteira uma vez que ela é quantizada com uma resolução equivalente a N bits, assumindo valores inteiros $X = 0, 1, 2, \dots, 2^N - 1$, correspondente aos números binários 00...0 até 11...1. A transformação da variável de inteiro para o valor real de x pode ser realizada por:

$$x = \frac{X + 0,5}{2^N} (x_{max} - x_{min}) \quad (2.5)$$

Soma-se 0,5 porque se considera que os valores de X inteiros são convertidos no meio da faixa.

2.3.3. Representação das variáveis em outras escalas

Os valores de $y \in [y_{min}, y_{max}]$, também podem ser representados em outras escalas, entretanto, considera-se a resolução N_T bits. Embora a resolução desejada de geração de valores de y seja N_Y , considera-se N_T bits porque os cálculos no sistema embarcado são realizados nessa resolução. Além disso, a resolução de N_Y é considerada para avaliação final de y devido às perdas na aproximação e na propagação da incerteza associada na geração de x , se houver alguma. O valor verdadeiro de y é calculado por:

$$y = f(x) \quad (2.6)$$

Considerando o valor máximo de y para toda a faixa do intervalo, a representação de y em outra escala (N_T bits), pode ser calculado por:

$$Y = \frac{2^{N_T}}{y_{max} - y_{min}} y - 0,5 \quad (2.7)$$

A quantização pode ser realizada diretamente para os valores de Y com a vantagem de gerar números inteiros. A representação do valor verdadeiro por um número inteiro em outra escala de resolução N_T pode ser obtida por:

$$\tilde{Y} = \text{round}(Y) = \text{floor}\left(\frac{2^{N_T}}{(y_{\max} - y_{\min})} y\right) \quad (2.8)$$

Os valores quantizados de y podem ser calculados a partir de (2.8) por:

$$\tilde{y} = \frac{\tilde{Y} + 0,5}{2^{N_T}} (y_{\max} - y_{\min}) \quad (2.9)$$

Os valores quantizados de \tilde{Y} e \tilde{y} são os valores que melhor representam Y e y na escala de resolução de N_T bits. Considera-se saturação para o valor máximo $2^{N_T} - 1$ quando ocorre estouro.

2.4. Análise da propagação da incerteza e erro de aproximação em um sistema digital embarcado

Para avaliar o efeito do erro de quantização das variáveis em um sistema digital embarcado, faz-se uma análise da propagação dos erros de quantização quando a função não linear aproximada é implementada em um sistema digital com capacidade de cálculo em ponto fixo.

Cada elemento do conjunto de valores de $y \in [y_{\min}, y_{\max}]$ é gerado a uma resolução equivalente a N_Y bits desconsiderando-se a forma como os valores de y são gerados, tem-se que o valor máximo da incerteza associada $\hat{\varepsilon}_y$, para todos os valores de y devem ser:

$$\hat{\varepsilon}_y = \frac{y_{\max} - y_{\min}}{2^{N_Y+1}} \quad (2.10)$$

No sistema digital embarcado, os valores da variável independente x possuem uma incerteza associados a sua geração. Conseqüentemente, define-se $\tilde{x} = \{\tilde{x}_1, \dots, \tilde{x}_n\}$ como os n possíveis valores aproximados de x e $\varepsilon_x = \{\varepsilon_{x_1}, \dots, \varepsilon_{x_n}\}$ como as incertezas associadas, tal que os valores aproximados são calculados por: $\tilde{x} = x + \varepsilon_x$.

A incerteza associada à variável independente x , ε_x , pode ser originada de diversas formas: devido à quantização em um conversor analógico digital, arredondamento ou truncamento de cálculo, precisão de um codificador ou de um relógio, etc. No caso de um conversor analógico digital ideal, os valores dos limites das incertezas são:

$$\varepsilon_{x_{\max}} = \frac{q}{2} \text{ e } \varepsilon_{x_{\min}} = -\frac{q}{2} \quad (2.11)$$

sendo q o passo de quantização.

Conseqüentemente, os valores da incerteza associada são limitados por: $\varepsilon_{x_i} \in [\varepsilon_{x_i, \min}, \varepsilon_{x_i, \max}]$, para $i = 1, \dots, 2^N$, em que normalmente são constantes para todo o conjunto.

$$\varepsilon_{x_i, \max} = \frac{x_{\max} - x_{\min}}{2^{N+1}} \text{ e } \varepsilon_{x_i, \min} = -\frac{x_{\max} - x_{\min}}{2^{N+1}} \quad (2.12)$$

Uma solução para o problema de aproximar a função não linear f em um sistema digital embarcado por funções lineares por partes é definida como $s = \tilde{f}(\tilde{x})$, em que s é uma aproximação de $y = f(x)$. A função aproximada s apresenta uma incerteza associada causada pela propagação da incerteza associada à variável livre ε_x , que é propagada através da função aproximada s . Conseqüentemente, deve-se garantir que a incerteza associada na geração dos valores da função aproximada s em relação à função não linear y esteja dentro da especificação de exatidão do projeto [4], ou seja:

$$|\varepsilon_s| \leq \hat{\varepsilon}_y \quad (2.13)$$

em que ε_s é a incerteza (ou erro) de geração da função aproximada s , definida como:

$$\varepsilon_s = y - s \quad (2.14)$$

A seguir, apresenta-se uma análise do erro de aproximação, sendo considerados dois casos de estudo. No primeiro caso, analisa-se o erro de aproximação considerando a variável independente x como exata. No segundo caso, analisa-se o erro de aproximação considerando que a variável independente x possui uma incerteza associada a sua geração, devido à conversão analógico-digital.

2.4.1. Análise do erro de aproximação considerando a variável independente x como exata

Determinam-se os limites do erro de aproximação considerando a variável independente x como exata. Neste sentido o erro de aproximação para todo o conjunto de valores de x é dado por:

$$\varepsilon_{A_i} = \tilde{f}(x_i) - f(x_i) \quad (2.15)$$

O erro de aproximação é definido por ε_A , que é limitado pelos valores máximos $\varepsilon_{A_{\max}}$ e mínimos $\varepsilon_{A_{\min}}$, que são constantes para o conjunto completo de elementos x :

$$\varepsilon_{Amin} \leq \varepsilon_A \leq \varepsilon_{Amax} \quad (2.16)$$

$$\varepsilon_{Amax} = \frac{(y_{max} - y_{min})}{2^{N_Y+1}} \text{ e } \varepsilon_{Amin} = -\frac{(y_{max} - y_{min})}{2^{N_Y+1}} \quad (2.17)$$

Os valores da função aproximada s , qualquer que seja, linear, quadrática, etc, devem ser calculados de forma quantizada na resolução disponível para o processador do sistema embarcado, com N_T bits.

$$s = \tilde{f}(x) \quad (2.18)$$

O cálculo dos valores quantizados da função aproximada S , na escala de resolução de N_T bits, pode ser encontrado utilizando as equações (2.7) e (2.8):

$$S = \frac{2^{N_T}}{(y_{max} - y_{min})} y - 0.5 \quad (2.19)$$

$$\tilde{S} = \text{round}(S) \quad (2.20)$$

Devido à quantização da função aproximada em outra escala de resolução, N_T bits, é necessário representar os valores do erro de aproximação ε_A , determinado pela equação (2.17), na escala de N_T bits de resolução, sendo determinado por:

$$E_{Amax} = \frac{2^{N_T}}{y_{max} - y_{min}} \varepsilon_{Amax} \text{ e } E_{Amin} = \frac{2^{N_T}}{y_{max} - y_{min}} \varepsilon_{Amin} \quad (2.21)$$

Como os cálculos no processamento dos sinais do sistema embarcado é realizado com uma resolução equivalente de N_T bits, os valores dos pontos de quebra da função não linear aproximada por funções lineares por partes, devem ser calculados e armazenados na tabela de equivalência na escala de N_T bits de resolução.

2.4.2. Análise do erro de aproximação considerando que a variável independente x tem associada uma incerteza devido à conversão digital

Considere-se que a incerteza associada à função aproximada $s = \tilde{f}(x)$ é limitada por: $\varepsilon_{s_i} \in [\varepsilon_{s_i, min}, \varepsilon_{s_i, max}]$, para $i = 1, \dots, 2^N$ e pode ser dividida em duas componentes: $\varepsilon_{s_i} = \varepsilon_{p_i} + \varepsilon_{A_i}$, sendo que ε_p se deve à propagação da incerteza ε_x através da função aproximada, e ε_A é o erro de aproximação de $f(\cdot)$ por $\tilde{f}(\cdot)$. Dessa forma os limites de incerteza ε_s podem ser calculados

a partir dos limites de suas componentes, que são definidos como: $\varepsilon_{p_i} \in [\varepsilon_{p_i, \min}, \varepsilon_{p_i, \max}]$ e $\varepsilon_{A_i} \in [\varepsilon_{A_i, \min}, \varepsilon_{A_i, \max}]$.

Para o cálculo dos limites da componente ε_p verifica-se a propagação, nos piores casos, dos valores de ε_x , através da função aproximada. Para tal, pode-se definir:

$$\lambda_{1i} = \tilde{f}(x_i + \varepsilon_{x_i, \max}) - \tilde{f}(x_i) \text{ e } \lambda_{2i} = \tilde{f}(x_i + \varepsilon_{x_i, \min}) - \tilde{f}(x_i) \quad (2.22)$$

Inicialmente não se dispõe da função aproximada $s = \tilde{f}(x)$. Entretanto, a equação (2.22) pode ser aproximada para utilizar a função não aproximada $y = f(x)$:

$$\lambda_{1i} = f(x_i + \varepsilon_{x_i, \max}) - f(x_i) \text{ e } \lambda_{2i} = f(x_i + \varepsilon_{x_i, \min}) - f(x_i) \quad (2.23)$$

Além disso, para os casos em que os valores de $\varepsilon_{x, \max}$ e de $\varepsilon_{x, \min}$ são pequenos para toda a faixa de variações de x e que a função f é diferenciável, pode-se fazer a seguinte aproximação:

$$\lambda_{1i} \approx -\lambda_{2i} \approx f'(x_i) \varepsilon_{x_i, \max} \quad (2.24)$$

Os limites da componente originada pela propagação da incerteza ε_x podem ser calculados por:

$$\varepsilon_{p_i, \max} = \max(\lambda_{1i}, \lambda_{2i}) \text{ e } \varepsilon_{p_i, \min} = \min(\lambda_{1i}, \lambda_{2i}) \quad (2.25)$$

O erro de aproximação ε_A pode ser calculado para todo o conjunto de valores de x por:

$$\varepsilon_{A_i} = \tilde{f}(x_i) - f(x_i) \quad (2.26)$$

Finalmente, os limites da incerteza ε_s podem ser calculados por:

$$\varepsilon_{s_i, \max} = \varepsilon_{p_i, \max} + \varepsilon_{A_i} \text{ e } \varepsilon_{s_i, \min} = \varepsilon_{p_i, \min} + \varepsilon_{A_i} \quad (2.27)$$

Definindo-se o erro máximo aceitável de geração dos valores de s em (2.10), pode-se definir a faixa aceitável do erro de aproximação $\varepsilon_{A, \min}$ e $\varepsilon_{A, \max}$ a partir de (2.13) e de (2.27), por:

$$|\varepsilon_s| \leq \widehat{\varepsilon}_y \quad (2.28)$$

$$-\widehat{\varepsilon}_y \leq \varepsilon_s \leq \widehat{\varepsilon}_y \quad (2.29)$$

$$-\widehat{\varepsilon}_y \leq \varepsilon_p + \varepsilon_A \leq \widehat{\varepsilon}_y \quad (2.30)$$

$$-\widehat{\varepsilon}_y = \varepsilon_{p_i, \min} + \varepsilon_{A_i, \min} \text{ e } \varepsilon_{p_i, \max} + \varepsilon_{A_i, \max} = \widehat{\varepsilon}_y \quad (2.31)$$

$$\varepsilon_{A_i, \min} = -\widehat{\varepsilon}_y - \varepsilon_{p_i, \min} \text{ e } \varepsilon_{A_i, \max} = \widehat{\varepsilon}_y - \varepsilon_{p_i, \max} \quad (2.32)$$

Ainda, deve-se garantir que a componente originada da propagação de incerteza em x seja sempre menor que o erro máximo aceitável de geração de s , ou seja:

$$\left| \varepsilon_{p_i, \min} \right| < \widehat{\varepsilon}_y \text{ e } \left| \varepsilon_{p_i, \max} \right| < \widehat{\varepsilon}_y \quad (2.33)$$

Os valores calculados a partir de (2.32) podem então ser utilizados para verificar se a resolução dos valores gerados, utilizando a função aproximada, corresponde à resolução mínima aceitável [4].

2.5. Normalização da função aproximada

Uma normalização da função aproximada pode ser feita para facilitar a interpretação da aproximação com relação aos limites de erros de aproximação. Este processo de normalização subtrai de todas as variáveis os valores da função quantizada y_q a N_T bits de resolução [20]. Pode-se então usar a seguinte transformação para normalizar a função desconsiderando sua forma:

$$S = s - y_q ; Y = y - y_q ; E_{A_{\min}} = \varepsilon_{A_{\min}} + Y ; E_{A_{\max}} = \varepsilon_{A_{\max}} + Y ; P_y = p_y - y_q \text{ e } P_x = p_x \quad (2.34)$$

Com esta transformação, deslocam-se as variáveis para em torno de zero, desta maneira elas assumem valores inteiros $0, \pm 1, \pm 2, \dots$ e $\varepsilon_A = Y - S$.

2.6. Aproximação de Funções Não Lineares utilizando Funções Lineares por Partes

Uma tabela de equivalência (LUT - *Look-up Table*) é um tipo de bloco lógico que contém células de armazenamento que são utilizadas para implementação de pequenas funções. Cada célula é capaz de armazenar um único valor, assim LUT's de vários tamanhos podem ser criados e implementados em um microcontrolador ou processador.

Funções não lineares podem ser geradas a partir de tabelas de equivalência, utilizando interpolação linear por partes, sendo uma das técnicas mais usadas, que comparadas com outros tipos de interpolação, tais como *splines* e polinomial, exigem menor poder computacional [9, 15].

A função aproximada é construída por partes a partir de m pontos de quebra (pontos comuns de dois segmentos lineares diferentes), $p_x = \{p_{x1}, \dots, p_{xm}\}$, $p_x \subseteq x$ e $p_y = \{p_{y1}, \dots, p_{ym}\}$, $p_y \subseteq y$, $m \leq n$, que são armazenados em uma tabela de equivalência [21].

Os pontos de quebra (p_x, p_y) , são armazenados com uma resolução equivalente de N_T bits. A função de aproximação $s = \tilde{f}(x)$ é composta por m funções lineares \tilde{f}_j definidas nos intervalos $[p_{xj}, p_{xj+1}]$, com $j = 1, \dots, m$, $p_{x1} = x_1$ e $p_{xm} = x_n$ é definida em geral como:

$$s = \tilde{f}(x) = a_j + b_j(x - p_{xj}) \quad (2.35)$$

sendo os valores dos coeficientes a_j e b_j , encontrados com:

$$a_j = p_{yj} \text{ e } b_j = \frac{p_{yj+1} - p_{yj}}{p_{xj+1} - p_{xj}} \quad (2.36)$$

para:

$$p_{xj} \leq x < p_{xj+1} \text{ e } \tilde{f}_{m+1}(x_n) = \tilde{f}_m(x_n) \quad (2.37)$$

2.7. Dimensionamento da tabela de equivalência (LUT - Look-up Table)

Em um sistema digital embarcado, a função aproximada é gerada a partir dos pontos de quebra armazenados na LUT. Supondo que não existe perda de resolução no cálculo dos valores de y , a resolução mínima de armazenamento N_T bits, na tabela de equivalência deve ser tal que o erro de quantização proporcionado (devido à resolução da tabela) seja igual ao mínimo valor do erro de aproximação ε_A , ou seja:

$$N_T \geq \log_2 \left(\frac{y_{\max} - y_{\min}}{2 \min(\{\varepsilon_{A_{\max}}, -\varepsilon_{A_{\min}}\})} \right) \quad (2.38)$$

em que $\{\varepsilon_{A_{\max}}, -\varepsilon_{A_{\min}}\}$ é a concatenação dos dois vetores.

O procedimento para o dimensionamento da tabela de equivalência (LUT) deve seguir as seguintes etapas:

- Definir a resolução mínima de geração dos valores da variável dependente, N_Y ;
- Calcular o valor da componente devida à propagação da incerteza ε_x usando (2.23) e (2.25);
- Calcular os valores limites do erro de aproximação ε_A usado (2.32) e garantindo (2.33). Caso contrário, deve se modificar a resolução desejada de

geração de y ou diminuir a incerteza associada à x ;

- Definir a resolução de armazenamento da tabela, N_T , usando (2.38);
- Utilizar um algoritmo específico para procurar os pontos de quebra da função aproximada.

O algoritmo para dimensionamento da tabela deve procurar os pontos de quebra de tal forma que o erro de aproximação para cada valor de x esteja sempre contido dentro dos limites aceitáveis, ou seja: $\varepsilon_{A_i} \in [\varepsilon_{A_i, \min}, \varepsilon_{A_i, \max}]$ [20].

2.8. Estudo de caso: Aproximação da Função Seno

Considere-se o caso de estudo da geração dos valores da função seno no primeiro quadrante, para a análise da propagação da incerteza e do erro de aproximação. Para a função seno $y = \text{sen}(x)$, por ser simétrica, é necessário aproximar apenas um quarto de função, $x \in [0, \pi/2]$ e $y \in [0, 1]$; os demais valores podem ser encontrados a partir destes.

Considere-se que os valores do ângulo x são quantizados em quatro bits ($N = 4$) e a incerteza associada é de $\frac{1}{2}$ LSB que é constante para todo o conjunto de x :

$$\varepsilon_{x_{\max}} = \frac{x_{\max} - x_{\min}}{2^{N+1}} = \frac{\pi/2}{2^5} = 0,049 \quad (2.39)$$

Na Tabela 2.1 apresentam-se os valores binários de x e o ângulo equivalente calculado utilizando a primeira transformação da equação (2.5):

Tabela 2.1. Valores de x e ângulo equivalente com duas casas decimais.

X	0	1	2	3	4	5	6	7
x	0,049	0,147	0,245	0,344	0,442	0,540	0,638	0,736
X	8	9	10	11	12	13	14	15
x	0,834	0,932	1,031	1,129	1,227	1,325	1,423	1,522

Na análise da propagação da incerteza e erros de aproximação na geração da função senoidal são analisados dois casos, o primeiro considerando a variável livre como exata ($N = N_Y$) e o segundo considerando uma incerteza associada a sua geração ($N \neq N_Y$).

2.8.1 Análise do erro de aproximação da função seno considerando a variável independente x como exata ($N = N_Y$)

Deseja-se gerar os valores aproximados de $y = \text{sen}(x)$ considerando a variável independente x como exata, a exatidão de geração é equivalente à resolução de x , ou seja, $N_Y = N = 4$ bits. Para uma mudança de escala da função aproximada com $N_T = 5$ bits, os limites do erro de aproximação são equivalentes a um erro de quantização de 4 bits:

$$\varepsilon_{A_{\max}} = -\varepsilon_{A_{\min}} = \frac{y_{\max} - y_{\min}}{2^{N_Y+1}} = \frac{1}{2^5} = 0,03125 \quad (2.40)$$

Neste caso os limites de erro de aproximação escalonados são calculados por (2.21):

$$E_{A_{\max}} = 1 \text{ e } E_{A_{\min}} = -1 \quad (2.41)$$

Considerem-se os pontos de quebra $p_x = [0 \ 6 \ 12 \ 15]$ e $p_y = [1 \ 19 \ 30 \ 31]$, na Figura 2.2, apresentam-se os valores dos pontos de quebra, a função aproximada \tilde{f} e os limites da função aproximada, em função dos valores de X .

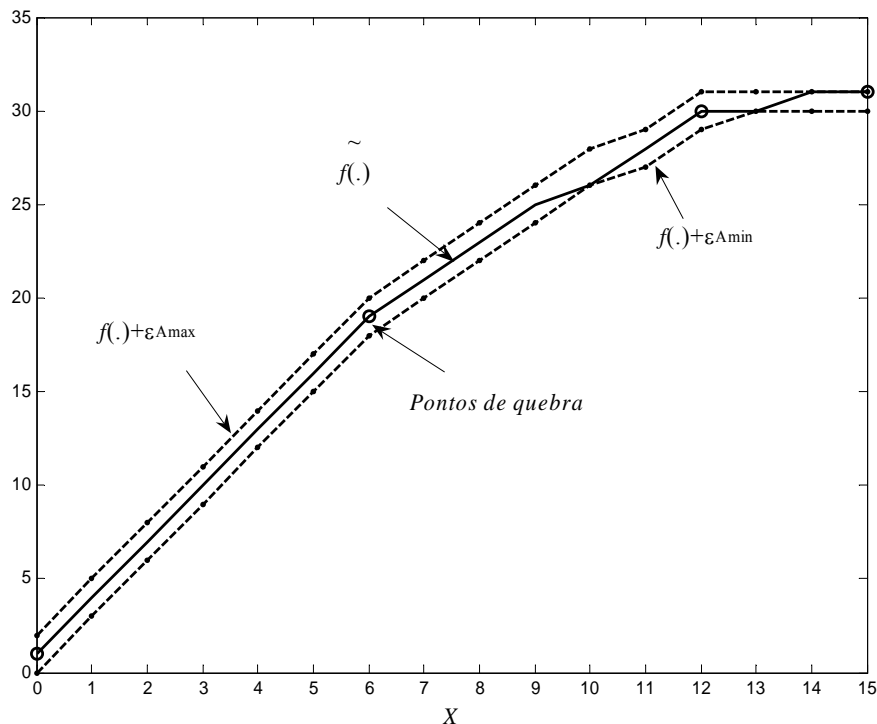


Figura 2.2. Valores aproximados de y e limites de aproximação.

Na Figura 2.3 apresenta-se a representação do problema de aproximar a função utilizando a normalização definida pela equação (2.34).

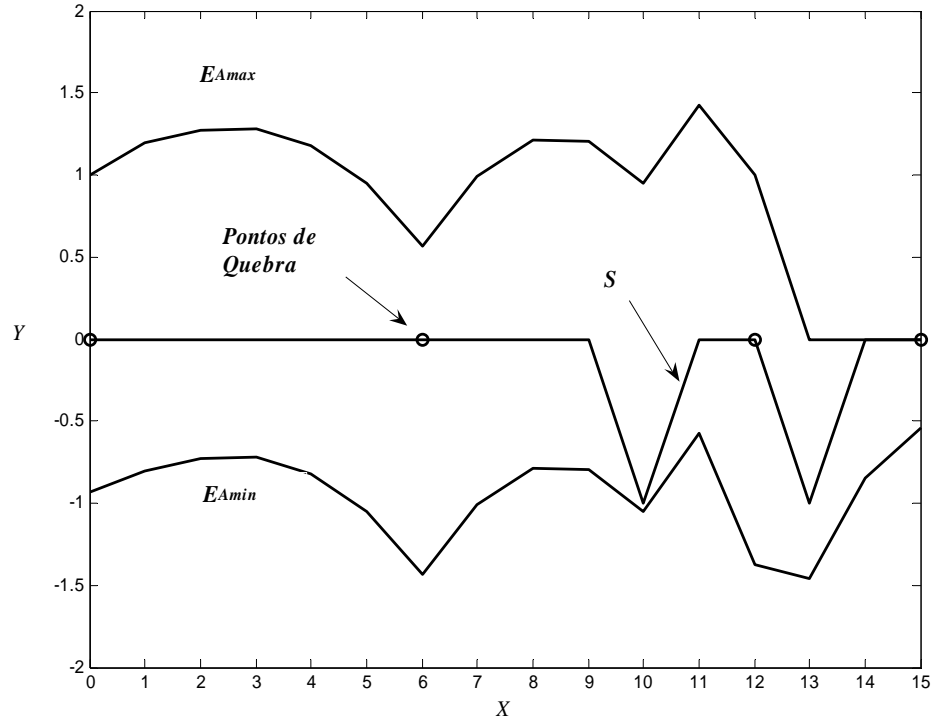


Figura 2.3. Erro de aproximação S e limites de erro de aproximação E_{Amin} e E_{Amax} normalizados.

2.8.2 Análise do erro de aproximação da função seno considerando a variável independente x associada a uma incerteza ($N \neq N_Y$)

Neste caso a variável independente x possui uma incerteza associada devido à conversão analógico-digital. Deseja-se gerar os valores aproximados de $y = \text{sen}(x)$, com resolução de geração de três bits ($N_Y = 3$), e dessa forma tem-se, a partir de (2.10):

$$\hat{\varepsilon}_y = \frac{(y_{\max} - y_{\min})}{2^{N_Y+1}} = \frac{1}{2^4} = 0,0625 \quad (2.42)$$

A função de geração a ser aproximada, devido à quantização, pode ser calculada por:

$$y = \text{sen}\left(\left(x_{\max} - x_{\min}\right) \frac{(\tilde{x} + 0,5)}{2^N}\right) = \text{sen}\left(\frac{\pi (\tilde{x} + 0,5)}{2 \cdot 16}\right) \quad (2.43)$$

Os limites da incerteza propagada e os limites do erro de aproximação ε_{Amax} , ε_{Amin} , ε_{pmax} , ε_{pmin} , são calculados a partir de (2.25) e (2.32)

$$\begin{aligned}
\lambda_{1i} &\approx -\lambda_{2i} \approx f'(x_i)\varepsilon_{x_i \max} = \varepsilon_{x_i \max} * \cos(x_i) \\
\varepsilon_{p_i \max} &= \max(\lambda_{1i}, \lambda_{2i}) \\
\varepsilon_{p_i \min} &= \min(\lambda_{1i}, \lambda_{2i}) \\
\varepsilon_{A_i \max} &= -\widehat{\varepsilon}_y - \varepsilon_{p_i \max} \\
\varepsilon_{A_i \min} &= -\widehat{\varepsilon}_y - \varepsilon_{p_i \min}
\end{aligned}
\tag{2.44}$$

A partir da Figura 2.4 pode-se observar como a incerteza inicial, constante para todos os valores de x , é modificada pela função não linear.

Na Figura 2.5 apresentam-se a função ideal, os limites da função aproximada e os limites de geração de y , em função dos valores de x .

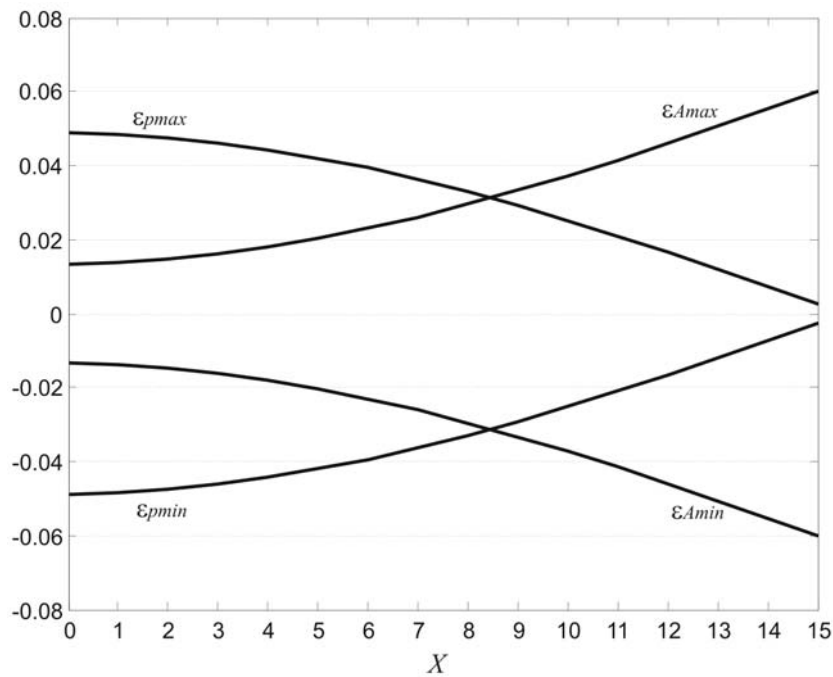


Figura 2.4. Limites de incerteza propagada e do erro de aproximação

A resolução mínima de armazenamento da tabela de equivalência é calculada a partir da equação (2.38) para $N_T \geq 5,22$ bits.

Nas Figuras 2.6 e 2.7 apresentam-se os erros de aproximação para duas soluções diferentes. Os locais dos pontos de quebra que são armazenados na tabela são representados por x sobre o eixo.

Pode-se observar que para um valor de N_T suficientemente grande, o erro de aproximação apresenta uma forma bem comportada, que no caso é a diferença da reta aproximada e a curva ideal. Entretanto, quando a resolução de armazenamento na tabela é reduzida, essa característica deixa de existir devido à quantização dos coeficientes e resolução limitada do cálculo da função aproximada.

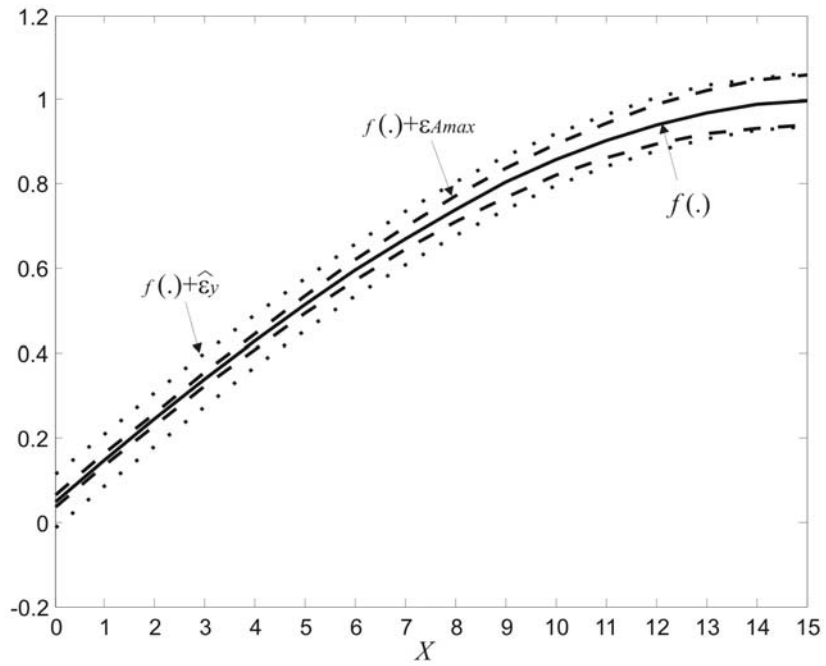


Figura 2.5. Função ideal, limites de geração dos valores de y e limites da função aproximada

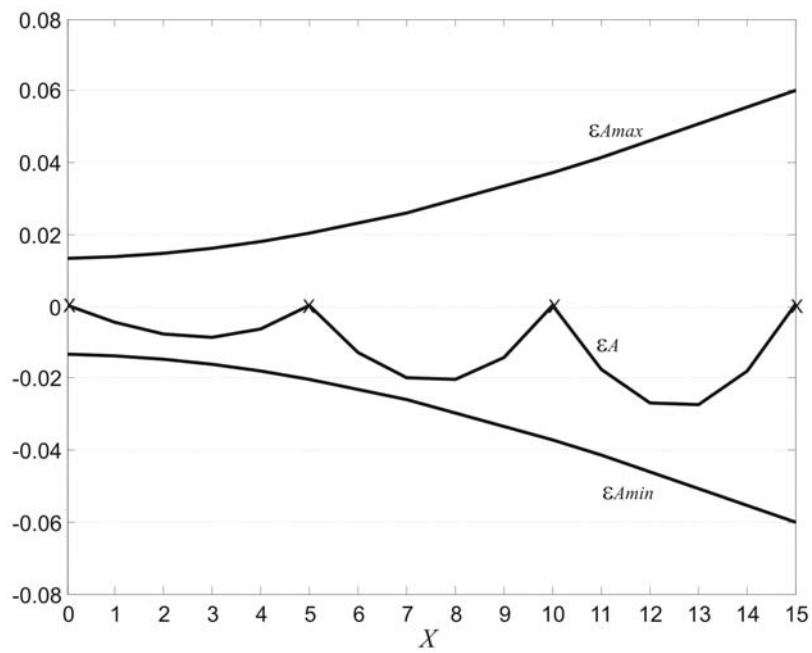


Figura 2.6. Erro de aproximação para $m = 4$ e $N_T = 10$, com coeficientes calculados utilizando pontos sobre a curva ideal

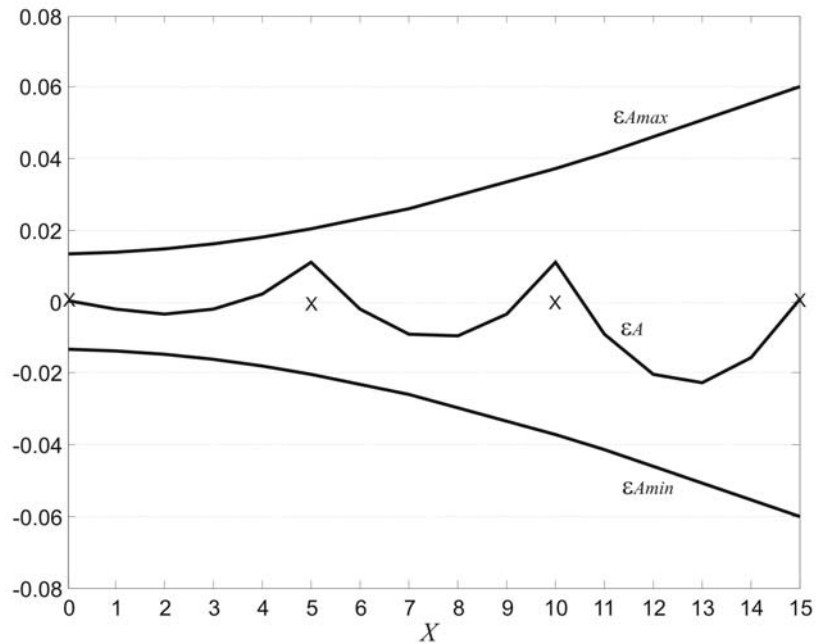


Figura 2.7. Erro de aproximação para $m = 4$ e $N_T = 10$, com coeficientes calculados utilizando pontos fora da curva ideal.

2.9. Algoritmos de Busca

Os principais métodos de busca apresentados na bibliografia são de natureza determinísticas, enumerativas e estocásticas.

Os métodos determinísticos geralmente fazem uso do cálculo de derivadas e necessitam de algum tipo de informação do gradiente, seja procurando o ponto em que a função derivada se anula, usando a direção para a qual aponta, ou fazendo aproximações de derivadas. A procura pelo ótimo, com derivadas, usa o ponto corrente como ponto de partida para a próxima iteração. Logo, a busca é local, porque ocorre na vizinhança do ponto corrente. Assim, quando esses algoritmos encontram soluções, há grande chance de ser um ótimo local. Um problema durante a implementação dos algoritmos de busca aparece quando a função a ser tratada não é contínua ou tem derivada complicada, como no caso de funções quantizadas. Esses métodos possuem grande rapidez e funcionam muito bem para problemas unimodais contínuos.

Nos métodos enumerativos, o algoritmo verifica todas as combinações possíveis de soluções, o que pode torná-lo inviável para regiões muito grandes e, conseqüentemente, diminuir a sua eficácia.

Os métodos estocásticos buscam a solução utilizando regras probabilísticas, métodos

de agrupamento (*clustering*), método recozido simulado (*simuling annealing*) e métodos axiomáticos. Dessa forma, a busca não é feita somente na vizinhança e, com isso, a chance de se encontrar um ótimo global aumenta. Algoritmos genéticos, quando são utilizados como funções de otimização, podem ser classificados como métodos estocásticos. Algoritmos genéticos diferem de outras técnicas de busca e otimização porque eles não usam diretamente uma variável previamente projetada; entretanto, eles manipulam a variável projetada codificando-a e utilizam somente valores da função objetivo para determinar a direção de busca. Assim também, algoritmos genéticos processam simultaneamente vários pontos em paralelo, tendo maior chance de convergir para um subótimo local que é usualmente o mínimo.

Comparando-se os métodos, observa-se que, se o tempo computacional e o domínio do problema não forem restrições do problema, os métodos enumerativos são a melhor opção, pois encontrarão a solução global. Se a solução do problema puder ser qualquer solução realizável, e ainda, a rapidez de convergência e precisão da resposta forem itens importantes, então o método adequado deve pertencer ao grupo determinístico. Sendo o problema complexo (muitas variáveis, descontínuo ou de difícil diferenciação) e necessitando-se da solução global em um tempo computacional razoável, as técnicas estocásticas são as mais indicadas. Sendo assim, os métodos estocásticos devem ser entendidos como um conjunto de técnicas e procedimentos genéricos e adaptáveis, a serem aplicados na solução de problemas complexos, para os quais outras técnicas conhecidas são ineficazes ou nem sequer são aplicáveis.

2.9.1 Algoritmo de Busca Enumerativo

Uma solução ao problema de aproximar uma função $y = f(x)$ utilizando o método enumerativo é combinar as posições dos pontos de quebra verificando que os valores estimados da grandeza y estejam sempre contidos dentro dos limites de erro de aproximação.

Um algoritmo pode ser implementado de forma a procurar os pontos de quebra da função de aproximação $y = f(x)$ usando apenas valores positivos $x_{min} = 0$ e $y_{min} = 0$. Considera-se que os pontos de quebra são armazenados na tabela de equivalência com uma resolução de N_T bits. Considera-se também que o sinal analógico é condicionado de forma a ocupar toda a faixa de entrada do conversor A/D.

Define-se então os pontos de quebra (p_x, p_y) , em que p_x são os valores equivalentes de x , quantizados na resolução N bits $x : 0 \leq p_x \leq 2^N - 1$ e p_y são os valores de y quantizados na resolução N_T bits $y : 0 \leq p_y \leq 2^{N_T} - 1$. A função aproximada realiza interpolação linear a partir

dos pontos de quebra armazenados na tabela de equivalência, utilizando aritmética de ponto fixo.

Considera-se a aproximação de uma função não linear utilizando aproximação de funções lineares por partes, com m pontos de quebra (p_x, p_y) , considerando que a variável livre x é exata ($N = N_Y$), distribuímos os pontos de quebra em uma região: $(E_{Amax} - E_{Amin}) \times 2^N$ pontos. Encontramos o número aproximado de combinações totais dos pontos de quebra, sendo algumas destas combinações soluções que cumprem as restrições que a função aproximada esteja contida dentro dos limites de erro de aproximação.

$$Combinações \approx r^2 \left(\frac{p!}{(p-m)!m!} \right)^r \quad (2.45)$$

sendo:

m = número de pontos de quebra para aproximar a função por partes, sem considerar os pontos extremos $(0, 2^N - 1)$.

$p = 2^N$ número de pontos do intervalo.

Se $N_Y = N \rightarrow r = 2^{(N_T - N_Y)}$ considerado constante em todo o intervalo.

De forma a aplicar esta última expressão (2.45), determina-se o número de combinações para o exemplo analisado na seção anterior, a aproximação da função seno no primeiro quadrante considerando a variável independente como exata, com $N = N_Y = 4$, $N_T = 5$ e quatro pontos de quebra ($m = 2$). O número total de combinações é calculado por:

$$Combinações = 2^2 \left(\frac{16!}{(16-2)!2!} \right)^2 = 57600 \quad (2.46)$$

Neste caso, o método enumerativo dá uma solução global, mas o tempo computacional é maior quando as especificações do problema aumentam. Este método é inviável para regiões muito grandes, conseqüentemente, prejudicando a sua eficiência.

Capítulo 3

Computação Evolutiva

Neste capítulo apresentam-se as aplicações e princípios básicos de computação evolutiva e suas implementações em algoritmos genéticos, estratégias evolutivas e programação evolutiva. Analisa-se a importância das técnicas evolutivas como métodos de busca em espaços complexos. Apresentam-se as estruturas das técnicas evolutivas com ênfase em algoritmos genéticos, descrevendo suas formas de representação, métodos de seleção e os diferentes operadores genéticos utilizados. Finalmente, são detalhadas as principais características dos algoritmos híbridos como uma ferramenta de processos de busca.

3.1. Introdução

Os sistemas baseados em computação evolutiva são particularmente aplicados em problemas complexos de otimização: problemas com diversos parâmetros ou características que precisam ser combinadas em busca da melhor solução; problemas com muitas restrições ou condições que não podem ser representadas matematicamente e problemas com grandes espaços de busca.

Por exemplo, os algoritmos genéticos tem sido aplicados em diversos problemas de otimização, tais como: otimização de funções matemáticas, otimização combinatória, otimização de planejamento, problema do caixeiro viajante, problema de otimização de rotas de veículos, otimização de projetos de circuitos eletrônicos, otimização de distribuição, etc. Uma importante aplicação dos algoritmos genéticos em controle é na busca das matrizes de

ponderação Q do estado e R do controle de forma a alocar a Auto-estrutura do controlador.

Diversos projetos de sistemas no mundo real são usualmente abordados como problemas de otimização. Entretanto, somente algumas combinações das variáveis envolvidas no projeto resultam realmente úteis quando são utilizadas para dar solução ao problema. Conseqüentemente, diversos algoritmos de otimização podem ser implementados para encontrar a melhor combinação das variáveis, que resultem na realização efetiva do projeto.

Algoritmos de otimização podem ser classificados em duas categorias: determinísticos e estocásticos. Os métodos determinísticos são baseados em técnicas de busca local, métodos de cobertura, métodos de bifurcação e limitados (*branch and bound*), aproximações baseadas em decomposição, representação integral e análise intervalar.

Métodos estocásticos incluem técnicas de busca aleatórias, recozido simulado (*simulated annealing*), métodos de agrupamento (*clustering*), modelos estocásticos e axiomas. Técnicas evolutivas, também são consideradas estocásticas, quando são utilizadas na solução de problemas de otimização.

Na atualidade sistemas baseados em computação evolutiva são muito utilizados como ferramentas de otimização, como por exemplo: em problemas de engenharia, ciências naturais, biologia e ciências da computação. A idéia básica é aplicar o processo de evolução natural como um paradigma de solução de problemas, utilizando para sua implementação um computador.

A utilização de sistemas baseados em métodos de aprendizado para a solução de problemas complexos de otimização necessita de algum tipo de operador de busca iterativa. Este processo de busca deve explorar o espaço que descreve todas as configurações possíveis de solução e as técnicas evolutivas é uma alternativa viável para realizar este procedimento.

Os algoritmos evolutivos apresentam uma estrutura que corresponde a uma seqüência de passos até a solução, sendo esta mesma estrutura aplicada a uma ampla gama de problemas, fornecendo uma robustez e flexibilidade dos algoritmos evolutivos. Conseqüentemente, os algoritmos evolutivos são aplicados a problemas complexos, para os quais outras técnicas conhecidas são ineficazes ou não podem ser aplicadas.

3.2. Princípios Básicos da Computação Evolutiva

A computação evolutiva pode ser classificada como sendo uma estratégia de solução concebida para resolver problemas genéricos de otimização e operar em espaços não lineares e não estacionários, ou seja, quando o problema está sujeito a pequenas variações em suas

especificações. Algoritmos evolutivos são métodos de busca estocásticos que imitam a evolução biológica natural. Os algoritmos evolutivos são compostos por uma seqüência de passos até a solução, sendo estes passos os mesmos para uma ampla gama de problemas, fornecendo robustez e flexibilidade. Sendo assim, a computação evolutiva é entendida como um conjunto de técnicas e procedimentos genéricos e adaptáveis, a serem aplicados na solução de problemas complexos, para os quais abordagens clássicas ou dedicadas não existem, não são aplicáveis ou falham quando são empregadas. Entretanto, as estratégias baseadas em computação evolutiva não podem ser escolhidas como a primeira abordagem na busca pela solução de um problema.

Na Figura 3.1 é mostrada a estrutura de um sistema baseado em computação evolutiva, basicamente, eles são modelos computacionais que recebem como entrada:

- Uma população de indivíduos (geração inicial), que correspondem às soluções candidatas junto a problemas específicos.
- Uma função que mede a adequação relativa de cada indivíduo frente aos demais (função de adequação, adaptabilidade, objetivo ou *fitness*).

Algoritmos evolutivos operam sobre populações de soluções potenciais aplicando o princípio de sobrevivência para produzir melhores aproximações para uma solução. Em cada geração, um novo conjunto de soluções é criado ao participar em um processo de seleção individual de acordo a seus níveis de aptidão (*fitness*) e a través de operadores que são simulados da evolução genética natural. Este processo conduz à evolução de populações de indivíduos que serão melhores em seu ambiente que os indivíduos de onde foram inicialmente criados, da mesma maneira que em uma adaptação natural [24].

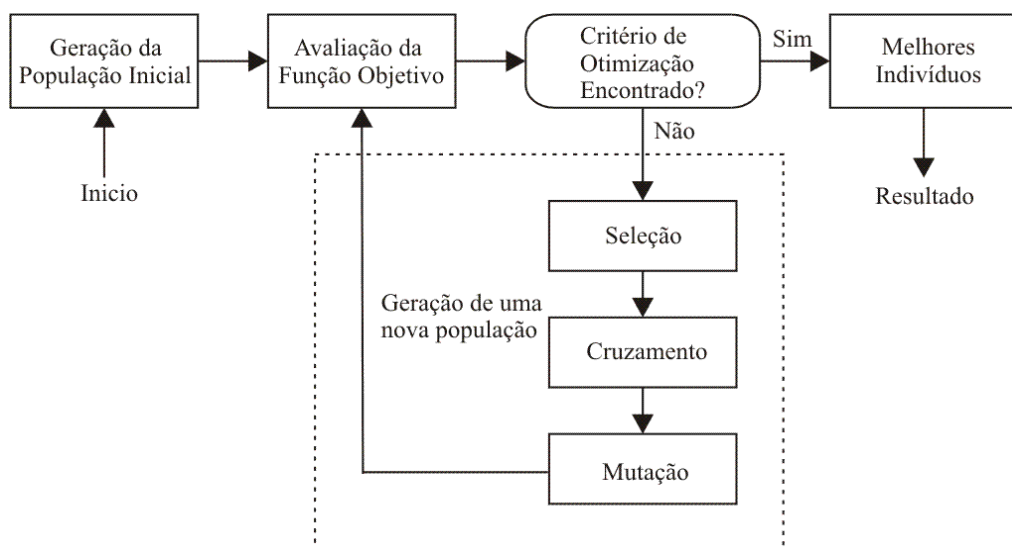


Figura 3.1. Estrutura de um programa evolutivo.

Cada indivíduo da população se encontra descrito através de uma lista ordenada (cromossomos), descritos a partir de um alfabeto finito. Cada atributo da lista é equivalente a um gene. O tamanho da lista está diretamente associado ao número mínimo de elementos necessários para descrever cada indivíduo da população (solução candidata).

As principais abordagens de algoritmos evolutivos propostas na bibliografia, que foram desenvolvidos independentemente são: *algoritmos genéticos*, *programação evolutiva* e *estratégias evolutivas*, sendo que as principais diferenças entre eles são os operadores genéticos empregados em suas implementações.

Os *algoritmos genéticos*, introduzidos por Holland e subseqüentemente estudados por De Jong, Goldberg, Davis, Eshelman, Forrest, entre outros autores, foram originalmente propostos com o objetivo de formalizar modelos de processos adaptativos, explicar rigorosamente processos de adaptação em sistemas naturais e desenvolver sistemas artificiais (simulados em computador) [25].

A *programação evolutiva*, introduzida por Fogel e estendida por Burgin, Atmar, entre outros autores, foi originalmente proposta como uma técnica para criar inteligência artificial através da evolução de máquinas de estado finito (FSM – *Finite state machine*). Uma FSM é uma máquina abstrata que transforma uma seqüência de símbolos de entrada em uma seqüência de símbolos de saída. Esta transformação depende de um conjunto finito de estados e um conjunto finito de regras de transição de estados, sendo que o desempenho de uma FSM com respeito ao ambiente é medido com base na capacidade de predição da máquina, isto é, comparar cada símbolo de saída com o seguinte símbolo de entrada [23], [25].

Estratégias evolutivas, desenvolvidas por Rechenberg e Schwefel, e estendida por Herdy, Kursawe e outros autores, foram inicialmente propostas com o objeto de solucionar problemas de otimização de parâmetros, tanto discretos como contínuos. As estratégias evolutivas empregam em sua implementação operadores de cruzamento e mutação [25].

Apesar das abordagens acima citadas terem sido desenvolvidas de forma independente, seus algoritmos possuem uma estrutura comum. Será usado o termo *algoritmo evolutivo* como uma denominação comum a todas elas. A estrutura de um algoritmo evolutivo pode ser dada na forma [17]:

```

Programa evolutivo
inicio
    t ← 0
    inicializar P(t)
    avaliação P(t)
    enquanto (não termina a condição de parada) fazer
    inicio
        t←t+1
        selecionar P(t) de P(t-1)
        modificar P(t)
        avaliar P(t)
    fim
fim

```

A programação evolutiva é um algoritmo probabilístico, o qual mantém uma população de indivíduos $P(t) = \{x_1^t, \dots, x_n^t\}$ na iteração (geração) t . Cada indivíduo representa um candidato à solução do problema em questão e, em qualquer programa evolutivo, assume a forma de alguma estrutura de dados S . Durante a iteração t cada solução x_i^t é avaliada e produz alguma medida de adaptação, ou *fitness*. Então, uma nova população é formada na iteração $t+1$ pela seleção dos indivíduos mais adaptados. Alguns indivíduos da população são submetidos a um processo de alteração por meio de operadores genéticos para formar novas soluções. Existem transformações m_i (mutação) que criam novos indivíduos através de pequenas modificações de atributos em um indivíduo ($m_i: S \rightarrow S$), e transformações de ordem superior c_j (cruzamento), que criam novos indivíduos através da combinação de dois ou mais indivíduos ($c_j: S \times \dots \times S \rightarrow S$). Após um número de gerações, a condição de parada deve ser atendida, a qual geralmente indica a existência, na população, de um indivíduo que representa uma solução aceitável para o problema, ou quando o número máximo de gerações foi atingido [17, 26].

As abordagens de computação evolutiva tais como *algoritmos genéticos*, *programação evolutiva* e *estratégias evolutivas* diferem em diversos aspectos, dentre os quais se destacam: estruturas de dados utilizadas para codificar um indivíduo, operadores genéticos empregados, métodos para criar a população inicial e métodos para selecionar indivíduos para a geração seguinte. Entretanto, elas compartilham o mesmo princípio comum: uma população de indivíduos sofre algumas transformações e durante a evolução os indivíduos competem pela sobrevivência. A seguir é apresentada uma descrição mais detalhada dos algoritmos genéticos.

3.3. Algoritmos Genéticos

Os algoritmos genéticos empregam uma terminologia originada da teoria da evolução natural e da genética para simular sistemas genéticos. Os algoritmos genéticos processam populações, sendo que um indivíduo da população é representado por um cromossomo, o qual contém a codificação (genótipo) de uma possível solução do problema (fenótipo). Os cromossomos são usualmente implementados na forma de estruturas, listas de atributos ou vetores, onde cada atributo é conhecido como gene [16].

O processo de evolução executado por um algoritmo genético corresponde a um procedimento de busca em um espaço de soluções potenciais para o problema. Esta busca requer um equilíbrio entre dois objetivos: o aproveitamento das melhores soluções e a exploração do espaço de busca [17]. Este equilíbrio está muito longe de ocorrer quando se consideram outras técnicas de busca, como por exemplo:

- Métodos clássicos de otimização, como o método do gradiente. Estes métodos aproveitam apenas a melhor solução na busca de possíveis aprimoramentos, sem realizar uma exploração do espaço de busca.
- Métodos de busca aleatórios. Estes métodos exploram o espaço de busca ignorando o aproveitamento de regiões promissoras do espaço [26].

Os algoritmos genéticos pertencem à classe de algoritmos probabilísticos, contudo, não são algoritmos puramente aleatórios, pois combinam elementos de direção e procura estocástica. Devido a isto, os algoritmos genéticos são mais robustos, existindo uma direção no processo de busca. Outra propriedade importante dos algoritmos genéticos é que eles mantêm uma população de soluções potenciais, ao contrario de outros métodos que processam um só ponto do espaço de busca [17].

Como foi mencionado anteriormente, algoritmo genético realizam um processo de busca multi-direcional, mantendo uma população de soluções potenciais trocando informações entre si. A população é processada mediante uma evolução simulada cujo objetivo é: a cada geração reproduzir soluções relativamente “boas”, enquanto são eliminadas soluções relativamente “ruins”. Para distinguir entre diferentes soluções utiliza-se uma função de avaliação ou de adaptabilidade (*fitness*) que simula o papel da pressão exercida pelo ambiente sobre o indivíduo [17].

A estrutura de um algoritmo genético simples é a mesma de um programa evolutivo como foi apresentado na Figura 3.1. Durante a iteração t , um algoritmo genético mantém uma população de soluções potenciais (indivíduos, cromossomos, lista de atributos ou vetores)

$P(t) = \{x_1^t, \dots, x_n^t\}$. Cada solução x_i^t é avaliada e produz uma medida de sua adaptação. Logo, uma nova população na iteração $t + 1$ é formada privilegiando a participação dos indivíduos mais adaptados. Alguns membros da nova população passam por alterações, por meio de operadores de cruzamento e mutação, para formar novas soluções potenciais. Este processo se repete até que um número pré-determinado de iterações seja atingido, ou até que um nível de adaptação esperado seja alcançado [17].

Conseqüentemente, para um problema particular, um algoritmo genético deve ter os seguintes componentes:

- Uma representação genética para soluções candidatas ou potenciais (processo de codificação);
- Uma maneira de criar uma população inicial de soluções candidatas ou potenciais;
- Uma função de avaliação que faz o papel da pressão ambiental, classificando as soluções em termos de sua adaptação ao ambiente (ou seja, sua capacidade de resolver o problema);
- Operadores genéticos que alteram as composições das gerações;
- Valores para os diversos parâmetros usados pelo algoritmo genético (tamanho da população, probabilidades de cruzamento e mutação, etc.).

3.3.1. Representação de Indivíduos (Codificação)

Cada indivíduo de uma população representa um candidato potencial à solução do problema em questão. No algoritmo genético clássico, proposto por Holland (1975) as soluções candidatas ou potenciais são codificadas em arranjos binários de tamanho fixo. A motivação para o uso de codificação binária vem da teoria dos esquemas (*schemata theory*), utilizada para explicar por que os algoritmos genéticos funcionam. Holland argumenta que seria benéfico para o desempenho do algoritmo maximizar o paralelismo implícito inerente ao algoritmo genético, e prova que um alfabeto binário maximiza o paralelismo implícito [25].

A representação binária dos indivíduos em arranjos binários do alfabeto $\{0, 1\}$ é da forma:

$$f: \{0,1\}^l \rightarrow \mathcal{R} \quad (3.1)$$

A representação binária dos algoritmos genéticos utiliza funções de codificação e decodificação $h: M \rightarrow \{0,1\}^l$ e $h': \{0,1\}^l \rightarrow M$ que facilitam o mapeamento de soluções $Z \in M$

para arranjos binários $h(Z) \in \{0,1\}^l$ e vice versa.

Em contraste com a representação dos indivíduos em *algoritmos genéticos*, *estratégias evolutivas* e *programação evolutiva* são diretamente baseadas em representações com vetores de valores reais, sendo geralmente aplicados para resolver problemas de otimização com parâmetros contínuos de valores reais [25], da forma:.

$$f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R} \quad (3.2)$$

A representação ou codificação dos indivíduos é uma das etapas mais importantes quando são definidos algoritmos genéticos. A definição inadequada da codificação pode levar a problemas de convergência prematura do algoritmo genético. A estrutura de um cromossomo deve representar uma solução como um todo, e deve ser a mais simples possível.

Em problemas de otimização restrita, a codificação adotada pode fazer com que indivíduos modificados pelos operadores de cruzamento e mutação sejam inválidos. Nestes casos, cuidados especiais devem ser tomados na definição da codificação e na aplicação dos operadores.

3.3.2. Definição da População Inicial

O processo de definição de uma população inicial para algoritmos genéticos é simples, cria-se uma população de cromossomos inicializados aleatoriamente, sendo que cada cromossomo é composto por um arranjo binário de tamanho finito. Em problemas com restrições, deve-se tomar cuidado para não gerar indivíduos inválidos na etapa de inicialização.

3.3.3. Seleção

Uma vez escolhido o tipo de codificação do cromossomo, o passo seguinte trata da definição do método de seleção a ser implementado, o qual envolve a escolha dos indivíduos na população que irá criar descendentes e quantos descendentes serão criados. O objetivo da seleção é privilegiar os indivíduos melhor adaptados, conseguindo que seus descendentes tenham um desempenho ainda melhor.

A seleção deve ser tal que produza um balanço adequado entre a pressão seletiva e a variação introduzida pelos operadores genéticos. Por exemplo, métodos de seleção com elevada pressão seletiva, tendem a gerar super-indivíduos, isto é, indivíduos com aptidão superiores aos demais, reduzindo a diversidade genética da população. A presença de um super-indivíduo pode gerar uma convergência prematura do algoritmo genético. Entretanto,

métodos de seleção com baixa pressão seletiva tendem a produzir progressos muito lentos no processo evolutivo. A seguir, são descritos alguns dos métodos de seleção mais comumente encontrados na bibliografia.

Seleção Proporcional à Função de Aptidão (Fitness)

No método de seleção proporcional à adaptabilidade também conhecido como roleta ou *Roulette Wheel*, atribui-se a cada indivíduo de uma população uma probabilidade de passar à próxima geração proporcional à sua aptidão ou *fitness*. Conseqüentemente, indivíduos com maior valor do *fitness* terão maior probabilidade de passar à próxima geração. Mas, este método de seleção pode fazer que indivíduos bem adaptados sejam perdidos, ou seja, não passem para a próxima geração, isto devido a que este método de seleção é tipicamente implementado como um operador probabilístico.

Para uma população de n indivíduos $P = \{h_1, h_2, \dots, h_n\}$, na qual o i -ésimo indivíduo h_i tem associado a ele uma medida de aptidão positiva e não nula ($F(h_i) \in \mathfrak{R}_0^+$), a probabilidade $Prob_i$ deste indivíduo ser selecionado é dada por:

$$Prob_i = \frac{F(h_i)}{\sum_{j=1}^n F(h_j)} \quad (3.3)$$

Na seleção por roleta, o *fitness* pode ser representado por segmentos consecutivos em uma roleta imaginária e a probabilidade de escolha de um segmento depende diretamente de seu tamanho [23]. A Figura 3.2 apresenta-se um exemplo de como seria a roleta para os indivíduos representados na iteração t .

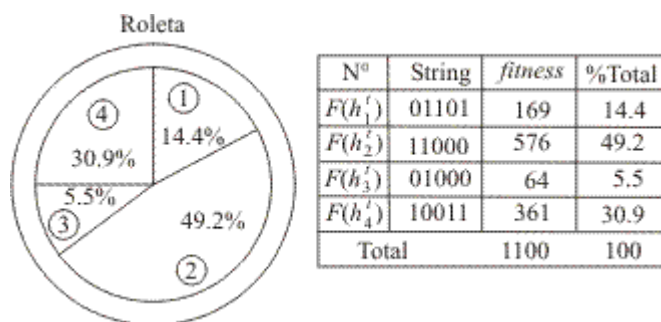


Figura 3.2. Seleção por roleta em algoritmos genéticos

Seleção Baseada na Classificação (Rank)

A seleção proporcional considerando a aptidão dos indivíduos pode ser problemática

se os indivíduos da população apresentarem desempenhos muito próximos entre si. Além disso, se o tamanho da população é pequeno, a perda de diversidade genética pode levar à convergência prematura, pois a busca fica reduzida a poucos pontos, causando uma diminuição no poder de exploração do algoritmo genético. Uma opção para se evitar o surgimento de super-indivíduos (relativos aos demais existentes na população atual) e a ocorrência de convergência prematura é reduzir as diferenças entre estes, a través de um mecanismo de seleção baseado em *rank* [24]. Esta estratégia utiliza as posições dos indivíduos quando são ordenados de acordo com o *fitness* para determinar a probabilidade de seleção. Podem ser usados mapeamentos lineares ou não-lineares para determinar a probabilidade de seleção.

Para uma população de n indivíduos $P = \{h_1, h_2, \dots, h_n\}$, são arranjados em ordem crescente de aptidão ou *fitness*, tal que $F(h_i) < F(h_j)$, para todo $1 \leq i < j \leq n$. Define-se SP como a pressão de seleção do ambiente sobre os indivíduos. Neste caso, o valor da função de aptidão pode ser calculado por:

- *Ranking Linear*: O ranking linear permite valores de pressão de seleção SP entre $[1; 2,0]$.

$$F(i) = 2 - SP + \frac{2 \cdot (SP - 1) \cdot (i - 1)}{n - 1} \quad (3.4)$$

- *Ranking não linear*: O ranking não linear permite valores de pressão de seleção entre $[1; n-2]$

$$F(i) = \frac{n \cdot v^{n-1}}{\sum_j v^{j-1}}; i = 1, \dots, n \quad (3.5)$$

sendo que v é calculado como a raiz do polinômio:

$$(SP - n)v^{n-1} + SP \cdot v^{n-2} + \dots + SP = 0 \quad (3.6)$$

Na Figura 3.3 é comparada a seleção por classificação linear e não linear, tais mecanismos de seleção melhoram o comportamento do algoritmo genético, com um controle melhor da pressão de seleção.

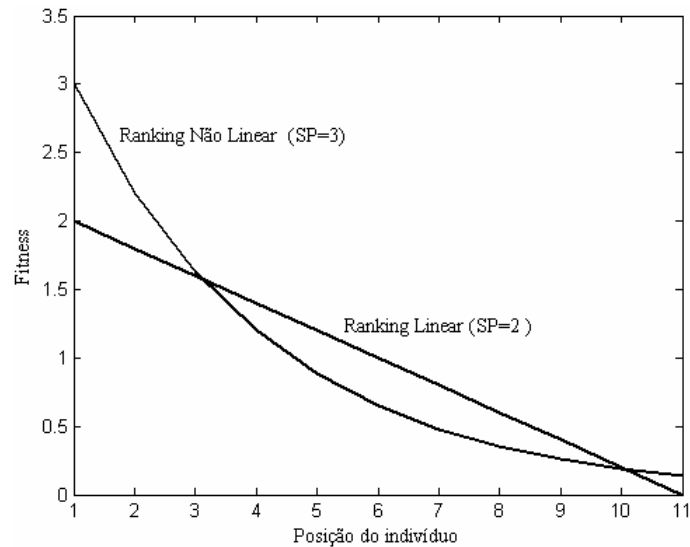


Figura 3.3 Assinação do *fitness* para o ranking linear e não linear.

A seleção por classificação tem a desvantagem de exigir a ordenação de toda a população, o que pode representar um custo computacional em algumas aplicações específicas.

Elitismo

O termo elitismo está associado à adoção de uma operação adicional junto aos métodos de seleção, que força o algoritmo genético a reter o melhor indivíduo ou um número de melhores indivíduos, a cada geração. Estes indivíduos poderiam ser perdidos se não fossem selecionados de forma determinística para compor a próxima geração, ou então fossem modificados por operadores de cruzamento ou mutação. Em grande parte dos casos, estratégias elitistas associadas aos métodos de seleção melhoram o desempenho do algoritmo genético [23].

3.3.4. Operadores Genéticos

Após a escolha da codificação e método da seleção a utilizar, o passo seguinte é o processo de reprodução do algoritmo genético. O processo de reprodução é responsável pela geração de novas soluções, produto da aplicação de operadores genéticos junto aos indivíduos selecionados. A variação genética na população é um ponto principal na evolução, garantindo a exploração de novas soluções. Conseqüentemente os operadores genéticos representam a fonte de diversidade e variabilidade.

A escolha dos operadores genéticos depende fortemente da codificação adotada para a representação genética. Os operadores genéticos mais comumente utilizados são o cruzamento

(também conhecido como recombinação) e a mutação. O operador de cruzamento combina os indivíduos trocando suas informações genéticas e também traz alguma inovação. No processo de cruzamento há uma tendência maior de gerar cromossomos com combinações de características já existentes na população, resultando em uma convergência mais efetiva do processo evolutivo. Entretanto, o operador de mutação introduz novas combinações genéticas até então inéditas na codificação. Na mutação, informações que foram perdidas ao longo do processo evolutivo podem ser resgatadas, ou mesmo informações totalmente novas podem ser introduzidas.

A seguir apresentam-se os principais aspectos relacionados aos operadores cruzamento e mutação.

O operador de Cruzamento

O operador de cruzamento é o responsável pelo intercâmbio de informação genética entre os indivíduos de uma população, produzindo novas soluções candidatas ou potenciais com algumas características já existentes dos pais. O processo de cruzamento depende da escolha dos pares (ou grupos) de indivíduos. A probabilidade de ocorrência de recombinação entre dois indivíduos de uma população é denominada *taxa de cruzamento*. Esta probabilidade de cruzamento geralmente varia entre 0,5 e 1,0. No entanto, uma alta probabilidade de cruzamento faz com que indivíduos com uma maior aptidão, sejam eliminados antes que o processo de seleção possa produzir aperfeiçoamento. Por outro lado, uma baixa probabilidade de cruzamento pode convergir lentamente devido à baixa taxa de exploração das características genéticas.

Quando é utilizada codificação binária na implementação dos algoritmos genéticos, o operador de cruzamento mais simples é o cruzamento de um ponto. Para a aplicação deste operador, são selecionados dois indivíduos (pais) e a partir de seus cromossomos são gerados dois novos indivíduos (filhos). Para gerar os filhos, seleciona-se um mesmo ponto de corte aleatoriamente nos cromossomos dos pais, e os segmentos de cromossomo criados a partir do ponto de corte são trocados. Este tipo de operador impõe a quebra da seqüência genética, definindo qual é a proporção de informação dos pais que cada descendente receberá.

Uma extensão do operador de cruzamento de um ponto é o cruzamento de dois pontos, no qual dois pontos de corte são escolhidos e material genético são trocados entre eles. Outra variação é o operador cruzamento de múltiplos pontos, o qual escolhe de forma aleatória mais de uma posição de troca.

Assim também existem operadores de cruzamento especialmente desenvolvidos para

uso com codificação em ponto flutuante. Um exemplo é o chamado *cruzamento aritmético*. Este operador está definido como uma combinação linear de dois vetores (cromossomos): sejam x_1 e x_2 dois indivíduos selecionados para cruzamento, então os dois filhos resultantes serão $x'_1 = a.x_1 + (1-a).x_2$ e $x'_2 = (1-a).x_1 + a.x_2$, sendo a uma variável aleatória definida no intervalo $a \in [0, 1]$. Este operador é particularmente apropriado para problemas de otimização numérica com restrições, onde a região factível é convexa. Isto porque, se x_1 e x_2 pertencem à região factível, combinações convexas de x_1 e x_2 serão também factíveis. Assim, garante-se que o operador de cruzamento não gera indivíduos inválidos [17].

No entanto, não há nenhum operador de cruzamento que claramente apresente um desempenho superior aos demais. Isto porque, cada operador de cruzamento é particularmente eficiente para uma determinada classe de problemas e extremamente ineficiente para outras.

O operador de Mutação

O operador de mutação modifica aleatoriamente um ou mais genes de um cromossomo. A probabilidade de ocorrência de mutação em um gene é denominada *taxa de mutação*. Usualmente, são atribuídos valores pequenos para a taxa de mutação. A principal contribuição do operador de mutação é criar uma variabilidade extra na população, mas sem destruir o progresso já obtido com a busca.

Considerando codificação binária, o operador de mutação simplesmente troca o valor de um gene em um cromossomo. A aplicação do operador de mutação depende da escolha do ponto de mutação ou posição escolhida aleatoriamente, sendo que todos os pontos têm a mesma probabilidade de serem escolhidos. Assim, se um gene selecionado para mutação tem valor 1, o seu valor passará a ser 0 após a aplicação da mutação, e vice versa.

No caso de problemas com codificação em ponto flutuante, os operadores de mutação mais populares são: mutação uniforme, mutação não uniforme e mutação gaussiana [17].

O operador de mutação uniforme seleciona aleatoriamente um componente $k \in \{1, 2, \dots, n\}$ do cromossomo $x = [x_1 \dots x_k \dots x_n]$ e gera um indivíduo $x' = [x_1 \dots x'_k \dots x_n]$, sendo x'_k é um número aleatório (com distribuição de probabilidade uniforme) amostrado no intervalo $x_k \in [x_{kmin}, x_{kmax}]$.

O operador de mutação não uniforme foi especialmente desenvolvido para problemas de otimização com restrições e codificação em ponto flutuante, destinada a realizar a sintonia fina aos indivíduos da população.

No caso do operador de mutação gaussiana, todos os componentes de um cromossomo

$\mathbf{x} = [x_1, \dots, x_n]$ são modificados na forma:

$$\mathbf{x}' = \mathbf{x} + N(0, \sigma) \quad (3.7)$$

sendo $N(0, \sigma)$ um vetor de variáveis aleatórias gaussianas independentes, com média zero e desvio padrão σ .

3.4. Algoritmos Híbridos

Quando empregamos técnicas heurísticas de busca, como por exemplo, algoritmos genéticos, nem sempre é garantido encontrar uma solução ótima ao problema em questão. Conseqüentemente, podem-se implementar algoritmos genéticos com combinação de algoritmos matemáticos de busca local, conseguindo uma maior capacidade de procurar soluções ótimas.

Algoritmos híbridos empregam os seguintes princípios na sua implementação:

1. *Utilização da codificação do AG*; o algoritmo híbrido utiliza a codificação empregada na implementação do algoritmo genético;
2. *Hibridização onde seja possível*; incorporar as características positivas de algoritmos genéticos nos algoritmos híbridos.
3. *Adaptação dos operadores genéticos*; Criar operadores de cruzamento e mutação para o novo tipo de codificação por analogia dos operadores binários de cruzamento e mutação. Incorporar heurísticas aos operadores.

Usamos o termo de *algoritmo genético híbrido* para algoritmos criados por aplicação destes três princípios [17].

Os algoritmos genéticos híbridos e os programas evolutivos compartilham idéias comuns: para algoritmos genéticos aplicados a sistemas complexos, deve ser utilizando uma estrutura de dados apropriada (Utilização de uma codificação atual), e operadores adequados genéticos (Adaptação de operadores genéticos).

Capítulo 4

Algoritmo Hierárquico Evolutivo para Determinação de Aproximação de Funções

Neste capítulo apresenta-se a proposta de um algoritmo hierárquico evolutivo para determinação de aproximação de funções não lineares. As funções não lineares são aproximadas por funções lineares por partes. O algoritmo hierárquico evolutivo é implementado inicialmente utilizando um algoritmo genético padrão. Devido a problemas de convergência do algoritmo genético padrão, seguidamente, implementou-se um algoritmo híbrido composto por um algoritmo genético padrão e um algoritmo de programação matemática. Finalmente, apresentam-se os aspectos relacionados à implementação do algoritmo de programação matemática.

4.1. Introdução

Diversas metodologias podem ser implementadas para determinar a aproximação de funções relacionais (pares ordenados da forma entrada – saída), por exemplo aproximações utilizando: análise de regressão, rede neural, lógica nebulosa (*Fuzzy Logic*), computação evolutiva [18], etc.

Para a aproximação por análise de regressão, primeiramente escolhe-se a forma de uma função para enquadrar os pares ordenados de entrada – saída. De acordo com a escolha, utiliza-se algum critério de avaliação do erro (por exemplo, erro médio quadrático). A escolha

da forma da função compreende um conjunto de funções ortogonais, que formam um espaço base. Um inconveniente de utilizar aproximação por análise de regressão é que com os teoremas que formalizam as propriedades de aproximação por séries de funções ortogonais sabe-se como fazer, mas não como é possível fazer. Por exemplo, estes algoritmos indicam que deve utilizar-se um polinômio de ordem n mais não indica o valor exato dos coeficientes do polinômio.

Assim também se utilizam sistemas dinâmicos treináveis como rede neural e lógica nebulosa (*Fuzzy Logic*), para estimar funções relacionais da forma entrada – saída. Diferentemente de um estimador estatístico, eles estimam a função sem um modelo matemático que represente a relação da saída dependendo da entrada. Conseqüentemente são estimadores de funções que utilizam modelos livres. Um inconveniente de usar estes métodos de aproximação é que eles não fornecem uma relação funcional, o que significa que as derivadas e integrais da função aproximada não são disponíveis.

A utilização dos métodos descritos anteriormente torna uma solução do problema de aproximação de funções muito difícil e complexa, o que viabiliza a utilização de algoritmos baseados em computação evolutiva (algoritmos genéticos, programação evolutiva e estratégias evolutivas), para aproximar funções relacionais (pares ordenados da forma entrada – saída).

Na determinação de aproximação de funções requer-se um método automatizado que procure as soluções das funções aproximadas, considerando-se os seguintes aspectos das funções: não diferenciáveis, complexas, descontínuas e com espaços de busca multimodais. Neste panorama pode-se implementar um algoritmo genético para encontrar soluções, com um processo de evolução robusto de uma representação das soluções [18].

Para a implementação de aproximação de funções não lineares em sistemas embarcados de baixo custo com capacidade de cálculo em ponto fixo, devem considerar-se aspectos como: incertezas associadas as variáveis livres, saturação na representação de números e efeitos de quantização devido à resolução de armazenamento dos pontos de quebra.

Funções aproximadas podem ser implementadas utilizando-se aproximação por funções lineares por partes, determinando-se o tamanho da tabela de equivalência (LUT), que será utilizada. Neste sentido a implementação do algoritmo genético deve atingir os objetivos seguintes: calcular o número e as posições dos pontos de quebra necessários para reproduzir os valores aproximados da função não linear a partir dos pontos de quebra. Determinar o tamanho da tabela de equivalência em bits levando em consideração aspectos como restrições e limites para aproximação de funções em sistemas embarcados.

4.2. Formulação para determinar aproximação de funções através de algoritmos genéticos

Na formulação e na implementação do algoritmo genético utilizado, algumas extensões foram incluídas, com o intuito de tornar o processo de busca de soluções mais eficiente. Essas extensões envolvem a utilização de aproximação de funções utilizando funções lineares por partes e uma normalização do problema para um melhor entendimento das soluções, baseado nos limites de erros de aproximação.

4.2.1. Aproximação de funções não lineares por funções lineares por partes

Uma função não linear definida por $y = f(x)$, pode ser aproximada através de funções lineares por partes, como visto na Seção 2.6. A função aproximada $s = \tilde{f}(x)$ é construída por partes a partir de valores armazenados em uma tabela de equivalência, que contém m pontos de quebra p_{xi} e p_{yi} , com $i = 1, \dots, m$. Os pontos de quebra p_{xi} são quantizados com uma resolução equivalente a N bits e os pontos de quebra p_{yi} são determinados com uma exatidão equivalente a uma resolução de N_T bits de forma que os valores gerados da função tenham uma resolução equivalente de N_Y bits (considerando as incertezas de propagação e de aproximação) [4]. A função não linear é aproximada considerando uma exatidão equivalente do sistema embarcado de N_T bits de resolução, com $N_T \geq N_Y$, isto é, os pontos de quebra são armazenados na tabela de equivalência com uma resolução equivalente de N_T bits.

4.2.2. Normalização das Soluções

Na Seção 2.4 foi definido o erro de aproximação como $\varepsilon_A = s - y$, limitado por $[\varepsilon_{Amin}, \varepsilon_{Amax}]$. O problema de aproximação consiste em determinar valores da função aproximada s tal que o erro de aproximação ε_A se encontre dentro deste limites de aproximação. Para facilitar a interpretação da aproximação com relação aos limites do erro de aproximação de ε_A , uma normalização das soluções foi estabelecida. Este processo de normalização subtrai de todas as variáveis os valores da função quantizada y_q a N_T bits de resolução. Com esta transformação, deslocam-se as variáveis para em torno de zero, desta maneira o erro de aproximação normalizado assumem valores inteiros limitados por um erro normalizado E_{Amin} e E_{Amax} , como foi apresentado na Seção 2.5 [21].

Uma vantagem de utilizar esta normalização no problema de aproximação de funções, é que se procura as soluções dos pontos de quebra através de limites de erros de aproximação

sem importar a forma da função não linear em questão.

4.3. Procedimento de Solução

Na metodologia proposta desenvolve-se um algoritmo hierárquico evolutivo cujo objetivo é de procurar uma solução ótima do problema de aproximação de funções. O algoritmo evolutivo foi desenvolvido em duas partes. Inicialmente, foi desenvolvido um algoritmo genético padrão e foi avaliado o seu desempenho através de estudo de casos. Os resultados apresentados com o uso desse algoritmo foram satisfatórios para casos com pequenas resoluções de N e N_Y até 10 bits. Entretanto, esse algoritmo apresentou problemas de convergência para resoluções maiores de 10 bits.

De forma a melhorar a convergência do algoritmo evolutivo, em seguida, ao algoritmo genético foi incorporado um bloco de programação matemática. Desta forma utiliza-se um algoritmo genético para determinar uma estrutura apropriada da solução, enquanto que o algoritmo da programação matemática identifica a estrutura de super-indivíduos usando como espaço de busca a população atual.

4.3.1. Algoritmo Hierárquico

O procedimento proposto possui dois níveis hierárquicos, como representado na Figura 4.1. No primeiro nível define-se o tamanho da tabela de equivalência ($m \times N_T$), com m pontos de quebra e uma resolução equivalente de N_T bits. No segundo nível é implementado um algoritmo evolutivo híbrido, que consiste de um algoritmo genético padrão, combinado com um bloco de programação matemática. O algoritmo evolutivo híbrido, nível mais hierárquico baixo, procura uma solução factível para a condição especificada no primeiro nível. Se a solução é encontrada, esta é repassada ao primeiro nível, que procederá reduzindo o tamanho da tabela de equivalência. Com esta nova condição, o segundo nível é novamente ativado. Se uma solução factível não é encontrada, considera-se como resultado a última solução encontrada (solução incumbente) [22].

Na Figura 4.1 são mostradas as especificações do problema como: o conjunto de valores quantizados da variável independente x , o conjunto de valores quantizados da variável dependente y e os erros de aproximação normalizados E_{Amin} e E_{Amax} , definido na Seção 2.5 pela equação (2.34).

Uma vez que as especificações de entrada $\{x, y_q, E_{Amin}, E_{Amax}\}$ são submetidas para o algoritmo hierárquico híbrido, as soluções obtidas para as especificações definidas são as

posições dos pontos de quebra (p_x, p_y) e o tamanho da tabela de equivalência ($m \times N_T$).

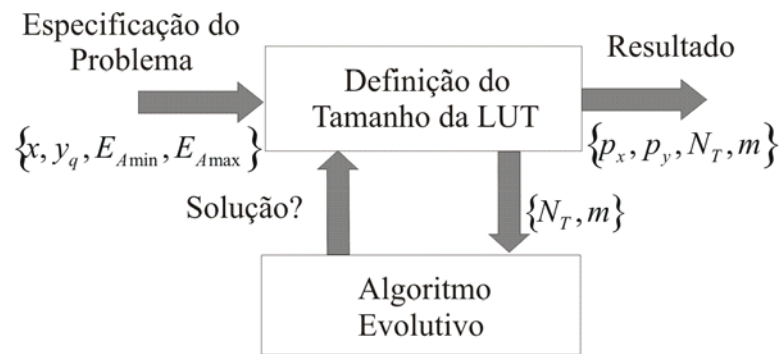


Figura 4.1. Diagrama do procedimento hierárquico de busca.

O algoritmo de mais alto nível procura as soluções de aproximação a partir de um par de resoluções (N e N_Y) e número de pontos de quebra inicial (m). Encontrando a solução para o tamanho da LUT em bits ($N_T \times m$), seguidamente, o algoritmo decresce um desses parâmetros e tenta achar solução. Esse procedimento é representado na Figura 4.2.

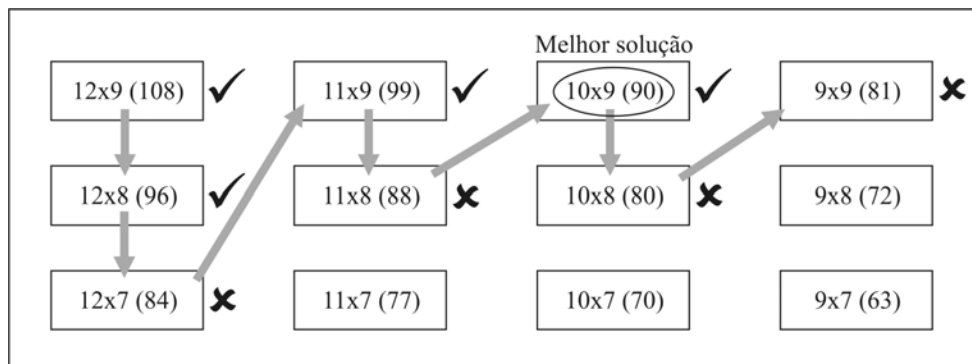


Figura 4.2. Definição do tamanho da LUT com solução para $N_T = 10$ e $m = 9$.

4.3.2. Algoritmo Genético Padrão

Para resolver o segundo nível, desenvolveu-se inicialmente um algoritmo genético padrão, cujos objetivos são determinar o mínimo tamanho da tabela de equivalência ($m \times N_T$), e determinar as posições dos pontos de quebra com as especificações determinadas no primeiro nível de hierarquia.

O algoritmo genético realiza o seguinte procedimento:

- O algoritmo se inicia com a criação de uma população inicial aleatória de indivíduos, representando-se os indivíduos com uma codificação de números inteiros projetados em uma região viável;

- Avalia-se a função de aptidão de cada indivíduo de acordo com critério pré-definido. Depois da avaliação da função de aptidão, os indivíduos são selecionados utilizando uma pontuação baseada em classificação não linear e seleção através de roleta, assim, os indivíduos que tem a maior probabilidade passarão sua informação genética para a geração seguinte;
- Aplica-se a operação de recombinação a dois indivíduos selecionados de acordo com uma classificação não linear. A operação de recombinação troca material genético entre os indivíduos produzindo soluções candidatas ou potencias com características dos pais;
- Aplica-se a operação de mutação aos indivíduos da população que simulara uma mudança aleatória de alguns bits no cromossomo dos filhos. O operador de mutação provê um mecanismo para exploração de novas regiões do espaço de solução evitando uma convergência prematura para uma solução mínima local;
- Ordenam-se os indivíduos da população atual, segundo o valor da função de aptidão, sendo o objetivo principal manter os melhores indivíduos para a geração seguinte.

Este processo evolutivo é repetido para um número especificado de gerações ou até que algum critério de parada seja atingido. Para o problema abordado nessa dissertação, o algoritmo proposto é descrito sumariamente a seguir:

Algoritmo Genético Padrão

Início

- $t \leftarrow 0$;
- Inicialize $P(t)$;
- Avalie $P(t)$;

Enquanto (não condição de parada) Faça

Início

- $t \leftarrow t+1$;
- Selecione $P(t)$ a partir de $P(t-1)$;
- Cruzamento $P(t)$;
- Mutação $P(t)$;
- Avalie $P(t)$;

Fim

- Retorna a melhor solução;

Fim

A seguir desenvolvem-se aspectos que envolvem a implementação do algoritmo genético padrão, como por exemplo: representação dos indivíduos, determinação da função de aptidão, método de seleção e operadores de reprodução genética.

Inicialização da População de Indivíduos

Na codificação define-se a maneira como os genes são armazenados no cromossomo, os quais representam parâmetros a serem otimizados pelo algoritmo genético. No algoritmo implementado utiliza-se uma representação dos cromossomos com codificação inteira, em que cada cromossomo tem um comprimento de m pontos de quebra (p_{xi}, p_{yi}) , $i = 1, \dots, m$, representados por valores inteiros com tamanhos de palavras de N e de N_T bits respectivamente, como representado na Figura 4.3. O cromossomo é inicializado aleatoriamente, porém, projetado dentro de uma região viável de variação dos pontos de quebra: $p_{xi} \in [0, 2^N - 1]$ e $p_{yi} \in [0, 2^{N_T} - 1]$, $i = 1, \dots, m$.

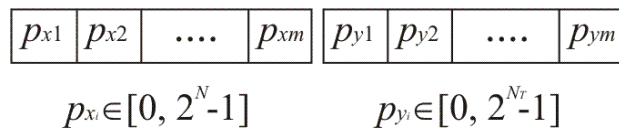


Figura 4.3. Representação do cromossomo.

O tamanho da população inicial é definido por Pop , conseqüentemente a população inicial se encontra constituída por Pop cromossomos, em que cada cromossomo representa uma solução candidata para o problema. Na Figura 4.4 apresenta-se a estrutura da população de indivíduos a ser otimizada pelo algoritmo genético.

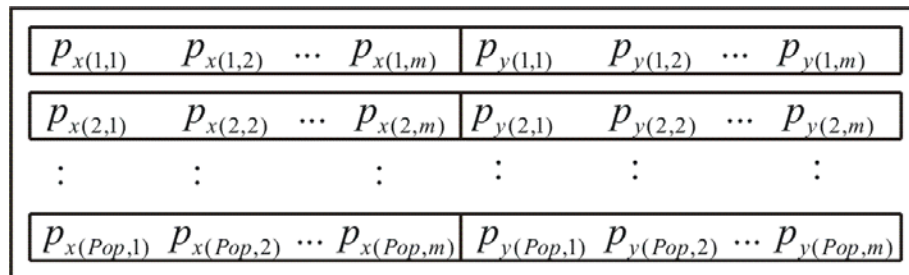


Figura 4.4. Estrutura da população total de indivíduos.

Avaliação da Função de Aptidão das Soluções Candidatas

O erro de aproximação de cada indivíduo da população é dado por $\varepsilon_i = s_i - y_q$, em que s_i é a função de aproximação do indivíduo i , y_q é o valor quantizado da função y . O algoritmo genético tem que minimizar o erro de aproximação considerando seus limites. Para cada indivíduo, a função e o erro de aproximação são calculados utilizando funções lineares por partes [21]. Como os limites do erro de aproximação não são constantes para faixa de variação de x (devido à própria quantização e à propagação da incerteza na variável independente) o erro de aproximação normalizado é determinado por:

$$\varepsilon_{AN}(x) = \left\{ \begin{array}{l} \frac{\varepsilon_A(x)}{E_{Amax}(x)} \mid \varepsilon_A(x) > 0 \\ \frac{\varepsilon_A(x)}{E_{Amin}(x)} \mid \varepsilon_A(x) < 0 \end{array} \right\} \quad (4.8)$$

Assim, cada indivíduo da população tem associado uma função de aptidão, que reflete quão boa ela é, comparada com outras na população. Define-se a função de aptidão como *fit*, para cada indivíduo *i*, determinada como uma combinação de três parâmetros:

$$fit(i) = f_1(i) + \sqrt{f_2^2(i) + f_3^2(i)} / w_f \quad (4.9)$$

sendo w_f um fator de ponderação, que foi ajustado para 40, e f_1 , f_2 e f_3 , são respectivamente valores máximo, médio e desvio padrão, dados por:

$$f_1(i) = \max(\varepsilon_{AN}^2) ; f_2(i) = mean(\varepsilon_{AN}^2) ; f_3(i) = std(\varepsilon_{AN}^2) \quad (4.10)$$

A função de aptidão, definida em (4.9), força o algoritmo a minimizar o máximo do erro de aproximação normalizado considerando a minimização também dos valores médio e desvio padrão deste erro. O fator de ponderação tem por objetivo evitar o mascaramento do primeiro parâmetro pelos outros dois [21].

A seguir, analisa-se o comportamento da função de aptidão proposta na equação (4.9), para isto, faz-se uso do estudo de caso apresentado na Seção 2.8.1, na qual foi analisado o erro de aproximação da função seno considerando a variável independente x como exata, isto é, $N = N_Y$. Foi considerado que a variável livre x é quantizada com uma resolução equivalente de $N = 4$ bits e os valores da função aproximada são gerados com uma resolução de $N_T = 6$ bits.

Inicialmente considera-se que os pontos de quebra são determinados sobre a curva da função quantizada. Na Figura 4.5 apresenta-se o comportamento da função de aptidão definido na equação (4.9), em que os pontos p_x variam no intervalo: $0 < p_{x1} < p_{x2} < 15$. Neste caso determina-se o valor mínimo da função de aptidão $fit_{min} = 0,4340$, com as soluções das posições dos pontos de quebra em $p_x = [0, 7, 12, 15]$ e $p_y = [3, 43, 60, 63]$.

Na Figura 4.6 apresenta-se a função aproximada com valor mínimo da função de aptidão. A função é aproximada por funções lineares por partes a partir das posições dos pontos de quebra armazenados em uma tabela de equivalência de um sistema embarcado. Na Figura 4.7 apresenta-se a normalização da função aproximada, observa-se que o erro normalizado S , encontra-se dentro dos limites do erro normalizado E_{Amin} e E_{Amax} . Observa-se que o mapeamento dos valores dos pontos $p_{x1} = 7$ e $p_{x2} = 12$ encontram-se sobre o eixo zero, pois os pontos p_{y1} e p_{y2} se encontram determinados sobre a função quantizada y_q .

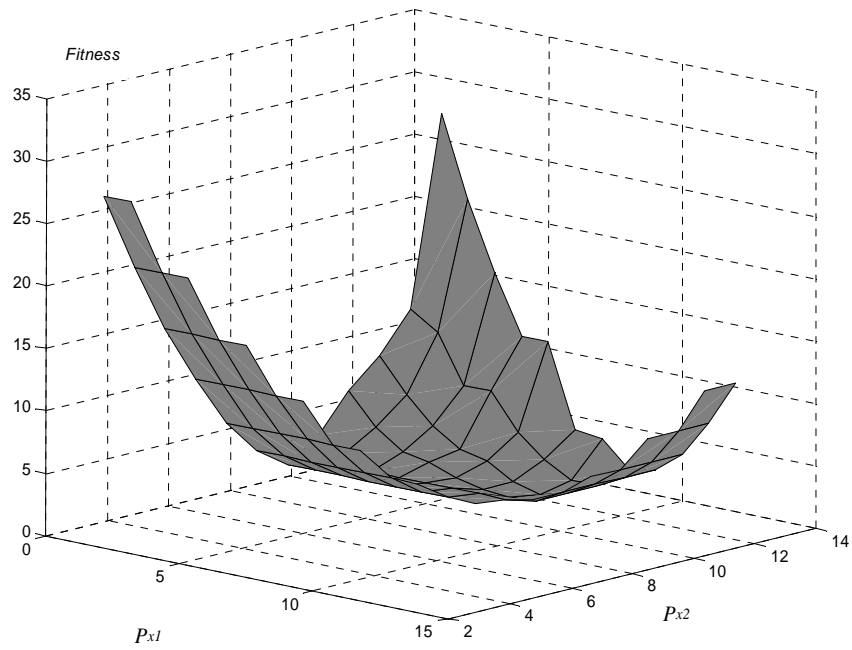


Figure 4.5. Comportamento da função de aptidão com pontos de quebra sobre a função quantizada, com parâmetros $N = N_Y = 4$, $N_T = 6$ e $m = 4$.

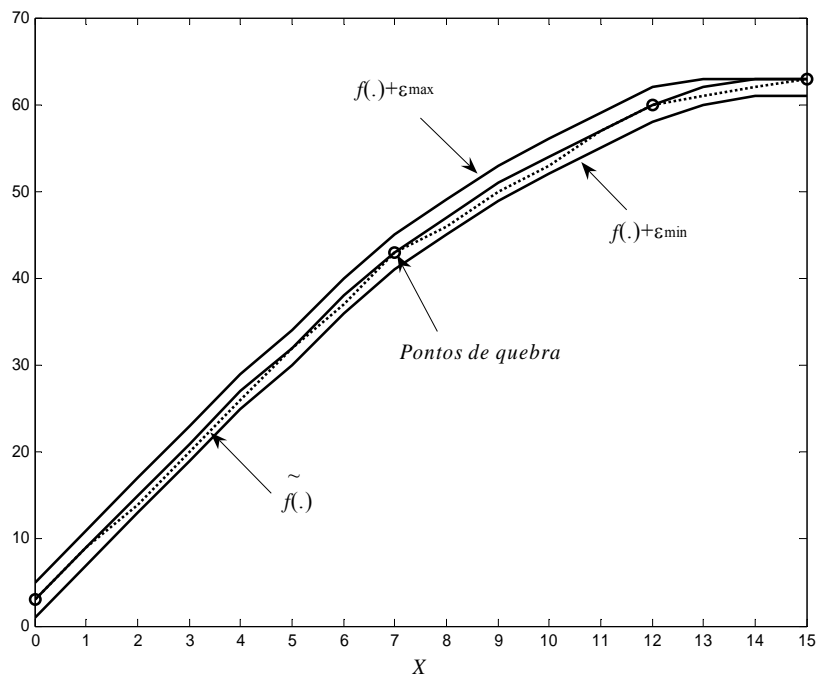


Figura 4.6. Função seno no primeiro quadrante aproximada por funções lineares por partes.

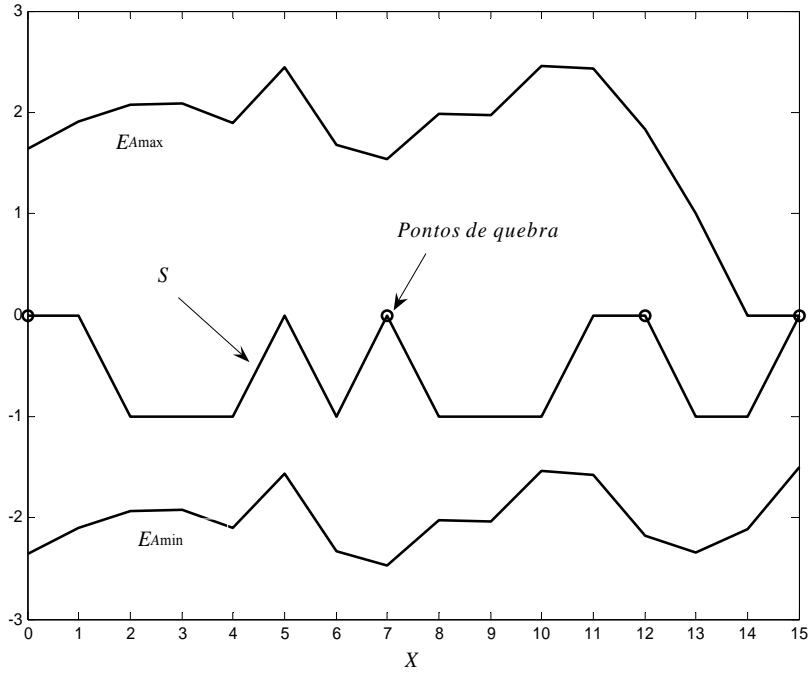


Figura 4.7. Erro de aproximação S e limites de erros de aproximação E_{Amin} e E_{Amax} normalizados.

Seguidamente, considera-se que os pontos p_{y1} e p_{y2} podem assumir qualquer valor inteiro que estejam dentro dos limites do erro de aproximação. Neste caso, a função de aptidão é calculada com a determinação das posições de quatro variáveis: p_{x1} , p_{x2} , p_{y0} , p_{y1} , p_{y2} e p_{y3} . Devido à dificuldade de representar o comportamento da função de aptidão em função de quatro variáveis, realiza-se uma codificação de varias variáveis para cada eixo, através das seguintes equações:

$$X = p_{x1} + \frac{(p_{y1} + r/2)}{(r+1)} + \frac{(p_{y0} + r/2)}{(r+1)^2} \quad (4.11)$$

$$Y = p_{x2} + \frac{(p_{y2} + r/2)}{(r+1)} + \frac{(p_{y3} + r/2)}{(r+1)^2} \quad (4.12)$$

sendo $r = 2^{(N_r - N_y)}$.

Na Figura 4.8 apresenta-se o comportamento da função de aptidão utilizando-se a transformação definida pelas equações (4.11) e (4.12). Observa-se nesta figura uma quantidade maior de combinações dos pontos de quebra, devido a que os pontos p_y assumem valores dentro do limite de erro normalizado. Neste caso, o valor mínimo da função de aptidão é de $fit_{min} = 0,3094$, com as soluções das posições dos pontos de quebra em: $p_x = [0, 7, 12, 15]$ e $p_y = [3, 43, 61, 63]$.

Na Figura 4.9 apresenta-se a solução para o mínimo valor da função de aptidão, a função é aproximada por funções lineares por partes. Na Figura 4.10 apresenta-se a solução da função aproximada utilizando-se a normalização, observa-se que os pontos de quebra p_y não se encontram sobre o eixo zero como o caso anteriormente estudado, isto porque os pontos de quebra não necessariamente estão determinadas sobre a função quantizada y_q .

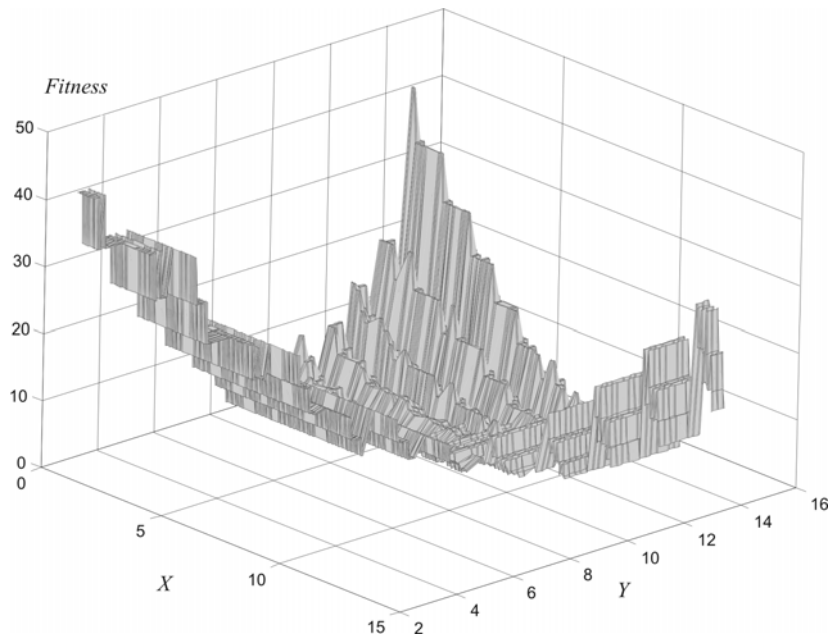


Figura 4.8. Comportamento da função de aptidão com parâmetros $N = N_Y = 4$, $N_T = 6$ e $m = 4$.

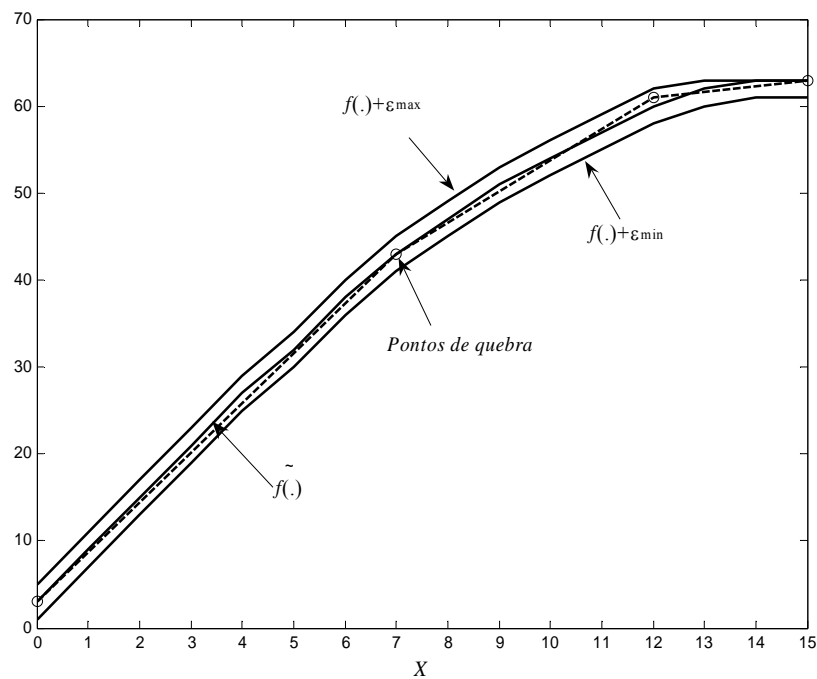


Figura 4.9. Função seno no primeiro quadrante aproximada por funções lineares por partes.

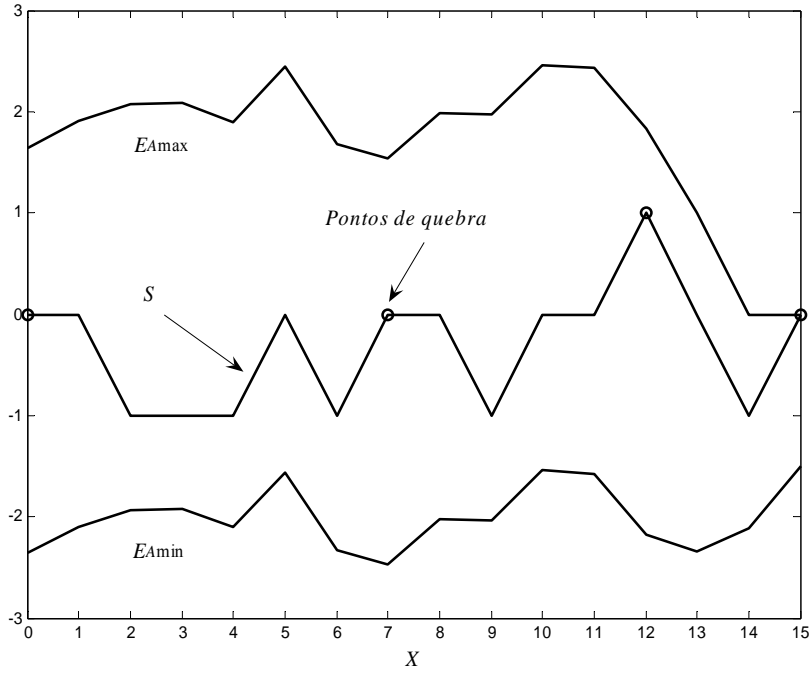


Figura 4.10. Erro de aproximação S e limites de erros de aproximação E_{Amin} e E_{Amax} normalizados.

Método de Seleção

O método da seleção envolve a escolha dos indivíduos na população que irão gerar descendentes, assim como o número de descendentes a serem criados. A partir da população principal n_{ind} indivíduos são selecionados para gerar novos cromossomos, utilizando uma pontuação baseada em atributos não lineares e seleção através de roleta [21], [24].

$$Prob(i) = \frac{n_{ind} \cdot v^{n_{ind}-1}}{\sum_j v^{j-1}} ; i = 1, \dots, n_{ind} \quad (4.13)$$

Sendo v a raiz real e positiva do polinômio:

$$(SP - n_{ind}) \cdot v^{n_{ind}-1} + SP \cdot v^{n_{ind}-2} + \dots + SP = 0 \quad (4.14)$$

e SP é o parâmetro que controla a pressão de seleção.

Considera-se um exemplo com $n_{ind} = 40$, o indivíduo com melhor fit é alocado na posição 1 e o indivíduo com pior fit é alocado na posição 40, a probabilidade de seleção $prob$ de cada indivíduo na população é calculado pela equação (4.13), sendo que o valor da probabilidade depende da posição de cada indivíduo na população, como apresentado na Figura 4.11.

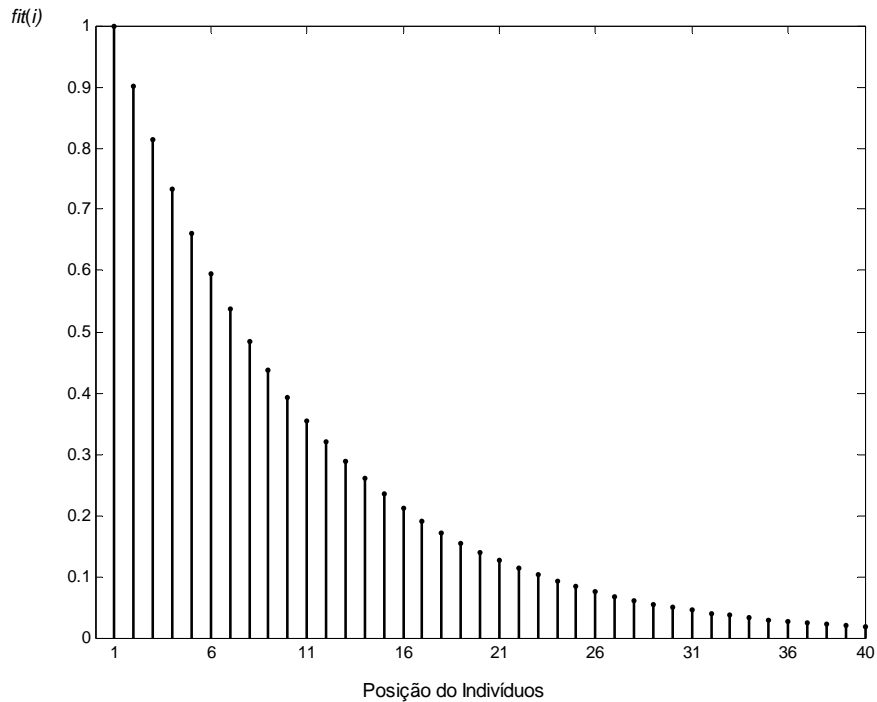


Figura 4.11. Probabilidade de seleção baseada na classificação não linear

Cruzamento dos Indivíduos

O passo seguinte, depois do processo de seleção, é aplicar a operação de cruzamento nos indivíduos da população. O cruzamento troca parte da informação genética entre dois indivíduos, produzindo novas soluções potenciais com algumas características dos pais.

Neste algoritmo genético aplica-se o operador de cruzamento binário simples (de um ponto) em que os cromossomos genitores são quebrados em uma determinada posição (escolhida aleatoriamente), sendo os segmentos resultantes permutados entre si, formando assim dois descendentes.

Neste algoritmo genético os cromossomos foram definidos utilizando-se uma codificação inteira, assim, para a aplicação do operador de cruzamento binário simples, requer-se fazer o mapeamento da representação do cromossomo de inteiro para binário. Conseqüentemente, na representação binária o cromossomo tem um comprimento em bits de $m \times (N + N_T)$. A Tabela 4.2 contém os parâmetros e o tamanho em bits do cromossomo. Observa-se que quando as especificações do problema são maiores o tamanho do cromossomo em bits é muito grande. Isto é um inconveniente devido ao esforço computacional necessário para realizar a operação de cruzamento.

Tabela 4.2. Tamanho do cromossomo em bits

N	N_T	m	Tamanho em bits
4	7	4	44
8	10	9	162
9	11	13	260
10	12	17	374
11	14	22	550
12	15	31	837

Para diminuir o esforço computacional do cruzamento binário, propõe-se utilizar a mesma representação inteira para realizar o operador de cruzamento. A seguir, apresenta-se um exemplo da operação de cruzamento utilizando a representação inteira e seu equivalente em representação binária. Definam-se dois genes G_1 e G_2 , com resoluções equivalentes a N bits, respectivamente. O cruzamento de G_1 e G_2 utilizando a representação inteira efetua-se da seguinte maneira:

Representam-se os genes G_1 e G_2 em forma binária e seu equivalente em forma inteira, e escolhe-se um ponto de cruzamento na posição $N - 2$.

$$\begin{array}{l}
 \begin{array}{c} \text{Ponto de} \\ \text{Cruzamento} \\ \downarrow \end{array} \\
 G_1 = a_{N-1} a_{N-2} a_{N-3} \dots a_2 a_1 a_0 = \text{floor}\left(\frac{G_1}{2^{N-2}}\right) * 2^{N-2} + \text{mod}(G_1, 2^{N-2}) \\
 G_2 = b_{N-1} b_{N-2} b_{N-3} \dots b_2 b_1 b_0 = \text{floor}\left(\frac{G_2}{2^{N-2}}\right) * 2^{N-2} + \text{mod}(G_2, 2^{N-2}) \\
 \begin{array}{c} \uparrow \end{array}
 \end{array}$$

O resultado do cruzamento dos genes G_1 e G_2 produz dois filhos definidos por G'_1 e G'_2 :

$$\begin{array}{l}
 G'_1 = a_{N-1} a_{N-2} b_{N-3} \dots b_2 b_1 b_0 = \text{floor}\left(\frac{G_1}{2^{N-2}}\right) * 2^{N-2} + \text{mod}(G_2, 2^{N-2}) \\
 G'_2 = b_{N-1} b_{N-2} a_{N-3} \dots a_2 a_1 a_0 = \text{floor}\left(\frac{G_2}{2^{N-2}}\right) * 2^{N-2} + \text{mod}(G_1, 2^{N-2})
 \end{array}$$

sendo que o operador “floor” efetua um arredondamento de um número e o operador “mod” calcula o resíduo da divisão de dois números.

Desta maneira pode-se realizar a operação de cruzamento sem precisar fazer o mapeamento de números inteiros para binários.

Mutação dos Indivíduos

Depois da operação de cruzamento os indivíduos da população atual sofrem um

processo de mutação, com o objetivo de introduzir indivíduos com nova codificação na população, tendo como consequência uma maior exploração do espaço de busca.

Neste sentido faz-se um mapeamento de indivíduos de representação inteira para representação binária. Seguidamente aplica-se o operador de mutação binário trocando-se os valores binários de 1 para 0 e vice-versa. Na Figura 4.12 apresenta-se um exemplo de uma mutação binária para uma variável inteira. Para todos os cromossomos, define-se com que probabilidade (P_m) cada gene presente no cromossomo será modificado.

Antes da Mutação	114	1 1 1 0 0 1 0
Depois da Mutação	98	1 1 0 0 0 1 0


 Ponto de Mutação

Figura 4.12. Representação da Operação de Mutação

Seleção da Nova População

Uma vez concluída a seleção probabilística baseada na posição através do método da roleta, processo de reprodução, os melhores indivíduos são selecionados para compor uma nova população. Este processo de seleção é do tipo $(\mu + \lambda)$ -ES (ES- Estratégias Evolutivas) [17], sendo que μ indivíduos produzem λ filhos. A nova população temporária de $(\mu + \lambda)$ indivíduos é reduzida por um processo de seleção determinística, em que os μ melhores indivíduos passarão definitivamente para a próxima geração.

4.3.3. Algoritmo Híbrido

Um dos principais inconvenientes na aplicação de algoritmos genéticos simples para problemas com espaços de busca complexos é seu alto custo computacional devido a sua baixa taxa de convergência. Neste panorama, de forma a melhorar o desempenho do algoritmo genético padrão e tornar mais rápido o processo de convergência, o algoritmo genético padrão é modificado em sua estrutura com o incremento de um algoritmo de programação matemática. A razão de utilizar uma estratégia híbrida é que um algoritmo híbrido pode combinar os méritos do algoritmo genético padrão de busca de soluções globais no espaço de soluções com uma técnica de programação matemática de busca local. Métodos de busca local podem melhorar potencialmente as soluções descobertas pelo algoritmo genético padrão durante o processo evolutivo, conseguindo soluções próximas do ótimo com uma maior taxa de convergência.

Neste sentido, desenvolveu-se um algoritmo híbrido composto pela combinação de um algoritmo genético padrão que fornece uma estrutura apropriada da solução e um algoritmo de programação matemática que determinara a estrutura de super-indivíduos a partir da população atual [22]. A seguir, apresenta-se sumariamente o algoritmo híbrido proposto:

Algoritmo Híbrido

Início

- $t \leftarrow 0$;
- Inicialize $P(t)$;
- Avalie $P(t)$;

Enquanto (não condição de parada) Faça

Início

- $t \leftarrow t+1$;
- Selecione $P(t)$ a partir de $P(t-1)$;
- Cruzamento $P(t)$;
- Mutação $P(t)$;
- Avalie $P(t)$;
- A partir de $P(t)$ gere super-indivíduos e o insere em $P(t)$;

Fim

- Retorna a melhor solução;

Fim

O algoritmo híbrido combina a capacidade de busca local da programação matemática com as propriedades de busca global do algoritmo genético padrão. Conseqüentemente a principal diferença entre o algoritmo genético padrão e o algoritmo híbrido se encontra na aplicação do algoritmo de programação matemática, responsável pela criação de super-indivíduos a partir da população atual de indivíduos. Uma vez criados os super-indivíduos, estes competirão por sua sobrevivência com os indivíduos da população atual introduzindo novas soluções à população atual com uma melhora do processo de exploração [31].

Devido a que os algoritmos híbridos aplicam técnicas de busca local às soluções geradas pelo algoritmo genético padrão, o gasto computacional do tempo de busca das soluções é incrementado durante o processo de busca genética.

Algoritmo de programação matemática

Para cada geração, super-indivíduos são gerados a partir da população atual $P(t)$ alterada pelo processo de seleção e reprodução. As novas soluções serão inseridas e substituirão aos piores indivíduos da população $P(t)$. Cada super-indivíduo é construído através de um algoritmo de busca local a partir da população atual. Utiliza-se um método simples de procura local que avalia o conjunto atual de dados para determinar uma

promissória direção de busca [31].

O procedimento do algoritmo de busca local consiste em ajustar as posições dos pontos de quebra das soluções da população atual $P(t)$. Considera-se o ajuste de uma solução de m pontos de quebra $(p_1, p_2, \dots, p_i, \dots, p_m)$, define-se uma nova solução obtida pelo algoritmo de busca local denotada por $(p_1, p_2, \dots, p'_i, \dots, p_m)$, sendo p'_i o novo ponto de quebra ajustado pelo algoritmo de busca local que substituirá p_i depois do resultado da avaliação da função de aptidão [32]. Para realizar o ajuste de um ponto de quebra p_i , o seguinte procedimento de busca é descrito sumariamente:

Passo 1: Especificar um ponto de quebra p_i para ajuste;

Passo 2: Gerar dois novos pontos a partir de p_i utilizando-se as equações seguintes:

$$p'_i = p_i + \gamma(p_{i+1} - p_i)$$

$$p''_i = p_i - \beta(p_i - p_{i-1})$$

sendo γ o coeficiente de expansão ($0 \leq \gamma \leq 1$) e β o coeficiente de contração ($0 \leq \beta \leq 1$).

Passo 3: Avaliar a função de aptidão para cada ponto de quebra p'_i e p''_i .

Passo 4: Substituir p_i pelo ponto de quebra p'_i ou p''_i que apresente menor valor da função de aptidão.

Passo 5: Voltar ao Passo 1 e selecionar o seguinte ponto de quebra para ajuste.

sendo γ e β variáveis aleatórias que toma valores no intervalo $[0, 1]$.

O novo ponto de quebra gerado ao redor do ponto de quebra p_i , que pode ser criado a partir: da operação de expansão ($p'_i \in \langle p_i, p_{i+1} \rangle$) ou da operação de contração ($p''_i \in \langle p_{i-1}, p_i \rangle$). Esta flexibilidade permite ao algoritmo de busca explorar o espaço de busca com uma maior liberdade. Isto também facilita uma sintonização das soluções ao redor de um ponto ótimo.

Capítulo 5

Resultados

Neste capítulo apresentam-se os principais resultados obtidos utilizando a estrutura do algoritmo hierárquico evolutivo desenvolvido no capítulo anterior. Para validar o algoritmo proposto apresenta-se um estudo de caso da aproximação da função seno no primeiro quadrante. Inicialmente apresentam-se os resultados utilizando o algoritmo genético padrão. Seguidamente os resultados utilizando a implementação do algoritmo híbrido composto por um algoritmo genético padrão e uma técnica de programação matemática. Apresenta-se uma comparação computacional das soluções do estudo de caso, entre o algoritmo genético e o algoritmo híbrido. Realiza-se uma análise exploratória do comportamento dos operadores genéticos e do algoritmo de programação matemática de busca local, durante processo de evolução. Finalmente apresenta-se uma estratégia de variação de parâmetros dos operadores genéticos (probabilidade de cruzamento e mutação), para realizar uma sintonia dos parâmetros melhorando o desempenho dos algoritmos propostos.

5.1. Estudo de caso: Aproximação da função seno no primeiro quadrante

Na atualidade, geradores digitais de funções são empregados em diversas áreas, como aplicações em radar, sistemas de comunicações, aplicações de controle digital, em instrumentação, etc. Neste sentido existe o interesse de viabilizar o problema de gerar valores de funções digitais utilizando-se sistemas embarcados de baixo custo com o objetivo de substituir geradores digitais convencionais e caros [28].

Nesta dissertação, para verificação dos algoritmos desenvolvidos, considera-se o estudo de caso de aproximar uma função seno no primeiro quadrante utilizando aproximação linear por partes. Para dar solução ao problema de aproximação da função seno, desenvolve-se um algoritmo hierárquico evolutivo descrito no capítulo anterior, que determinará os pontos de quebra e a memória mínima necessária de armazenamento para resolver o problema de aproximação linear de funções por partes considerando as especificações do projeto apresentadas no Capítulo 2. Os valores digitais da função seno são gerados a partir dos pontos de quebra que são armazenados em uma tabela de equivalência (LUT) do sistema embarcado.

5.2. Procedimento para solução

Na Seção 4.3. foi apresentado um algoritmo hierárquico evolutivo para determinar as posições dos pontos de quebra e a memória mínima necessária de armazenamento para aproximar funções utilizando aproximação linear por partes. O algoritmo proposto inclui a implementação de um algoritmo evolutivo e nessa dissertação é implementado independentemente com um algoritmo genético padrão e um algoritmo híbrido, como representado na Figura 5.1. Os algoritmos genéticos são executados independentemente para determinar as soluções da aproximação de funções considerando-se as especificações do projeto apresentadas no Capítulo 2. Os dois algoritmos genéticos são comparados com o fim de avaliar o desempenho durante o processo de busca das soluções.

O procedimento proposto possui dois níveis hierárquicos. No primeiro nível define-se o tamanho da tabela de equivalência ($m \times N_T$), com m pontos de quebra e uma resolução equivalente de N_T bits. No segundo nível implementa-se um algoritmo evolutivo que pode ser feito utilizando um algoritmo genético padrão ou um algoritmo híbrido que procurará uma solução factível para condição especificada no primeiro nível, como representado na Figura 5.1. Se a solução é encontrada, esta é repassada ao primeiro nível, que procederá reduzindo o tamanho da tabela de equivalência. Com esta nova condição, o segundo nível é novamente ativado. Se uma solução factível não é encontrada, considera-se como resultado a última solução encontrada [22].

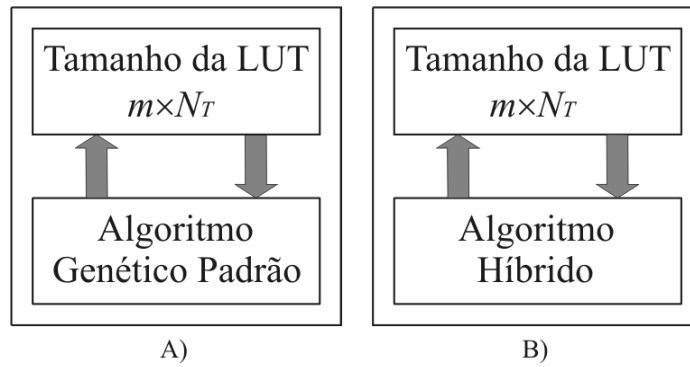


Figura 5.1. Algoritmo hierárquico evolutivo: A) Composto por um algoritmo genético padrão; B) Composto por um algoritmo híbrido.

A seguir apresentam-se os resultados do estudo de caso da aproximação da função seno no primeiro quadrante, utilizando a estrutura hierárquica evolutiva.

5.3. Resultados utilizando o Algoritmo Genético Padrão

A seguir apresentam-se os resultados computacionais obtidos para o estudo de caso da aproximação da função seno no primeiro quadrante. Considera-se a função seno definida como $y = \text{sen}(x)$, com limites $x \in [0, \pi/2]$ e $y \in [0, 1]$. Analisa-se o caso que não existe incerteza associada à variável livre, e para isso faz-se $N_Y = N$.

Os resultados apresentados com o uso desse algoritmo foram satisfatórios para casos com pequenas resoluções de N e N_Y até 10 bits. Entretanto, esse algoritmo apresentou problemas de convergência para resoluções maiores de 10 bits. Na Tabela 5.1 contém os parâmetros e resultados utilizados nas simulações e o tamanho mínimo da LUT (em bits) obtido para cada aproximação especificada. As simulações foram realizadas utilizando o programa MATLAB para um computador Pentium IV, 2.5GHz e memória de 512MB.

Tabela 5.1. Parâmetros e resultados da aproximação utilizando o algoritmo genético.

Parâmetros de entrada			Resultados			
N	N_Y	População	N_T	Pontos de quebra: m	Tamanho da LUT (bits)	Tempo computacional por época (s)
8	8	40	10	9	90	0,188
9	9	40	11	13	143	0,280
10	10	50	12	17	204	0,485

A seguir apresenta-se uma solução da aproximação da função seno, com parâmetros da simulação: $N = 8$, $N_Y = 8$ e $N_T = 10$. Neste caso utiliza-se uma probabilidade de cruzamento $P_c = 0,9$ e uma probabilidade de mutação $P_m = 0,04$. Determina-se um tamanho da população

inicial para 40 indivíduos. Nas simulações utilizou-se como critério de parada um número máximo de gerações de 500 ou quando a função de aptidão atinge um valor menor ou igual que um, indicando a existência na população de um indivíduo que representa uma solução para o problema, isto é, o erro de aproximação normalizado S se encontra dentro os limites de erro de aproximação E_{Amin} e E_{Amax} .

Na Figura 5.2 apresenta-se uma solução para a representação da aproximação da função seno, sendo $S = y - y_q$ o erro da função de aproximação normalizado, E_{Amin} e E_{Amax} os limites de erros de aproximação normalizados. O valor do *fitness* obtido foi de 0,9967. Observa-se que S não ultrapassa os limites de erros de aproximação.

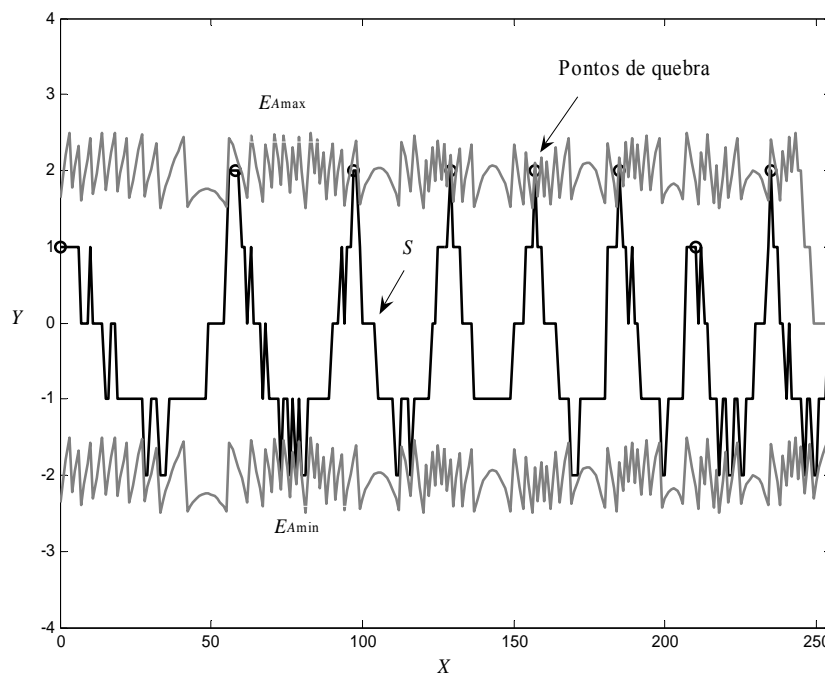


Figura 5.2. Erro de aproximação para a função seno com $N = 8$, $N_Y = 8$ e $N_T = 10$

A Tabela 5.2 contém os pontos de quebra (p_x, p_y) e o erro da função aproximada S encontrados pelo algoritmo hierárquico evolutivo utilizando o algoritmo genético padrão em sua estrutura.

Tabela 5.2. Pontos de quebra com parâmetros $N = 8$, $N_Y = 8$, $N_T = 10$ e $m = 9$.

Pontos de quebra	1	2	3	4	5	6	7	8	9
p_x	0	58	97	129	157	185	210	235	255
p_y	4	361	578	732	844	931	985	1017	1023
S	1	2	2	2	2	2	1	2	0

Um dos principais inconvenientes na aplicação de algoritmos genéticos simples para problemas com um grande espaço de busca é seu alto custo computacional devido a sua baixa taxa de convergência. Para diminuir esta dificuldade e aumentar a capacidade de exploração no espaço de busca, desenvolvem-se uma solução híbrida, com combinação de um algoritmo genético padrão e um método de programação matemática.

A seguir, apresentam-se os resultados para a aproximação da função seno utilizando-se a estrutura do algoritmo híbrido proposto.

5.4. Resultados utilizando o Algoritmo Híbrido

O algoritmo híbrido encontra-se composto por um algoritmo genético padrão e um algoritmo de programação matemática de busca local. Na Seção 4.3 foram descritos os detalhes que envolvem a implementação do algoritmo híbrido e o procedimento de busca local.

Considera-se o estudo de caso da aproximação da função seno no primeiro quadrante utilizando-se o algoritmo genético híbrido considerando-se o caso que não existe incerteza associada à variável livre, isto é $N_Y = N$.

Os resultados apresentados com o uso do algoritmo híbrido foram satisfatórios, conseguindo encontrar soluções para resoluções de N e N_Y de até 12 bits, respectivamente. Na Tabela 5.3 contém os parâmetros e resultados utilizados nas simulações e o tamanho mínimo da LUT (em bits) obtido para cada aproximação especificada. Em cada caso utiliza-se uma probabilidade de cruzamento $P_c = 0,9$ e uma probabilidade de mutação $P_m = 0,04$. Para cada especificação de parâmetros determina-se um número de indivíduos da população. As simulações foram realizadas utilizando o programa MATLAB para um computador Pentium IV, 2.5GHz e memória de 512MB.

Tabela 5.3. Parâmetros e resultados da aproximação utilizando o algoritmo híbrido.

Parâmetros de entrada			Resultados			
N	N_Y	População	N_T	Pontos de quebra: m	Tamanho da LUT (bits)	Tempo computacional por época (s)
8	8	40	10	9	90	0,188
9	9	40	11	13	143	0,280
10	10	50	12	17	204	0,485
11	11	60	14	22	308	1,93
12	12	80	15	31	465	6,15

A seguir, apresentam-se os resultados da aproximação da função seno com parâmetros $N = 8$, $N_Y = 8$ e $N_T = 10$. Na Figura 5.3 apresenta-se uma solução para a representação da

aproximação da função seno, sendo $S = y - y_q$ o erro da função de aproximação normalizado, E_{Amin} e E_{Amax} os limites de erros de aproximação normalizados. O valor do *fitness* obtido foi de 0,9627. Observa-se que a solução S não ultrapassa os limites de erros de aproximação.

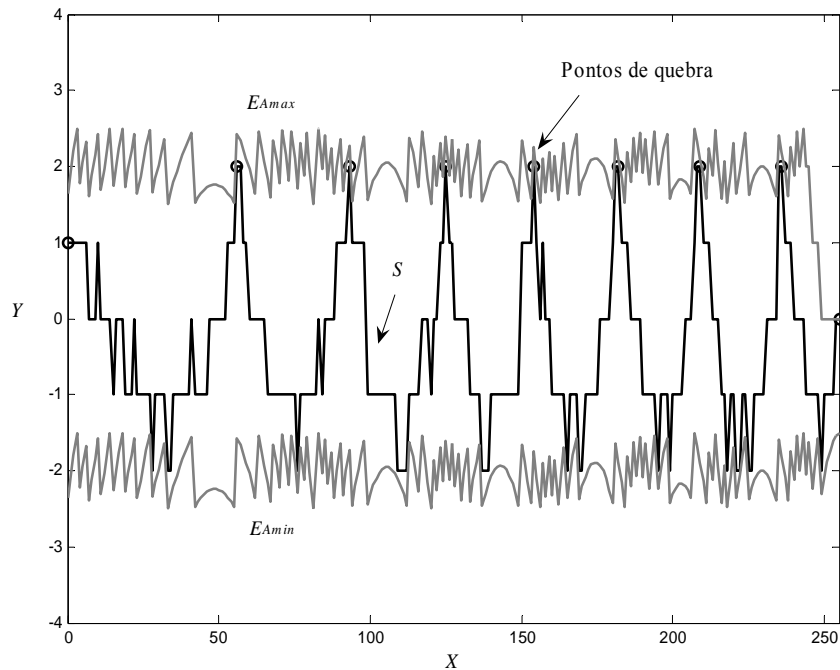


Figura 5.3. Aproximação para a função seno com $N = 8$, $N_Y = 8$ e $N_T = 10$.

A Tabela 5.4 contém os pontos de quebra (p_x, p_y) e o erro da função aproximada S encontrado pelo algoritmo hierárquico evolutivo utilizando o algoritmo genético híbrido.

Tabela 5.4. Pontos de quebra para $N = 8$, $N_Y = 8$ e $N_T = 10$.

Pontos de quebra (m)	1	2	3	4	5	6	7	8	9
p_x	0	56	93	125	154	182	209	236	255
p_y	4	349	557	714	833	923	984	1018	1023
S	1	2	2	2	2	2	2	2	0

5.5. Comparação Computacional entre o Algoritmo Genético Padrão e o Algoritmo Híbrido

A seguir, compara-se o desempenho da taxa de convergência entre o algoritmo genético padrão e algoritmo híbrido. Para isto, determinam-se os parâmetros N e N_Y , desde 8 bits até 12 bits como é especificado na Tabela 5.5. Para cada caso foram feitas 100 simulações independentemente para ambos algoritmos, determinando-se os valores da média final de épocas e desvio médio padrão em que são encontradas as soluções. Em cada caso utiliza-se

uma probabilidade de cruzamento $P_c = 0,9$ e uma probabilidade de mutação $P_m = 0,04$. A escolha de estes valores de probabilidades de P_c e P_m foi feita através de uma análise de variação da probabilidade de cruzamento e mutação que será estudado na Seção 5.7.

Tabela 5.5. Comparação de taxa de convergência do algoritmo genético padrão e o algoritmo híbrido.

Parâmetros				Algoritmo Genético Padrão		Algoritmo Híbrido	
N	N_Y	N_T	m	Média de época final	Desvio padrão médio	Média de época final	Desvio padrão médio
8	8	10	9	383	56	62	35
9	9	11	13	769	42	40	1,8
10	10	12	17	1522	123	341	65
11	11	14	22	-	-	850	83
12	12	15	31	-	-	1522	194

Na Figura 5.4 apresentam-se os gráficos de convergência para uma simulação dos algoritmos padrão e híbrido, com parâmetros $N = 8$, $N_Y = 8$, $N_T = 10$ e $m = 9$. O algoritmo genético padrão converge à solução com 307 gerações, entretanto, o algoritmo híbrido converge à solução com 57 gerações. Os dois algoritmos são inicializados com os mesmos valores das probabilidades de cruzamento e mutação ($P_c = 0,9$ e $P_m = 0,04$) e o mesmo valor da semente aleatória, isto é, a geração da seqüência de números aleatórios é a mesma.

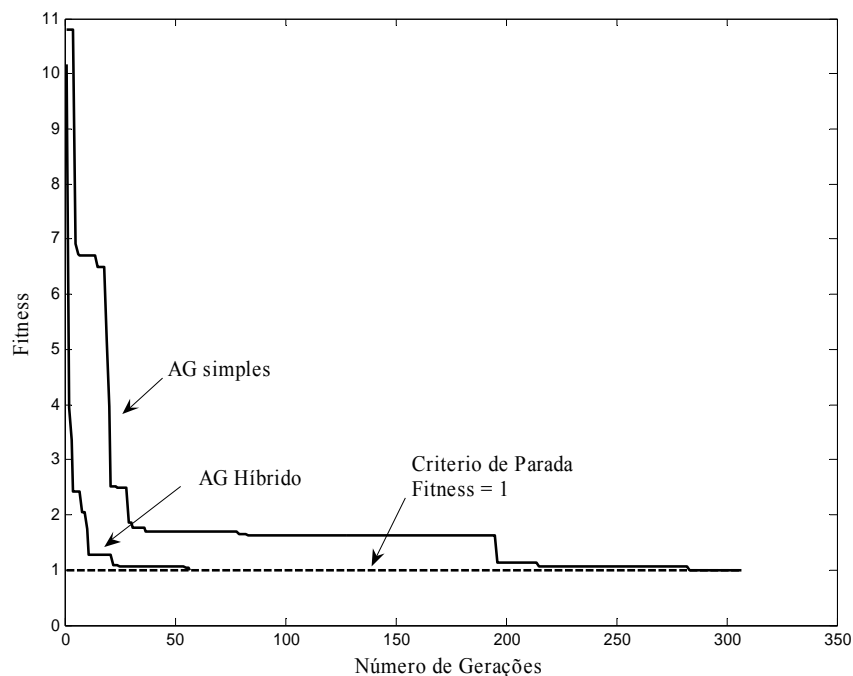


Figura 5.4. Simulação do Algoritmo Genético Padrão e Algoritmo Híbrido com parâmetros $N = 8$, $N_T = 10$ e $m = 9$.

Na Figura 5.5 apresenta-se a simulação com parâmetros $N = 9$, $N_Y = 9$, $N_T = 11$,

$m = 13$, $P_c = 0,9$ e $P_m = 0,04$. O algoritmo genético padrão converge à solução com 661 gerações, entretanto, o algoritmo híbrido converge à solução com 47 gerações. Neste caso também os dois algoritmos foram inicializados com o mesmo valor da semente aleatória.

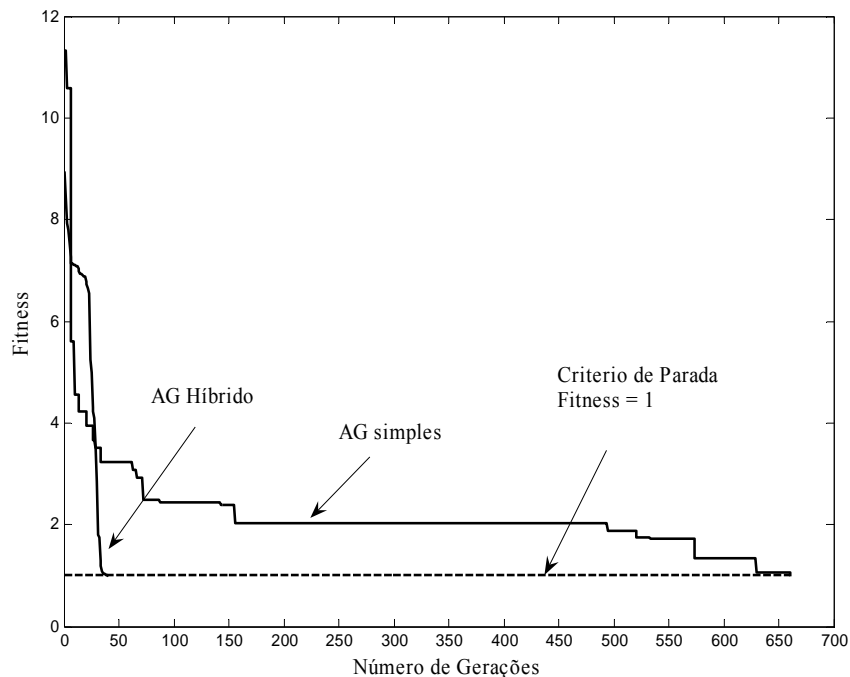


Figura 5.5. Simulação do Algoritmo Genético Padrão e Algoritmo Híbrido com parâmetros $N = 9$, $N_T = 11$ e $m = 13$

Os testes revelam que o algoritmo híbrido apresenta uma melhor taxa de convergência com respeito ao algoritmo genético padrão. Entretanto o algoritmo híbrido é mais lento que o algoritmo genético padrão, devido ao gasto computacional que se emprega para executar o algoritmo matemático de busca local.

5.6. Análise exploratória dos operadores genéticos e o algoritmo de programação matemática durante o processo evolutivo

No processo de implementação do algoritmo genético requer-se escolher um conjunto de operações genéticas entre muitas possibilidades. Por exemplo, a operação de cruzamento com um ponto de corte, a mutação bit a bit, a seleção baseada em roleta. Esta escolha pode ser efetiva para um tipo de algoritmo genético mais não para outros. Neste sentido, realiza-se uma análise do desempenho dos operadores utilizados no algoritmo híbrido, como por exemplo:

- Diversidade genética da população;
- Pressão de seleção;

- Número de cruzamentos por geração;
- Número de mutações por geração;
- Quantificação da contribuição do algoritmo de programação matemática.

A seguir analisa-se o desempenho do algoritmo genético para o estudo de caso da aproximação da função seno, com parâmetros $N = 8$, $N_Y = 8$, $N_T = 10$ e $m = 9$.

Diversidade Genética da População

Define-se a medida da diversidade genética como gdm (*Genetic Diversity Measure*) [29], que pode ser calculada como a razão entre o valor médio e máximo da função de aptidão em cada geração, com intervalo de variação $0 \leq gdm \leq 1$. Quando o valor de gdm se encontra próximo do valor um indica que todos os indivíduos têm o mesmo código genético, isto é, eles representam a mesma solução no processo de evolução. Na Figura 5.6 apresenta-se a o comportamento da diversidade genética (gdm) durante o processo de evolução para a aproximação da função seno para $N = 8$ e $N_T = 10$. A medida do gdm pode ser utilizada como uma variável de controle do algoritmo genético para prever uma convergência prematura para uma solução mínima local. O valor do gdm pode ser utilizado para variar as taxas de cruzamento e mutação do algoritmo genético tal que sejam introduzidas novas características genéticas. Entretanto, neste estudo foi utilizado apenas para verificação.

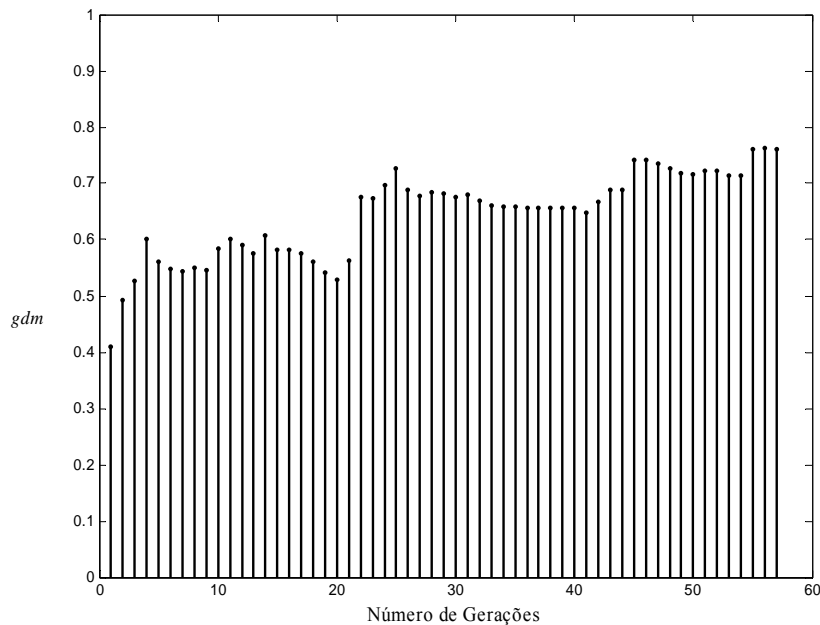


Figura 5.6. Medida da diversidade genética (gdm)

Pressão de Seleção

Neste algoritmo o processo de seleção encontra-se baseado em uma classificação não linear (tipo *rank*), com pressão de seleção $SP = 4$. Para um tamanho da população de 40 indivíduos a pressão de seleção é representada na Figura 5.7 para os pontos (p_x, p_y) , observa-se o comportamento não linear da pressão de seleção durante o processo evolutivo.

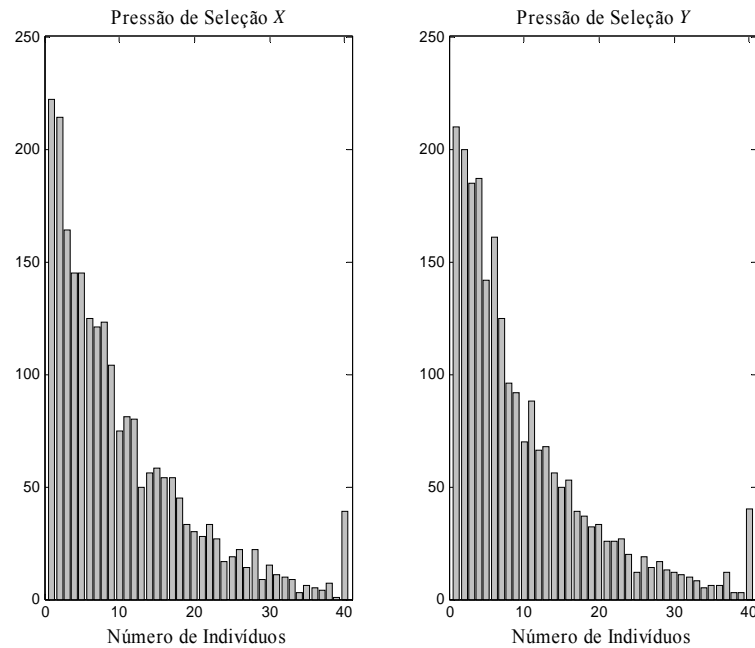


Figura 5.7. Pressão de seleção dos indivíduos p_x e p_y

Número de cruzamentos por geração

O algoritmo genético híbrido utiliza um operador de cruzamento simples de um ponto, na Figura 5.8 apresenta-se o número de cruzamentos produzidos pelos indivíduos da população em cada geração. Para a aproximação da função seno com parâmetros $N = 8$ e $N_T = 10$. Para um tamanho da população de 40 indivíduos, o valor médio de cruzamentos é de aproximadamente 36 em cada época.

Número de mutações por geração

O algoritmo implementado utiliza um operador de mutação bit a bit. Para uma população de 40 indivíduos e parâmetros $N = 8$, $N_T = 10$ bits e uma população de tamanho 40, tem-se 2880 bits em p_x e 3600 bits em p_y . Na Figura 5.9 apresentam-se o número de mutações para os pontos de quebra p_x e p_y , com um valor médio de 176 mutações em p_x e um valor médio de 48 mutações em p_y para cada geração respectivamente.

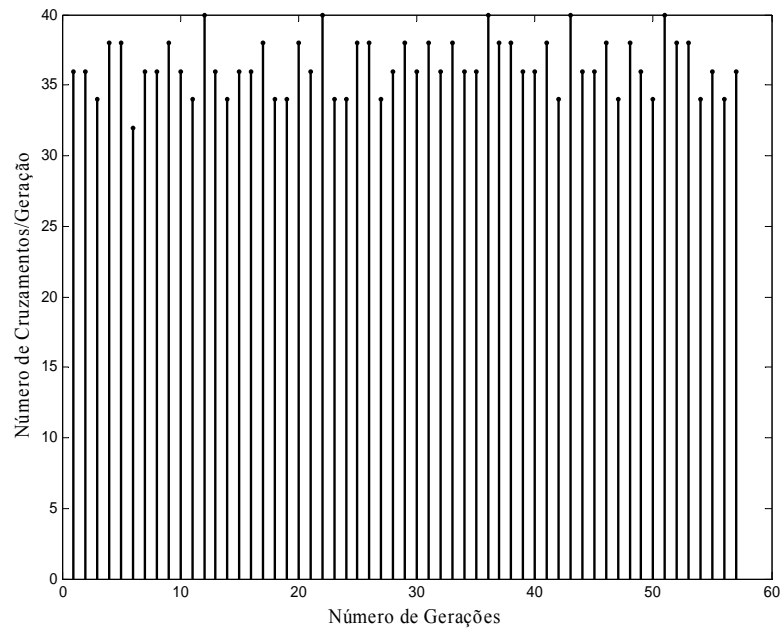


Figura 5.8. Número de cruzamentos em cada época.

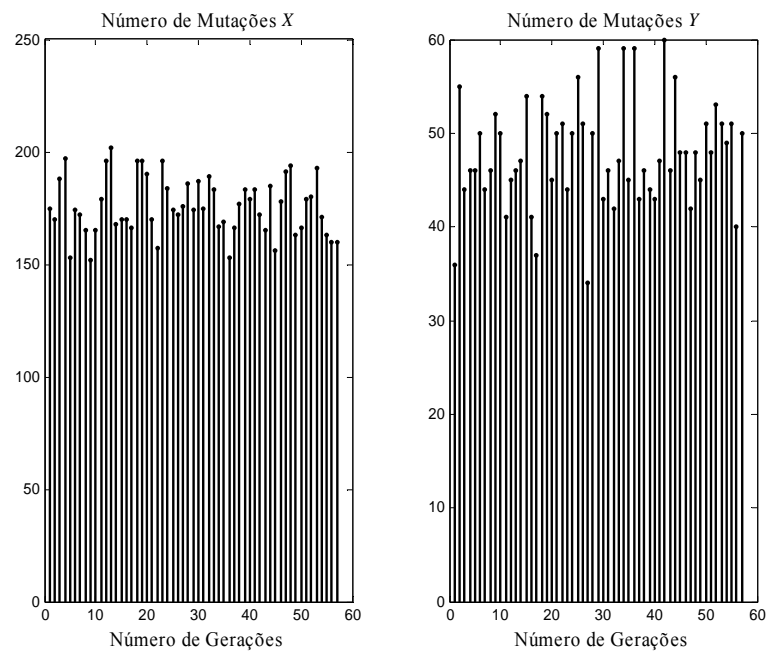


Figura 5.9. Número de mutações em cada geração para p_x e p_y .

Quantificação da contribuição do algoritmo de programação matemática

O algoritmo híbrido está composto por um algoritmo genético padrão que determina uma estrutura apropriada da solução e o algoritmo da programação matemática fornece novos indivíduos potenciais criados a partir da população atual, que serão inseridos na mesma. Para analisar a contribuição do algoritmo de programação matemática no processo de otimização,

deve-se determinar a quantidade de novos indivíduos que são introduzidos por época.

Na Figura 5.10 apresenta-se o comportamento das novos indivíduos introduzidos (indivíduos com o melhor *fitness*) pelo algoritmo de programação matemática em cada geração. Observa-se que a contribuição de novos indivíduos a partir deste algoritmo é não linear durante o processo de evolução. Neste estudo de caso, com parâmetros $N = 8$ e $N_T = 10$, o número total de novos indivíduos introduzidos pelo algoritmo de programação matemática foi de 560.

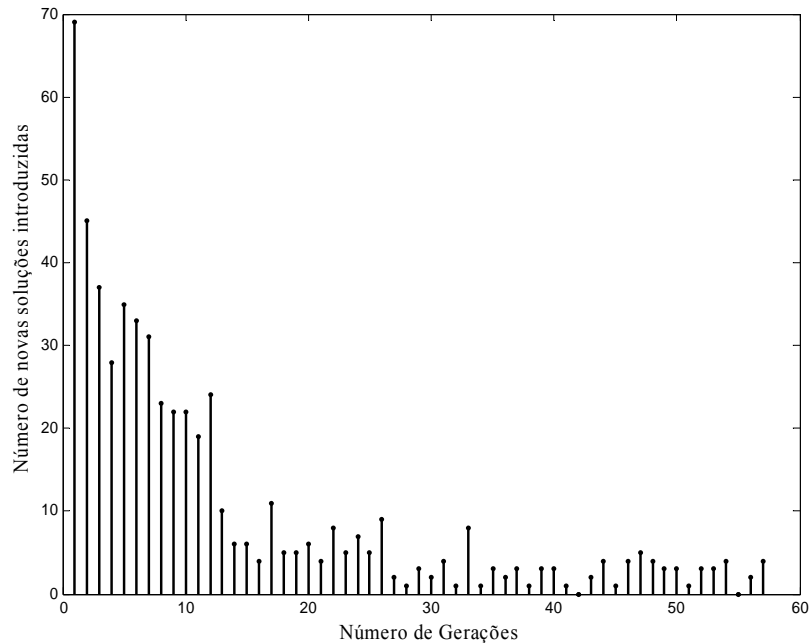


Figura 5.10. Número de novos indivíduos introduzidos pelo algoritmo de programação por época.

5.7. Estratégia de variação dos parâmetros: P_c e P_m

Na implementação de algoritmos genéticos requer-se escolha de um conjunto de parâmetros iniciais que tem uma grande influência no desempenho da evolução do algoritmo genético padrão, como por exemplo: tamanho da população inicial, número máximo de gerações, probabilidade de cruzamento e mutação. Qual a melhor escolha dos parâmetros, é uma importante questão. Para responder esta questão, muitos aperfeiçoamentos dos algoritmos genéticos podem ser realizados. Uma estratégia utilizada na referência [29], analisa o comportamento das soluções comparando a média de época final das soluções quando é variado as probabilidades de cruzamento (P_c) e a probabilidade de mutação (P_m).

Nesta análise a probabilidade de cruzamento é mantida constante enquanto variam os

valores da probabilidade de mutação e vice-versa. Assim também a semente aleatória é mantida constante para cada combinação de P_c e P_m . Na Figura 5.11 apresentam-se as variações das probabilidades de cruzamento e mutação, com parâmetros $N = 8$ e $N_T = 10$. Foram feitas 100 simulações para cada caso, determinando-se o valor da média das épocas finais em que encontra uma solução e desvio padrão de elas.

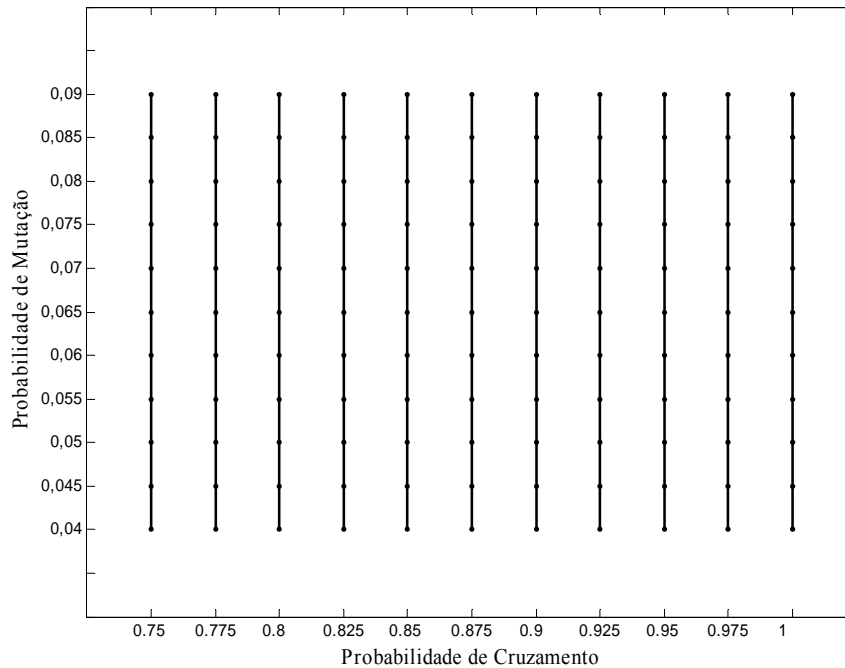


Figura 5.11. Variação das probabilidades de cruzamento e mutação (P_c, P_m).

A través de mecanismos de análise dos parâmetros dos algoritmos genéticos pode-se explorar o comportamento e a sintonia dos parâmetros com a finalidade de melhorar a taxa de convergência e o desempenho do algoritmo genético [30]. Nas Figuras 5.12 e 5.13 apresentam-se as variações das probabilidades de cruzamento e mutação, respectivamente. A resposta da curva de probabilidade de cruzamento sugere uma tendência linear, entretanto, a resposta da curva de probabilidade de mutação sugere uma tendência quadrática. A análise de exploração das relações entre a probabilidade de cruzamento e mutação permite entender o comportamento de estas variáveis e do alcance de valores que podem ser inicializados [29]. Destas figuras pode-se observar que uma alta probabilidade de cruzamento com uma baixa probabilidade de mutação determina uma rápida taxa de convergência do algoritmo proposto.

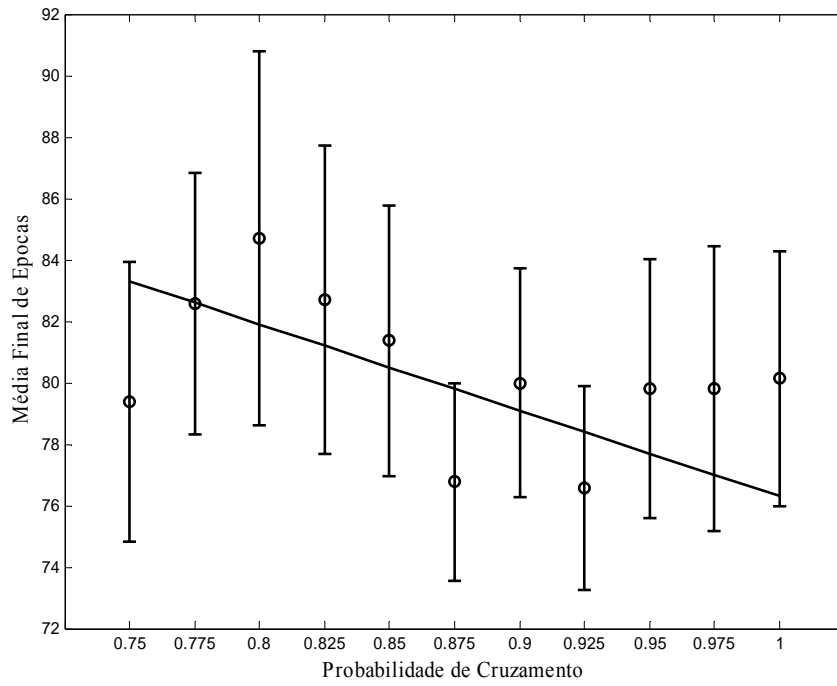


Figura 5.12. Resposta da curva de probabilidade de cruzamento para um total de 1100 simulações.

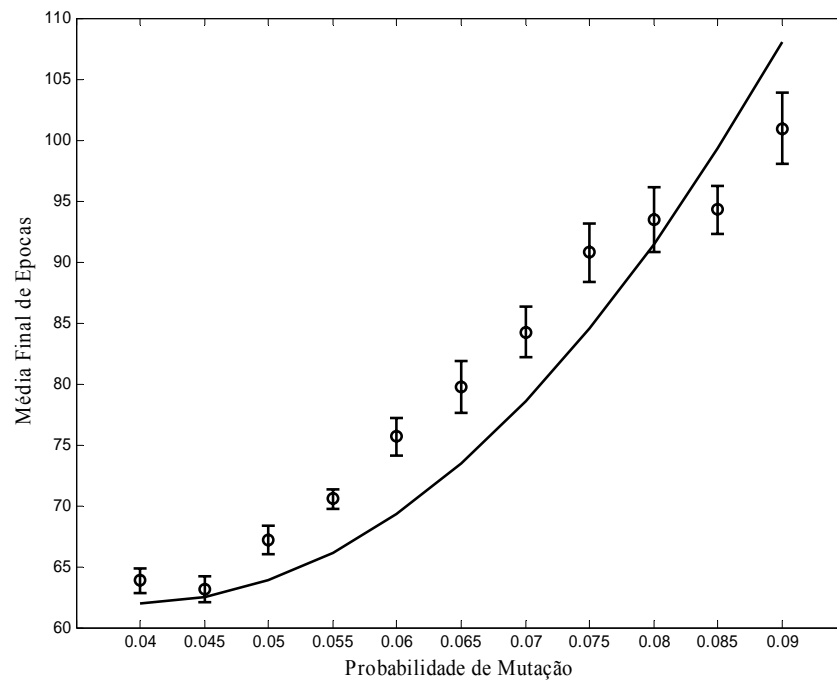


Figura 5.13. Resposta da curva de probabilidade de mutação para 1100 simulações.

Capítulo 6

Conclusões e Sugestões

6.1. Conclusões

Nesta dissertação foi apresentado um procedimento para determinar aproximação de funções não lineares para sistemas embarcados utilizando algoritmos genéticos, considerando-se aproximação de funções por funções lineares por partes e sistemas embarcados de baixo custo e com resolução limitada.

Neste procedimento, para obter os resultados foram tomadas as seguintes considerações:

- Para a implementação de aproximação de funções não lineares em sistemas embarcados de baixo custo com capacidade de cálculo em ponto fixo, foram considerados aspectos como: incertezas associadas as variáveis livres, saturação na representação de números e efeitos de quantização devido à resolução de armazenamento dos pontos de quebra. Para isto, foram analisados as restrições e limites para aproximação de funções não lineares em sistemas embarcados.
- A utilização de algoritmos genéticos padrão para a determinação de funções de aproximação mostrou ser uma alternativa viável para resolver este tipo de problemas. Os testes realizados mostraram que o algoritmo genético padrão apresenta um problema de convergência quando o espaço de busca de soluções é muito complexo, isto é, resoluções de N e N_Y maiores de 10 bits. Porém, uma

alternativa de solução para espaços de busca mais complexos foi o desenvolvimento de um algoritmo híbrido, composto por um algoritmo genético padrão e que emprega um algoritmo de programação matemática de busca local. O algoritmo híbrido mostrou um melhor desempenho na busca de soluções, conseguindo soluções próximas do ótimo com uma maior taxa de convergência, com resoluções de N e N_Y até 12 bits.

- Foram comparados as taxas de convergência do algoritmo genético padrão com o algoritmo híbrido através de um estudo de caso da aproximação da função seno no primeiro quadrante. Os diversos testes mostraram que o algoritmo híbrido apresenta uma melhor taxa de convergência.
- O algoritmo genético padrão foi modificado na sua estrutura por um algoritmo de programação matemática de busca local, introduzindo uma componente de exploração fina das soluções melhorando as soluções descobertas pelo algoritmo genético padrão.
- Foi realizada uma análise para a sintonia dos valores dos parâmetros de probabilidade cruzamento e mutação dos operadores genéticos. Uma ótima sintonia dos parâmetros do algoritmo genético tem como objetivo melhorar a taxa de convergência e o desempenho do algoritmo proposto.

6.2. Sugestões

Estudos complementares podem ser desenvolvidos, dentre os quais se podem destacar:

- Análise dos casos quando o algoritmo proposto não converge a uma solução.
- Estudar uma nova codificação dos pontos de quebra.
- Realizar uma interface gráfica para automatizar o processo de determinação de aproximação de função para sistemas embarcados.
- Análise do caso de determinação de aproximação de funções multivariáveis.

Referências Bibliográficas

- [1] ALAN BURNS, ANDY WELLINGS, Real-Time Systems and Programming Languages, 1996 Addison Wesley.
- [2] LIU, J.W.S, Real-Time Systems, 2000 by Prentice Hall, Inc., Upper Saddle River, New Jersey 07458.
- [3] VAHID, F., GIVARGIS, T., Embedded System Design: A Unified Hardware/Software Approach. Department of Computer Science and Engineering University of California, 1999.
- [4] CATUNDA, S.Y.C., SAAVEDRA, O.R., Constraints definition and evaluation of piecewise polynomial approximation functions for embedded systems, 19th IEEE Proceedings of Instrumentation and Measurement Technology Conference 2002, pp. 1103-1108 vol2.
- [5] CHUA, L.O, AN-CHANG DENG, Canonical Piecewise Linear Modeling, Circuits and Systems, IEEE Transactions on, Vol. 33, Issue: 5, May 1986, pp: 511-525.
- [6] MANIS, G., PAPAKONSTANTINOU, G., TSANAKAS, P., Optimal piecewise linear approximation of digitized curves, Digital Signal Processing Proceedings, 1997, DSP97, 13th International Conference on, Vol. 2, 2-4 July 1997, pp: 1079-1081.
- [7] JULIAN, P., JORDAN, M., DESAGES., A., Canonical piecewise linear approximation of smooth functions, Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, Vol. 45, Issue: 5, May 1998, pp: 567-571.
- [8] LYGOURAS, J.N., Memory reduction in look-up tables for fast symmetric function generators, Instrumentation and Measurement, IEEE Transactions on, Vol. 48, Issue: 6, Dec1999, pp: 1254-1258.
- [9] FLAMMINI, A., MARIOLI, D., TARONI, A., Application of an optimal look-up table to sensor data processing, Instrumentation and Measurement, IEEE Transactions on, Vol. 48, Issue: 4, Aug. 1999, pp: 813-816.
- [10] SHU-CHIEN HUANG, YUNG-NIEN SUN, Determination of Optimal Polygonal Approximation Using Genetic Algorithms, Evolutionary Computation Proceedings, 1998, IEEE World Congress on Computational Intelligence, The 1998 IEEE International Conference on, 4 – 9 May 1998, pp: 124 – 129.
- [11] HAUSER, J.W., PURDY, C.N., Approximating Functions for Embedded and ASIC Applications, Circuits and Systems, 2001. MWSCAS 2001. Proceedings of the 44th

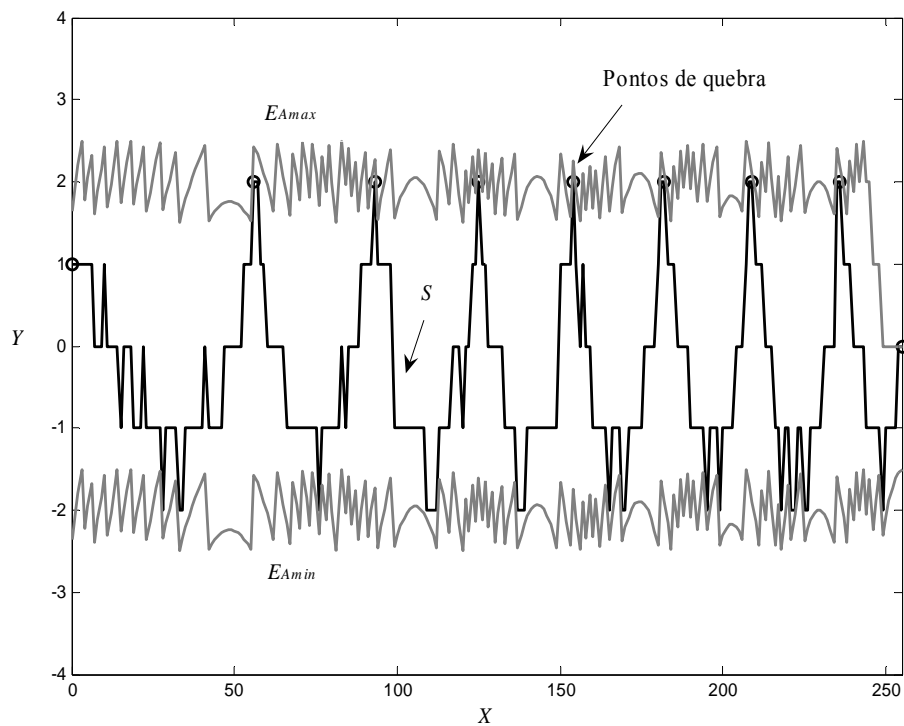
- IEEE 2001 Midwest Symposium on, Volume: 1, 14-17 Aug. 2001, pp: 478-481.
- [12] HAUSER, J.W., PURDY, C.N., Efficient Function Approximation for Embedded and ASIC Applications, Computer Design, 2001. ICCD 2001. Proceedings. 2001 International Conference on, 23-26 Set. 200, pp: 507-510.
- [13] CHUA, L.O., YING, R., Canonical piecewise-linear analysis, Circuits and Systems, IEEE Transactions on, Vol. 30, Issue: 3, Mar. 1983, pp: 125-140.
- [14] WEI-YEN WANG; TSU-TIAN LEE; CHING-LANG LIU; CHI-HSU WANG, Function Approximation using Fuzzy Neural Networks with Robust Learning Algorithm, Systems, Man and Cybernetics, Part B, IEEE Transactions on, Volume: 27, Issue: 4, Aug. 1997, pp: 740-747.
- [15] CATUNDA, S.Y.C., NAVINER, J. F., DEEP, G.S., FREIRE, R.C.S., Optimized look-up table for inter-mesurand compensation, Instrumentation and Measurement Technology Conference, 2001, IMTC 2001, Proceedings of the 18th IEEE, Vol. 1, 21-23 May 2001, pp: 128-132.
- [16] GOLDBERG, D.E., Genetic Algorithms in Search, Optimization, and Machine Learning, Reading Mass.: Addison – Wesley Pub. Co., 1989.
- [17] MICHALEWICZ, Z., Genetic Algorithms + DataStructures = Evolution Programs, Third, Revides and Extended Edition, Pub. Springer, 1996.
- [18] AHMED, M.A., DEJONG, K.A., Function Approximator design using Genetic Algorithms, Evolutionary Computation, IEEE International Conference on, 13-16 April 1997, pp: 519 – 524.
- [19] The MathWorks, Simulink Fixed Point, For Use With Simulink, User's Guide Version 5, 2004.
- [20] CATUNDA, S.Y.C., NAVINER, J.F., DEEP, G.S., FREIRE, R.C.S., Look-up table otimizada para reconstrução de valores de medição utilizando sensores não lineares. XIII Congresso Brasileiro de Automática, Florianópolis, Brasil, Setembro 2000.
- [21] CATUNDA, S.Y.C., SAAVEDRA, O.R., FONSECA NETO, J.V., e MORAIS, M.R.A., Determinação de tabela de equivalência e pontos de quebra de funções de aproximação lineares por partes usando computação evolutiva, 6º SBIA Simpósio Brasileiro de Automação Inteligente, 14-17 Setembro 2003, pp: 662-667
- [22] MAURICIO, J.M., CATUNDA, S.Y.C., SAAVEDRA, O.R., FONSECA NETO, J.V., Aproximação de funções: Uma abordagem utilizando computação evolutiva híbrida, 8th Simpósio Brasileiro de Redes Neurais, São Luis, Brasil, Setembro 2004, 3590.
- [23] FOGEL, D.B., An introduction to simulated evolutionary optimization, Neural Networks, IEEE Transactions on, Vol. 5, Issue :1, Jan. 1994, pp : 3-14.
- [24] HARTMUNT POHLHEIM, GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB <http://www.geatbx.com/docu/index.html>, acessado em 09/11/2004.
- [25] BACK, T., HAMMEL, U., SCHWEFEL, H.P., Evolutionary computation: comments on the history and current state, Evolutionary computation, IEEE Transactions on, Volume: 1, Issue: 1, April 1997, pp: 3 – 17.
- [26] VON ZUBEN, F.J., Computação Evolutiva: Uma Abordagem Pragmática, Notas de aula da disciplina IA707 DCA/FEEC/Unicamp/Brasil, 2000.
- [27] MICHALEWICZ, Z., Heuristic Methods for Evolutionary Computation Techniques, Journal of Heuristics, 1995, Vol. 1, N° 2, pp: 177-206.
- [28] YEARY, M.B., FINK, R.J., BECK, D., GUIDRY, D.W., BURNS, M., A DSP-based mixed-signal waveform generator, Instrumentation and Measurement, IEEE Transactions on, Volume: 53, Issue: 3, June 2004, pp: 665-671.
- [29] CZAN, A., MACNISH, C., VIJAYAN, K., TURLACH, B., GUPTA, R., Statistical Exploratory Analysis of Genetic Algorithms, Evolutionary Computation, IEEE

- Transactions on, Volume: 8, Issue: 4, Aug. 2004, pp:405-421.
- [30] CUI ZHI-HUA, ZENG JIAN-CHAO, XU YU-BIN, Convergence and calculation efficiency analysis of abstract model of nonlinear genetic algorithm based on function group, System, Man and Cybernetics, 2003. IEEE International Conference on, Volume 5, 5-8 Oct. 2003, pp: 4655-4659
 - [31] JOHN YEN, BOGJU LEE, A simples genetic algorithm hybrid, Evolutionary Computation, 1997, IEEE International Conference on, 13-16 April 1997, pp: 175-180.
 - [32] Yen, J., LIAO, J.C., BOGJU LEE, RANDOLPH, D., A hybrid approach to modeling metabolic systems using a genetic algorithm and simplex method, Systems, Man and Cybernetics, Part B, IEEE Transactions on, Volume : 28, Issue : 2, April 1998, pp: 173-191.

Apêndices

Apêndice A – Soluções da Aproximação da Função Seno

A1 Gráfico da aproximação da função seno com parametros $N = 8$, $N_Y = 8$, $N_T = 10$ e $m = 9$.



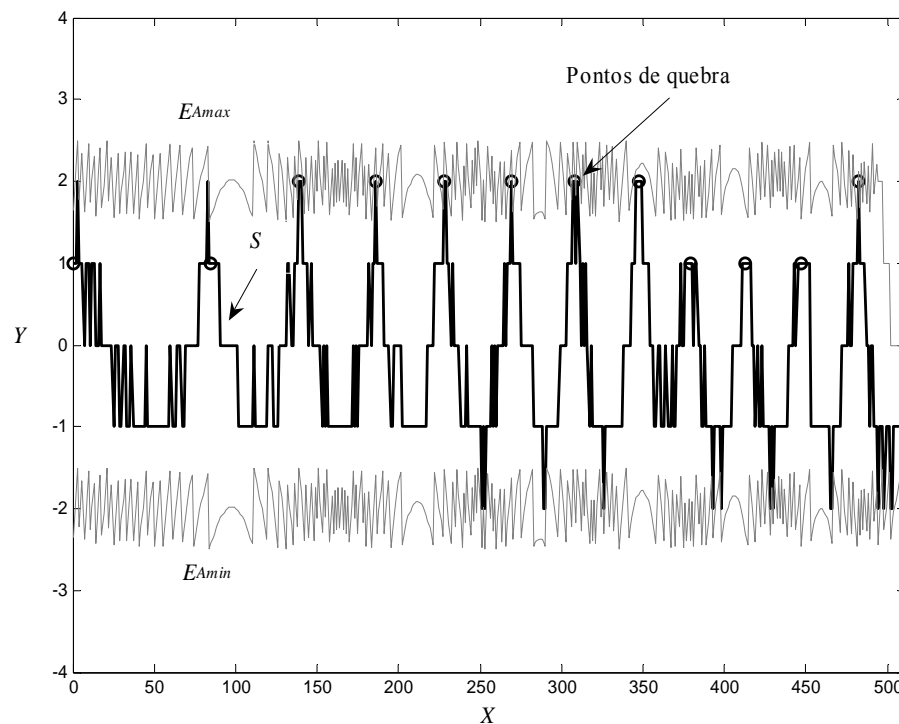
Pontos de quebra:

$$P_x = [0 \ 56 \ 93 \ 125 \ 154 \ 182 \ 209 \ 236 \ 255]$$

$$P_y = [4 \ 349 \ 557 \ 714 \ 833 \ 923 \ 984 \ 1018 \ 1023]$$

$$Fitness = 0,9627$$

A2 Gráfico da aproximação da função seno com parametros $N = 9$, $N_Y = 9$, $N_T = 11$ e $m = 13$.



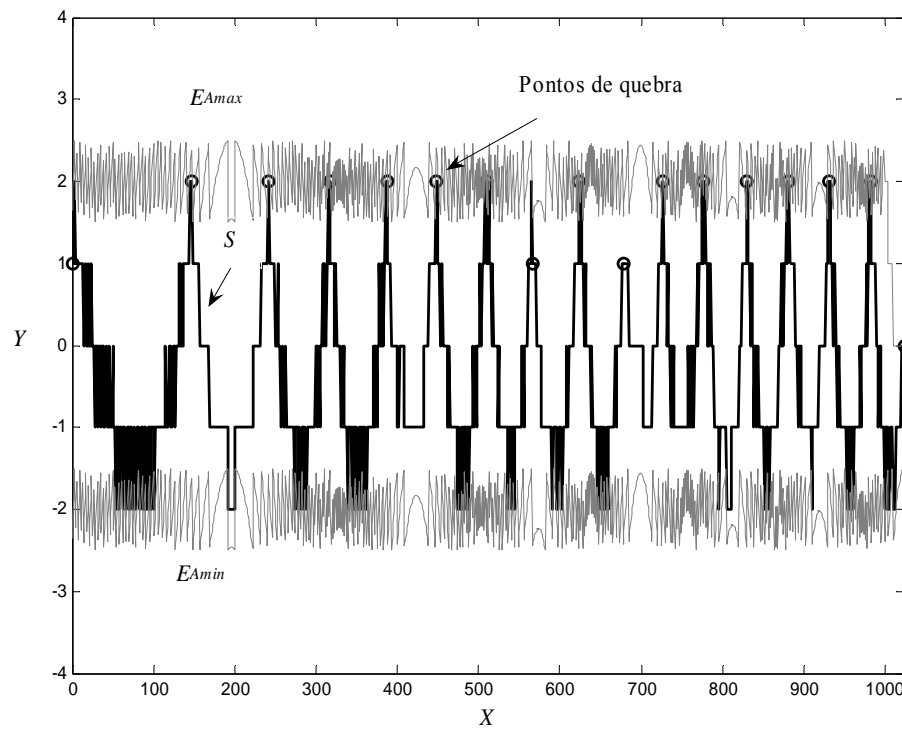
Pontos de quebra:

$$P_x = [0 \ 85 \ 139 \ 186 \ 228 \ 269 \ 308 \ 347 \ 379 \ 413 \ 447 \ 482 \ 511]$$

$$P_y = [4 \ 532 \ 851 \ 1110 \ 1322 \ 1508 \ 1663 \ 1794 \ 1882 \ 1956 \ 2009 \ 2041 \ 2047];$$

$$Fitness = 0,9051$$

A3 Gráfico da aproximação da função seno com parâmetros $N = 10$, $N_Y = 10$, $N_T = 12$ e $m = 17$.



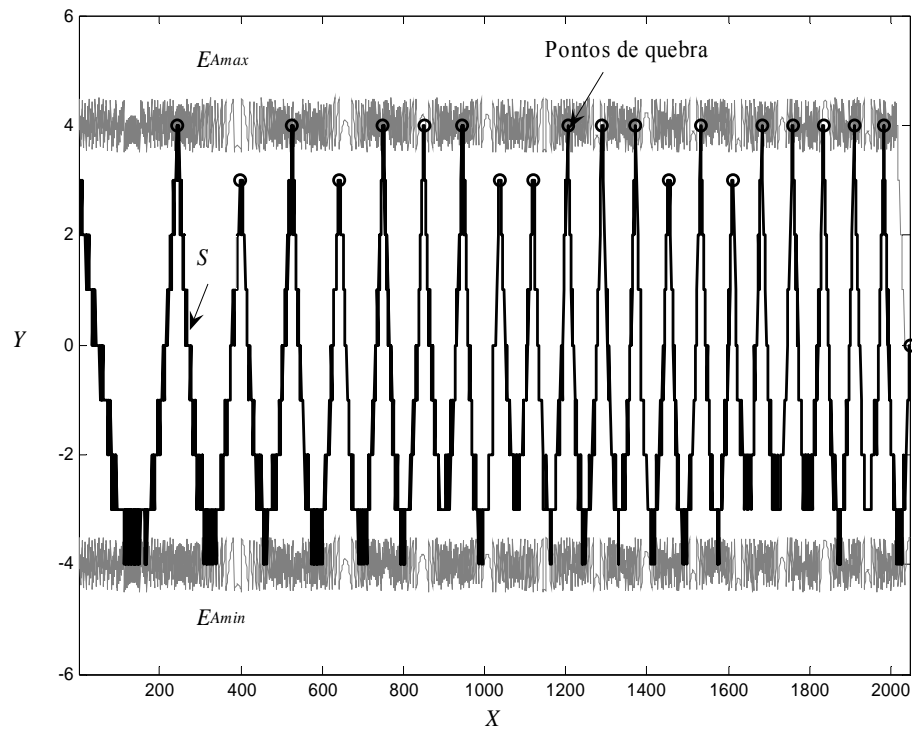
Pontos de quebra:

$P_x = [0 \ 147 \ 241 \ 317 \ 387 \ 448 \ 510 \ 567 \ 624 \ 679 \ 726 \ 777 \ 830 \ 881 \ 931 \ 982 \ 1023]$

$P_y = [4 \ 920 \ 1484 \ 1918 \ 2295 \ 2602 \ 2891 \ 3133 \ 3352 \ 3538 \ 3678 \ 3808 \ 3918 \ 4000 \ 4056 \ 4089 \ 4095]$

$Fitness = 0,9878$

A4 Gráfico da aproximação da função seno com parâmetros $N = 11$, $N_Y = 11$, $N_T = 14$ e $m = 22$



Pontos de quebra:

$P_x = [0 \ 245 \ 399 \ 526 \ 643 \ 750 \ 851 \ 946 \ 1038 \ 1119 \ 1205 \ 1290 \ 1372 \ 1453 \ 1533 \ 1611 \ 1683 \ 1759 \ 1834 \ 1910 \ 1983 \ 2047]$

$P_y = [9 \ 3070 \ 4945 \ 6441 \ 7765 \ 8922 \ 9959 \ 10880 \ 11716 \ 12405 \ 13084 \ 13699 \ 14237 \ 14713 \ 15128 \ 15477 \ 15751 \ 15988 \ 16168 \ 16296 \ 16367 \ 16383]$

$Fitness = 0,983$

Apêndice B – Variação dos Parâmetros do Algoritmo Genético

Variação da probabilidade de cruzamento e mutação

Na Tabela contém as variações das probabilidades de cruzamento e mutação para obter a Figura 5.11, o valor da média das épocas finais em que encontra uma solução e desvio padrão de elas. Assim também a porcentagem de acertos do algoritmo para converge a uma solução. Para cada combinação dos valores de P_c e P_m estabelecem-se 100 iterações, com parâmetros $N = 8$, $N_Y = 8$, $N_T = 10$ e $m = 9$.

P_c	P_m	Média de época final	Desvio padrão médio	Porcentagem de acertos (%)
0,75	0,04	63,315	3,4161	92
	0,045	57,722	3,0889	97
	0,05	63,684	4,1905	95
	0,055	74,117	4,2389	94
	0,06	72,406	6,4195	96
	0,065	75,158	4,5645	95
	0,07	88,688	6,1298	96
	0,075	88,355	5,9261	93
	0,08	90,365	7,3069	96
	0,085	92,892	7,2721	93
	0,09	106,57	9,5844	96
	0,04	69,939	4,2361	98
	0,045	67,531	3,6834	98
	0,05	67,292	3,8979	96
	0,055	70,447	4,3034	94
	0,06	77,66	4,7872	94

0,775	0,04	69,939	4,2361	98
	0,045	67,531	3,6834	98
	0,05	67,292	3,8979	96
	0,055	70,447	4,3034	94
	0,06	77,66	4,7872	94
	0,065	81,255	5,8423	94
	0,07	82,333	5,2041	93
	0,075	85,552	6,4834	96
	0,08	107,04	9,1649	95
	0,085	98,638	8,2827	94
	0,09	100,59	8,6137	95
0,8	0,04	57,516	3,1977	95
	0,045	64,876	3,6548	97
	0,05	68,105	3,9834	95
	0,055	68,516	3,8437	95
	0,06	67,063	4,1941	96
	0,065	92,129	7,5627	93
	0,07	98,865	6,6247	96
	0,075	110,46	8,8919	96
	0,08	89,427	6,5685	96
	0,085	99,305	8,298	95
	0,09	115,39	10,096	94
0,825	0,04	64,402	3,4831	97
	0,045	55,619	3,2299	97
	0,05	74,312	5,2315	93
	0,055	71,32	4,669	100
	0,06	78,177	6,5952	96
	0,065	81,753	5,661	97
	0,07	85,969	5,8861	98
	0,075	93,768	8,373	95
	0,08	96,596	6,6474	94
	0,085	92,551	7,2513	98
	0,09	115,12	9,7457	97
0,85	0,04	65,927	3,4163	96
	0,045	66,229	3,4154	96
	0,05	68,521	4,4528	94
	0,055	70,071	3,8215	98
	0,06	79,957	5,7372	93
	0,065	75,879	4,9103	99
	0,07	73,447	4,8812	94
	0,075	93,887	7,4784	97
	0,08	102,37	8,7117	95
	0,085	102,68	8,4652	95
	0,09	96,054	7,0537	92
0,875	0,04	61,063	3,1565	95
	0,045	65,978	3,6272	92
	0,05	63,925	3,6722	93
	0,055	75,379	4,9784	95
	0,06	67,907	4,9881	97
	0,065	75,865	5,6917	96
	0,07	85,442	6,6775	95
	0,075	88,677	6,938	96
	0,08	85,711	5,4498	97

	0,085	84,695	6,5615	95
	0,09	89,823	7,4485	96
0,9	0,04	62,06	3,4717	100
	0,045	64,274	3,1615	95
	0,05	68,978	3,7901	92
	0,055	70,588	3,627	97
	0,06	76,02	4,7017	99
	0,065	90,604	6,709	96
	0,07	85,278	6,6338	97
	0,075	85,453	5,1194	95
	0,08	83,274	6,6526	95
	0,085	95,732	7,7179	97
0,09	97,702	6,6367	94	
0,925	0,04	60,837	3,2958	98
	0,045	63,948	3,8805	96
	0,05	70,021	3,7062	97
	0,055	65,289	3,5304	97
	0,06	76,726	4,7923	95
	0,065	75,571	4,6152	98
	0,07	76,198	5,4539	96
	0,075	95,146	7,4591	96
	0,08	83,566	6,9372	99
	0,085	85,615	5,5916	96
0,09	89,49	6,5866	96	
0,95	0,04	65,817	3,3593	93
	0,045	61,99	3,5261	96
	0,05	63,28	3,8476	93
	0,055	71,204	5,0096	93
	0,06	73,51	6,4025	96
	0,065	79,071	5,7822	98
	0,07	83	5,7251	95
	0,075	91,167	7,2846	96
	0,08	91,115	5,3949	96
	0,085	104,17	9,203	95
0,09	93,531	8,2041	96	
0,975	0,04	65,553	3,8456	94
	0,045	62,926	3,644	95
	0,05	60,84	2,9639	100
	0,055	69,632	4,63	95
	0,06	83,429	5,5538	98
	0,065	67,378	4,1914	98
	0,07	88,66	6,886	97
	0,075	86,215	5,4237	93
	0,08	107,82	7,6412	96
	0,085	90,175	7,4618	97
0,09	95,102	7,6437	98	
1,0	0,04	65,897	3,8414	97
	0,045	63,293	3,0045	92
	0,05	69,979	3,5352	95
	0,055	69,474	4,5531	95
	0,06	79,468	4,7725	94
	0,065	82,355	5,03	93
	0,07	78,777	6,4463	94

	0,075	79,859	5,0548	99
	0,08	90,708	7,0226	96
	0,085	90,577	6,8343	97
	0,09	111,06	9,2632	97