UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

LEONARDO DE ASSIS SILVA

PID INTELIGENTE: SÍNTESE OTIMIZADA DE CONTROLADORES *FUZZY* VIA ALGORITMOS EVOLUTIVOS

VITÓRIA – ES
JULHO/2014

LEONARDO DE ASSIS SILVA

PID INTELIGENTE: SÍNTESE OTIMIZADA DE CONTROLADORES *FUZZY* VIA ALGORITMOS EVOLUTIVOS

Parte manuscrita do Projeto de Graduação do aluno **Leonardo de Assis Silva**, apresentada ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Ing. Edson De Paula Ferreira
Coorientador: Prof. MSc. Diego Nunes Bertolani

VITÓRIA – ES
JULHO/2014

LEONARDO DE ASSIS SILVA

PID INTELIGENTE: SÍNTESE OTIMIZADA DE CONTROLADORES *FUZZY* VIA ALGORITMOS EVOLUTIVOS

Parte manuscrita do Projeto de Graduação do aluno **Leonardo de Assis Silva**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovada em 31, de julho de 2014.

COMISSÃO EXAMINADORA:

_____

Prof. Dr. Ing. Edson De Paula Ferreira
Universidade Federal do Espírito Santo - UFES
Orientador

_____

Prof. MSc. Diego Nunes Bertolani
Instituto Federal do Espírito Santo - IFES
Coorientador

_____

Prof. Dr. Alessandro Mattedi
Universidade Federal do Espírito Santo - UFES
Examinador

_____

Prof. Dr. Wagner Teixeira da Costa
Instituto Federal do Espírito Santo - IFES
Examinador

Dedico este projeto primeiramente aos meus pais e meu irmão pelo amor, carinho e eterno apoio.
Aos meus grandes amigos e professores que tentaram fazer de mim um bom profissional, pela atenção, ajuda e paciência.
À minha amada namorada Carolina pelo seu carinho, amor, apoio e compreensão.
A Deus que sempre me iluminou e me guiou por todos os caminhos percorridos.

Agradeço primeiramente aos meus pais, meu irmão e à minha namorada Carol pelo amor, carinho, e eterno apoio e incentivo. Aos meus grandes amigos e a todos os professores que tentaram fazer de nós graduandos bons profissionais, pela atenção, ajuda e descontração em todos os momentos passados juntos. Agradeço também ao Prof. Dr. Ing. Edson de Paula Ferreira e ao Prof. MSc. Diego Nunes Bertolani pela orientação neste projeto e pela excelência como profissionais e pessoas. Por fim, a Deus que me deu saúde e iluminação, além de me permitir conhecer todos citados. Tenho certeza que sempre poderei contar com todos vocês. Um sincero obrigado a todos.

RESUMO

Na literatura, verifica-se um grande esforço no desenvolvimento de métodos de síntese de controladores práticos e com menor complexidade. Dentre eles, tem destacado relevo os baseados em inteligência computacional, que generalizam os métodos clássicos de síntese. Neste trabalho é apresentada uma interface, denominada AEFuzzy, desenvolvida para a síntese de termos ou ganhos de controladores *Fuzzy*, utilizando algoritmos evolutivos (Genético e Evolução Diferencial). A estratégia é baseada no conhecimento sobre as estruturas clássicas de PID. Os resultados foram obtidos a partir de três plantas com diferentes características: uma de quarta ordem, uma de segunda ordem com atraso e outra de terceira ordem de fase não mínima. Dos testes foi possível confirmar que a ferramenta AEFuzzy consegue sintonizar um PID *Fuzzy* para qualquer sistema compensado. Acreditamos que apesar de o estudo ter sido realizado unicamente em simulação, a complexidade dos estudos de caso justifica a sua aplicabilidade prática em situações recorrentes em plantas reais.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Abordagem clássica

Os sistemas de processos, industriais ou não, na maioria das vezes necessitam se adequar a certas características parametrizadas, como: baixa oscilação, rapidez em resposta, estabilidade relativa e absoluta, boa regulação, precisão e rejeição adequada a perturbações, erros de modelagem e ruídos diversos. Para poder obter esse resultado a partir de uma planta fora destas condições, o sistema precisa de uma realimentação da saída para quantificar sua diferença em relação à referência de entrada, e de uma estrutura que consiga controlá-lo: os compensadores. Devido a isso, e pelo fato de a sintonia desses controladores não ser trivial, muitas teses e artigos focaram no desenvolvimento de métodos de síntese e modelagem práticos e com menos complexidade, como visto em Féo (2004) e em Bertolani (2013).

Os controladores proporcional-integral-derivativos (incluindo P, PD, PI e PID) e os compensadores em atraso e avanço compõem o conjunto dos controladores clássicos, como pode-se verificar em Kuo e Golnaraghi (2009). Essas estruturas de controle já estão consolidadas na literatura e tem uma gama enorme de aplicações práticas. Pela diferença em suas constituições, cada compensador causa um efeito diferente no sistema. Contudo, os compensadores PID, por sua simplicidade, flexibilidade e desempenho aceitável em uma grande variedade de situações, são os mais populares dentro do meio industrial.

Os controladores PIDs, são compostos por três termos, como o próprio nome sugere. Esta estrutura adiciona dois zero e um polo na origem (integrador) em malha aberta. Os ganhos desse compensador afetam a localização dos polos de malha fechada, cujo número aumenta com a introdução do integrador, além de controlar a planta procurando obter os atributos necessários para respeitar e se enquadrar às especificações definidas pelo projetista. Sua função de transferência pode ser conferida na Equação (1):

$$G_{PID}(s) = Kp + Kd.s + \frac{Ki}{s} \tag{1}$$

Cada termo influencia na dinâmica do sistema de forma diferente. O aumento do termo proporcional (P) pode aumentar a rapidez com a diminuição do tempo de subida, pode melhorar a regulação e a rejeição a perturbações, e aumentar a precisão diminuindo o erro

estacionário. Por outro lado, pode aumentar o sobressinal e o tempo de estabilização, aumentando a oscilação. O aumento da banda passante em frequência pode aumentar a sensibilidade a ruídos, e o aumento do ganho proporcional (*Kp*) também pode aumentar muito o esforço de controle. Deste modo, a sintonia clássica procura ajustar esses parâmetros para obter uma resposta dentro de especificações globais, no tempo e/ou na frequência, estabelecidas a priori. Neste processo, muitas vezes é necessário priorizar uma característica com uma importância maior no projeto em detrimento de outra (*trade-off*).

Muitas vezes em projetos de sintonia, o esforço de controle é deixado de lado pelo projetista, que foca em rapidez e precisão, sendo obrigado a utilizar soluções de alto ganho. Na simulação, em termos ideais, a solução parece satisfatória, mas os ganhos elevados podem gerar uma resposta não desejada, devido às possíveis saturações das variáveis de entrada do sistema. Neste trabalho, uma ênfase especial é dada a análise do esforço de controle, extremamente relevante e negligenciado na literatura.

A introdução do termo integral (I) adiciona um polo na origem em malha aberta, com o objetivo de melhorar a precisão na intenção de acompanhar referências de maior ordem. Desta forma, o tipo do sistema aumenta, melhorando o erro estacionário. Em geral, isso diminui a estabilidade relativa, o que pode requerer correções adicionais para aumentar o amortecimento, diminuindo o sobressinal. Este termo, pode trazer sérios problemas em sua sintonia, pois, com seu ganho (*Ki*) ajustado para o erro especificado, o sistema pode ficar tão lento, que a melhoria em precisão não justificaria o tempo gasto para a realização da tarefa, incompatível com o exigido. Além disso, a estabilidade relativa pode piorar tanto que o sistema é obrigado a oscilar bastante para zerar a integral de erros iniciais muito grandes.

O termo derivativo (D) possibilita um ganho simultâneo de precisão e estabilidade relativa. Esse parâmetro adiciona um zero no sistema de malha aberta, podendo aumentar bastante a banda passante, amplificando ruídos indesejados. Neste caso, uma solução mais adequada poderia ser a utilização de um compensador em avanço, constituído por um polo e um zero dominante em baixa frequência.

Os aspectos supra citados mostram o cuidado que se deve ter no projeto de sintonia de controladores clássicos. Muitas vezes a literatura propõe modificações heurísticas para

ampliar o uso desses compensadores em sistemas não lineares, ou com parâmetros variáveis, através da adaptação de ganhos. Entretanto, em casos onde o sistema apresenta fatores complicadores, como incertezas e não linearidades relevantes, uma das alternativas mais promissoras é o controle inteligente.

1.2 Motivação

Quando o sistema apresenta fatores complicadores, como incertezas e não linearidades relevantes, uma das alternativas mais promissoras ao controle clássico é o controle inteligente. O controle inteligente baseia-se no uso das três técnicas básicas da inteligência computacional cognitiva: A lógica multivalente *Fuzzy*, as Redes Neurais e os algoritmos evolutivos. Os sistemas *Fuzzy* modelam a maneira do ser humano de lidar com informações imprecisas, as Redes Neurais modelam aproximadamente a arquitetura do cérebro humano, como uma rede de neurônios (*perceptrons*) e sua habilidade de aprender e de ser treinada, e os algoritmos evolutivos são usados como algoritmos de otimização estocásticos, baseando-se na teoria da evolução natural, auxiliares para sintonia de ganhos, parâmetros diversos e outras tarefas auxiliares na síntese de controle. No capítulo 2 serão explicitados dois algoritmos evolutivos: o Algoritmo Genético (AG) e o Algoritmo de Evolução Diferencial (ED), implementados na ferramenta desenvolvida neste trabalho. (LIN; LEE, 1996).

Por ser um estimador numérico capaz de modelar sistemas não lineares complexos, com bom desempenho em ambientes com imprecisão, incerteza e ruído, o sistema *Fuzzy* vem sendo utilizado no controle de sistemas lineares e não lineares. A sua versatilidade, simplicidade e flexibilidade são também grandes atrativos, e umas das causas da disseminação do seu uso.

Nessas condições, propõe-se uma estratégia de ajuste de controladores PID-*Fuzzy* utilizando algoritmos evolutivos, tanto na síntese de termos, quanto na síntese de ganhos, buscando um sistema otimizado. Além disso, vê-se importante fazer uma análise crítica da bibliografia sobre a sintonia de controladores clássicos e suas modificações heurísticas para sistemas não lineares, e sobre propostas de algoritmos baseados em inteligência computacional, verificando atributos como sua generalidade, complexidade, desempenho, entre outros.

Desde a consolidação do conceito por Zadeh (ZADEH, 1965) com a publicação do seu artigo sobre conjuntos *Fuzzy*, muitas pesquisas foram realizadas sobre otimização e sintonia de controladores *Fuzzy* (PIs, PDs e PIDs) por algoritmos evolutivos. No estudo essas mesmas técnicas foram utilizadas para obter resultados que tivessem desempenho igual ou superior às estruturas clássicas usadas anteriormente. Os critérios de desempenho foram baseados na rapidez de resposta, nas características de oscilação, ou de erro estacionário. Os resultados, mostrados no capítulo 4, reforçam as técnicas *Fuzzy* como ferramentas adequadas de síntese de controladores mais eficientes.

1.3 Análise da literatura especializada

Em Carli, Liguori e Marroni (1994) é proposta uma estratégia de controlador PI, onde os parâmetros de controle, *Kp* e *Ki*, são alterados dependendo do valor do erro estacionário e de sua derivada por um sistema *Fuzzy* embarcado. De forma parecida, em Sharkawy (2010) é apresentado um controlador PID utilizado para o controle de sistemas de freio automotivo ABS (do inglês Anti-lock Braking System). No entanto, diferentemente de Carli, Liguori e Marroni (1994), os parâmetros, *Kp*, *Kd* e *Ki*, são sintonizados em tempo de execução por um sistema *Fuzzy*-Genético. Ou seja, caso o algoritmo genético encontre, no decorrer de suas iterações, uma configuração do sistema *Fuzzy*, que ajuste os parâmetros do PID de forma a obter uma resposta mais próxima da desejada, ele alterará o sistema para a nova versão automaticamente.

Um controlador híbrido *Fuzzy*- Não Linear é proposto em Pandolfi (2012) para o controle dos movimentos em marcha ré de Robôs Móveis Multiarticulados (RMMAs). Essa estratégia utiliza o método de Wang & Mendel para a sintonia do controlador *Fuzzy*, além do método analítico para o ajuste da parte não linear. A importante diferença entre esta proposta e as citadas anteriormente, é que Pandolfi (2012) propõe um controlador *Fuzzy*, enquanto nos casos de Carli, Liguori e Marroni (1994) e Sharkawy (2010), a técnica *Fuzzy* é utilizada como uma estratégia de ajuste de ganhos de controladores clássicos em tempo de execução.

Abeywardena et al. (2009) se diferencia dos demais trabalhos mencionados por aplicar a Lógica *Fuzzy* como uma estratégia para estabilizar e controlar um Veículo Aéreo Não Tripulado (VANT). Seu controlador *Fuzzy* foi projetado e simulado com um modelo dinâmico

não linear do sistema de propulsão, e seus resultados demonstraram que condições estáveis de *hovering* (pairar) podem ser alcançadas com o controlador desenvolvido. Este trabalho demonstra a grande aplicabilidade dos sistemas *Fuzzy*, que podem em muitos casos servir de alternativa para controladores clássicos, principalmente em plantas não lineares complexas.

Além de contribuir com o desenvolvimento de sistemas *Fuzzy* para a área de controle e modelagem de plantas, a relevância desta pesquisa está em propor uma nova metodologia de ajuste de controladores *Fuzzy* utilizando algoritmos evolutivos. De uma forma geral, é comum na literatura encontrar trabalhos que utilizam a tecnologia *Fuzzy* para simular o conhecimento de um especialista, com o objetivo de ajustar parâmetros de controladores lineares. O algoritmo genético entra, comumente, no ajuste de termos *Fuzzy* ou até mesmo na sintonia de ganhos de um controlador puramente linear. O que se propõe neste trabalho é realizar a síntese de um controlador PID *Fuzzy* que apresente comportamento adequado para um conjunto de plantas diferentes, reforçando sua utilidade. Assim como o emprego de algoritmos evolutivos no processo de ajuste, tanto no aspecto de síntese de termos, quanto na síntese de ganhos.

1.4 Objetivo geral e objetivos específicos

O objetivo geral deste projeto é desenvolver uma aplicação de síntese (sintonia) de ganhos e/ou de modelagem (síntese de termos *Fuzzy*), utilizando técnicas de inteligência computacional.

Os objetivos específicos deste trabalho são:

- Realizar uma análise crítica sobre a literatura em modelagem e controle inteligente, mormente as propostas versando sobre estruturas do tipo PID;
- Analisar a bibliografia sobre a sintonia de controladores clássicos e as modificações heurísticas propostas para ampliar seu espectro de uso em sistemas não lineares, e sobre algoritmos propostos baseados em técnicas de inteligência computacional, para então montar uma estrutura de um controlador PID inteligente em simulação;
- Desenvolver um *software* baseado em algoritmos evolutivos para encontrar o melhor ajuste de termos ou de ganhos na sintonia de PIDs *Fuzzy*, procurando obter igual ou

melhor desempenho que um PID Clássico (menor sobressinal, maior velocidade de resposta, menor esforço de controle, etc);

- Simular e sintonizar múltiplos controladores PID *Fuzzy* utilizando o programa criado;
- Proceder a análise de desempenho e a validação das soluções propostas em três estudos de caso, em simulação, para três plantas distintas, onde o AG ou ED, utilizados, ajustarão um PID *Fuzzy* a partir de um controlador clássico.

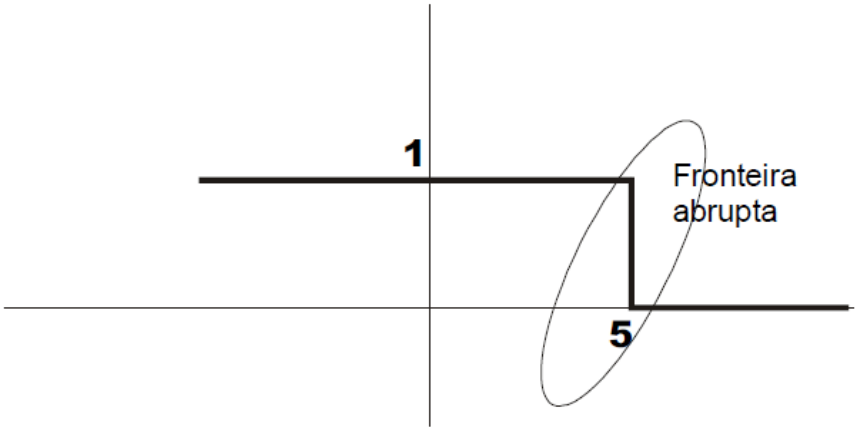
2 *FUZZY* E ALGORITMOS EVOLUTIVOS

Este trabalho apresenta uma aplicação de sintonia de ganhos e termos de controladores inteligentes utilizando técnicas de inteligência computacional. Assim sendo, esse capítulo traz um estudo sobre a teoria da Lógica *Fuzzy* nas síntese de controle, bem como dos algoritmos evolutivos, mostrando como foram empregados na construção da ferramenta proposta.

2.1 Lógica *Fuzzy* e seu uso em controladores inteligentes

A lógica bivalente clássica é um método de classificação, onde o item de estudo é identificado como pertencente ou não a um certo conjunto. Os conjuntos que pertencem à lógica clássica são conhecidos como conjuntos *crisp*. O parâmetro que representa o quanto um item pertence ou não a um determinado conjunto é denominado pertinência (representada pela letra grega μ), e seu valor ($\mu(x)$) pode variar de 0 a 1 (0 a 100%). O item que pertencer a um conjunto *crisp* possui pertinência 1, enquanto um item que não pertence, possui pertinência 0, por isso bivalente. Um exemplo disso pode ser visto na Figura 1 que segue:

Figura 1 – Conjunto *crisp* "menor ou igual a 5"



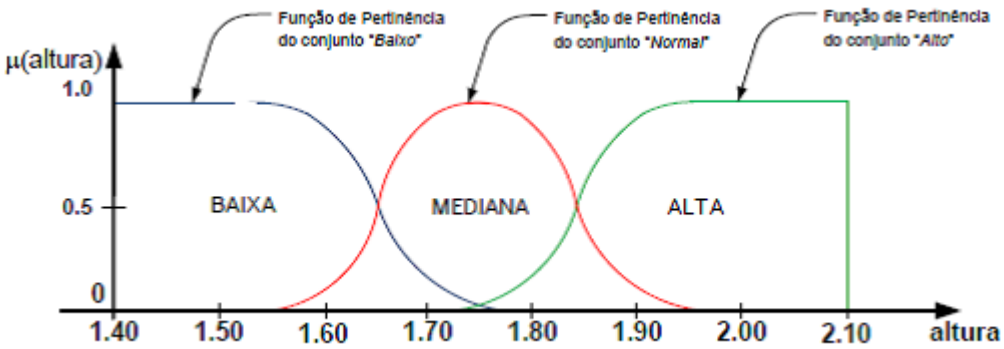
Fonte: FERREIRA, 2009.

A Figura 1 apresenta o conjunto "menor ou igual a 5", onde pode ser notada a transição abrupta do nível de pertinência 1 (pertence) para o de pertinência 0 (não pertence). Isso demonstra como essa abordagem é brusca e rígida, não sendo esta a melhor abordagem no que tange a controladores. Esta formulação de problema se restringe aos casos onde não se encontram subjetividades e incertezas, sem mecanismos analíticos de quantificação. (FERREIRA, 2009).

A necessidade de um método que consiga tratar a informação de uma forma mais suave ao passar de uma classificação vizinha para outra (encontro de conjuntos adjacentes), fica mais fácil de ser percebida quando se trabalha com problemas subjetivos. Por exemplo, caso um rapaz fosse convidado a ficar em uma praça movimentada no centro de uma cidade, e para cada pessoa que passasse, ele tivesse que classificar se a pessoa é *alta*, *mediana* ou *baixa*, provavelmente as pessoas com altura menor que 1,60 m seriam classificadas como baixas e pessoas com altura maior que 1,90 m seriam classificadas como altas.

Como o rapaz não sabe a altura verdadeira das pessoas, a percepção que ele tem dos classificados é subjetiva. As pessoas entre 1,70 m e 1,80 m poderiam ser consideradas medianas, apesar de deixar o rapaz um pouco em dúvida. Portanto, uma representação mais adequada a este caso, poderia ser a apresentada pela Figura 2, a seguir, que divide o universo da variável *altura* em conjuntos de fronteiras mais suaves:

Figura 2 – Possível representação de conjuntos para variável *altura*



Fonte: PANDOLFI, 2012.

Esse conceito é explorado na Lógica *Fuzzy*, que por sua vez, é um método de classificação matemático para tratar situações imprecisas. A representação formal da Lógica *Fuzzy* para as funções de pertinência e conjuntos no universo *Fuzzy*, de acordo com Ferreira (2009, p. 13), é:

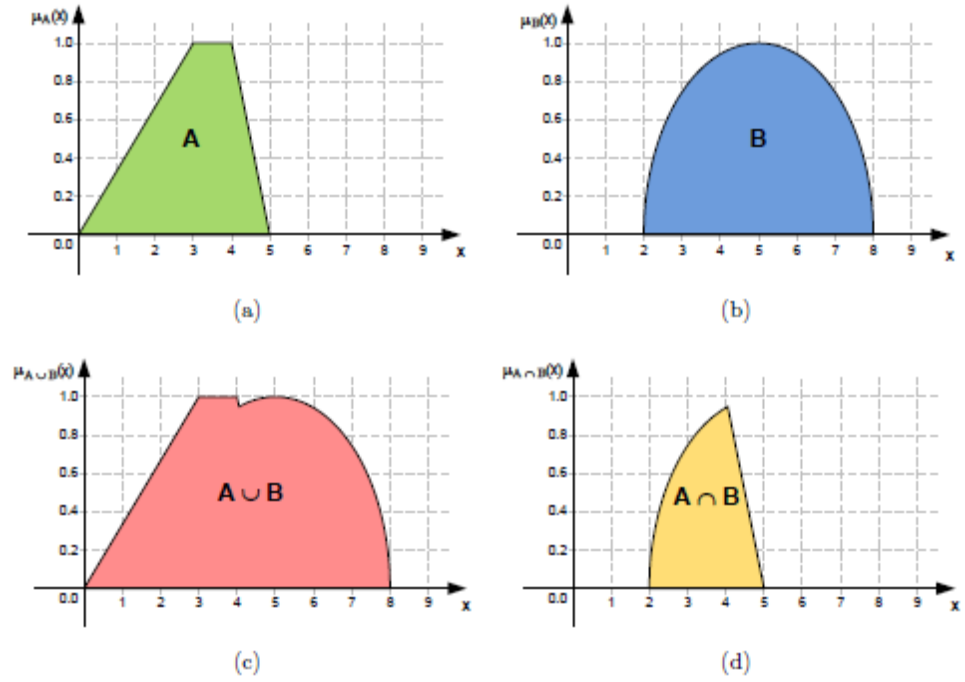
> Considere o conjunto *U*, onde os elementos de interesse estão definidos. *U* é denominado universo de discurso. Um conjunto *Fuzzy A* no universo de discurso *U*, pode ser definido como o conjunto dos pares ordenados:
> $A = \{(x, \mu_A(x)) / x \in U\}$

Isto é, um conjunto *Fuzzy A* qualquer é composto pelos pares ordenados dos pontos (*x*) pertencentes ao universo *U* e suas respectivas pertinências ($\mu_A(x)$).

Assim como na matemática usual, a Lógica *Fuzzy* também possui operações. Essas operações são realizadas sobre conjuntos, gerando no fim um novo conjunto *Fuzzy*. Essas idéias só foram formalizadas de fato por Zadeh em 1965, que apresentou em seu artigo *Fuzzy sets* (conjuntos *Fuzzy*) a teoria dos conjuntos linguísticos *Fuzzy*, e introduziu o termo "Lógica *Fuzzy*". (FERREIRA, 2009).

As operações mais básicas são a união e a interseção, e suas determinações podem consistir na utilização de dois operadores: *máx* e *min*, respectivamente (esses operadores também podem ser representados por: *máx* = $\vee$ e *min* = $\wedge$). A interseção *Fuzzy* de dois conjuntos, *A* e *B*, gera um novo conjunto *C*, obtido através do mínimo valor de pertinência entre os dois conjuntos para cada valor de *x*: $C = min(A,B)$, ou seja: $\mu_C(x) = min(\mu_A(x), \mu_B(x))$. Do mesmo modo, a união *Fuzzy* de dois conjuntos, *A* e *B*, gera um novo conjunto *C*, obtido através do máximo valor de pertinência entre os dois conjuntos para cada valor de *x*: $C = máx(A,B)$, ou seja: $\mu_C(x) = máx(\mu_A(x), \mu_B(x))$. A Figura 3 ilustra bem esse conceito:

Figura 3 – Ilustração das operações de união e interseção entre os conjuntos *A* e *B*

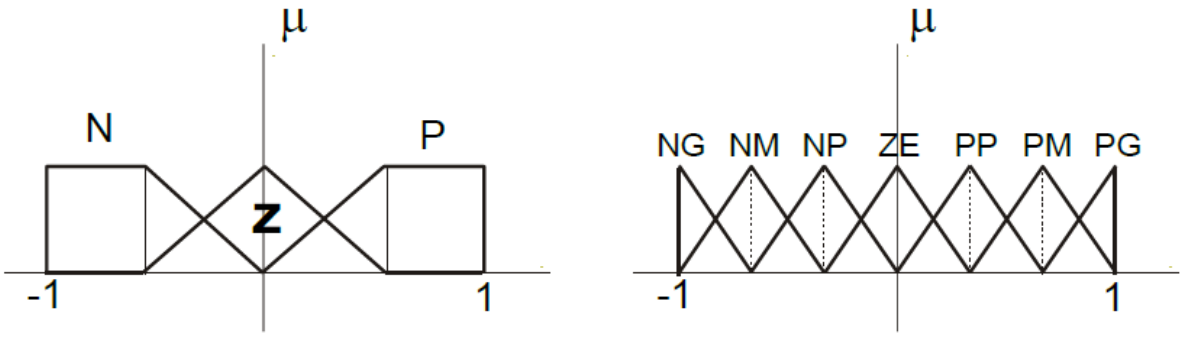


Fonte: PANDOLFI, 2012.
Legenda: (a) Conjunto *A*
(b) Conjunto *B*
(c) União entre conjuntos *A* e *B*
(d) Interseção entre conjuntos *A* e *B*

A variável linguística, na Lógica *Fuzzy*, é utilizada para descrever um conhecimento, que pode assumir valores num conjunto de termos e sentenças em linguagem natural ou artificial. Por exemplo, como citada anteriormente, a variável linguística *altura* pode assumir os valores: alta, mediana, baixa, muito alta, muito baixa, etc. Esta formulação é importante para o projeto proposto neste trabalho, visto que os sistemas *Fuzzy* utilizados para a síntese de um controlador PID *Fuzzy* são estruturados em sua base por toda esta teoria discutida.

Uma variável linguística de um controlador PID *Fuzzy* pode ser, por exemplo, a componente proporcional obtida a partir do erro da saída em relação ao ponto de operação. Essa variável pode ser descrita por quantos conjuntos *Fuzzy* forem necessários para representá-la. A divisão do universo de discurso em termos diferentes é denominada particionamento *Fuzzy*. A quantidade de conjuntos escolhida é chamada de "granularidade". Quanto maior a granularidade, mais termos existirão e mais precisa será a representação do sistema. A Figura 4 mostra o particionamento de 3 e 7 termos para a variável linguística "Erro":

Figura 4 – Particionamento *Fuzzy* da variável linguística "Erro"



Fonte: FERREIRA, 2009.
Legenda: (a) Particionamento com granularidade 3 (conjuntos N, Z e P)
(b) Particionamento com granularidade 7 (conjuntos NG, NM, NP, Z, PP, PM e PG)
Nota: (a) Z representa zero, P representa positivo, e N representa negativo
(b) ZE, PP, PM, PG, NP, NM e NG representam, respectivamente: zero, positivo pequeno, médio e grande, negativo pequeno, médio e grande

Os sistemas *Fuzzy* permitem a representação e quantificação da ambiguidade descrevendo o sistema, estabelecendo relações entre conjuntos *Fuzzy* de entrada e saída. Essas relações tratam tanto dados numéricos quanto subjetivos, podendo ser expressas na forma de regras Se-Então. Esse modelo de sistema baseado em regras é conhecido também como Sistema de Inferência *Fuzzy* (*Fuzzy Inference Systems: FIS*), Modelo Nebuloso, e Sistema Especialista *Fuzzy*.

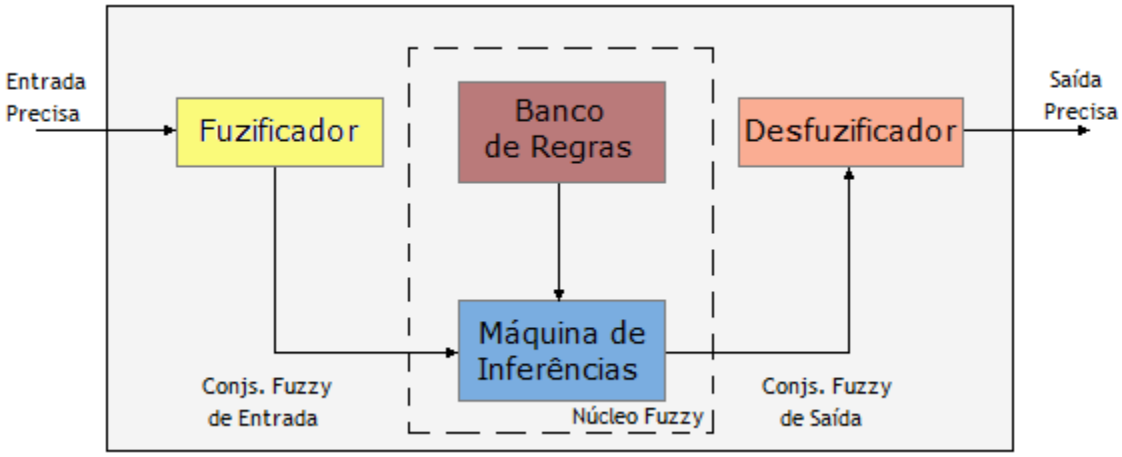
Em Mamdani (1977), um exemplo de aplicação dessas regras pode ser verificado, onde as entradas são: o erro de pressão (representado por PE) e a variação do erro de pressão (representada por CPE), e a saída especifica a mudança a ser feita na quantidade de calor a ser aplicada no processo (HC): "SE PE é Negativo Pequeno E CPE é Positivo Pequeno ENTÃO HC é Positivo Médio." (MAMDANI, 1977, p.1184, tradução nossa).

Na literatura, os dois tipos de Sistemas de Inferência *Fuzzy* mais importantes são: o Modelo Linear de Takagi-Sugeno (TAKAGI; SUGENO, 1985) e o Modelo Relacional ou Linguístico de Mamdani (MAMDANI, 1977). A principal diferença entre as duas estratégias é a forma de representar a consequência de uma regra, ou seja, a resposta do sistema. Essa escolha de cada método na maneira de realização da inferência resultou na necessidade de um número menor de regras para o modelo de Takagi-Sugeno. (SIVANANDAM, 2007).

O modelo de Mamdani apresenta o resultado de uma regra na forma de termos linguísticos, o que permite introduzir o conhecimento de especialistas na base de regras do sistema. Já o modelo de Takagi-Sugeno usa uma função dos valores numéricos das entradas como a resposta do modelo. Isso não permite a adição do conhecimento de especialistas e ainda necessita que os coeficientes da função sejam estimados. Além disso, seus procedimentos de identificação são complexos, o que justifica a popularidade do modelo de Mamdani. (SIVANANDAM, 2007).

A arquitetura básica de um Sistema de Inferência *Fuzzy* é constituída por um Fuzificador, um Banco de Regras, uma Máquina de Inferências, e um Desfuzificador, como ilustra a Figura 5:

Figura 5 – Arquitetura básica de um sistema *Fuzzy*



Fonte: Produção do próprio autor.

O Fuzificador mapeia os valores obtidos nas entradas, que geralmente são dados numéricos reais, em valores linguísticos no universo de discurso relativo a cada entrada, normalmente representados na forma de *singletons*. O *singleton* é um conjunto *Fuzzy* o qual tem pertinência $\mu_S(x) = 1$ no ponto em que $x = x_o$ e 0 para todos os outros pontos. Para o caso da fuzificação, x_o é valor mapeado da entrada.

O Banco de Regras, como o nome sugere, armazena todas as regras Se-Então do sistema. Cada regra explicita uma relação causal entre entradas e saídas do sistema em questão, obtida através de algum método e/ou da experiência de especialistas. Um método consagrado para obtenção de regras a partir de dados é o "Método de Wang & Mendel" (WANG; MENDEL, 1992), que apesar de seu prestígio, foi demonstrado ser um método bastante ineficiente no

levantamento de regras, como pode ser visto em Pandolfi (2012) para o caso de Robôs Móveis Multiarticulados.

Os conjuntos *singletons* chegam do Fuzificador na Máquina de inferências, que verifica a informação recém chegada relacionando-a com as regras descritas no Banco de Regras do modelo. A resposta é obtida através de um processo que simula o modo humano de decisão, utilizando uma combinação das regras que envolvem esses *singletons* por meio de uma operação de "composição". Essa operação é uma junção de operadores, tendo várias possibilidades. Os tipos mais utilizados, pela sua simplicidade, são: *max-min* e *max-produto*.

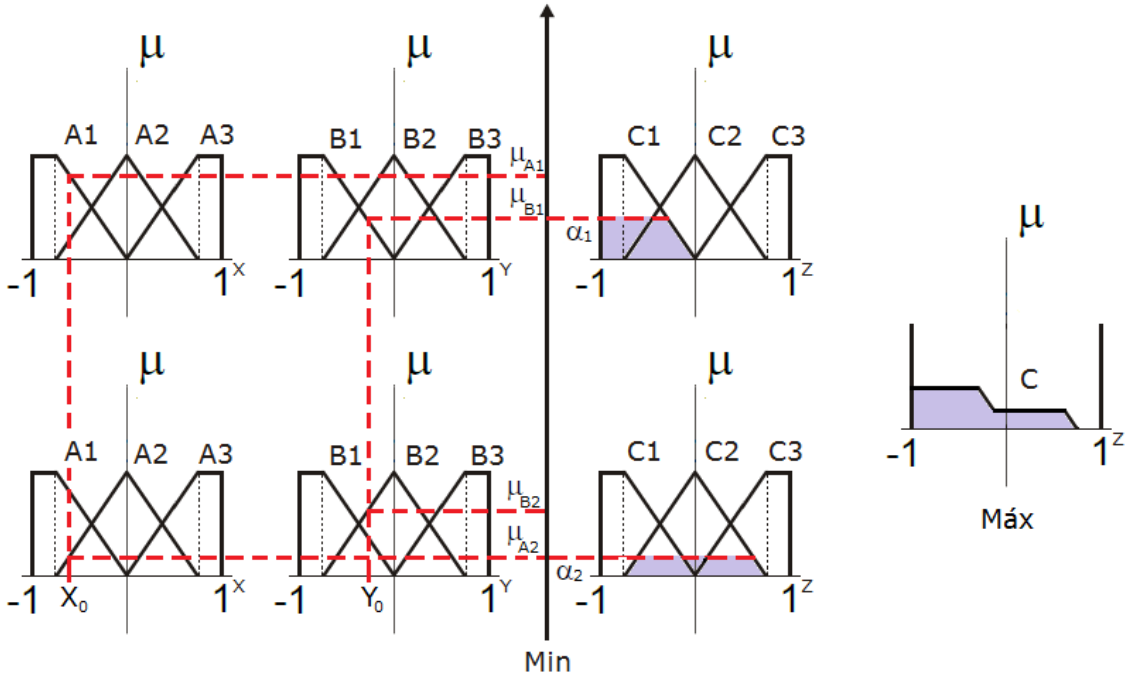
A representação matemática das operações de composição *max-min* e *max-produto* podem ser conferidas nas Equações (2) e (3), respectivamente:

$$\mu_C(z) = \left(\alpha_1 \wedge \mu_{C1}(z)\right) \vee \left(\alpha_2 \wedge \mu_{C2}(z)\right) \quad (2)$$

$$\mu_C(z) = \left(\alpha_1 . \mu_{C1}(z)\right) \vee \left(\alpha_2 . \mu_{C2}(z)\right) \quad (3)$$

Para simplificar, têm-se duas regras, que geram *C1* e *C2*. Na regra 1, α_1 é obtido através do mínimo entre a pertinência da primeira entrada no conjunto *A1*, e a pertinência da segunda entrada no conjunto *B1* ($\alpha_1 = \mu_{A1}(x) \wedge \mu_{B1}(y)$, pois $\alpha_1 = \mu_{A1}(x)$ e $\mu_{B1}(y) = \mu_{A1}(x)$ *interseção* $\mu_{B1}(y)$). Intuitivamente, a primeira regra diz que: SE *A1* e *B1* ENTÃO *C1*, logo, o conjunto resultante é *C1* e sua função de pertinência é representada por $\mu_{C1}(z)$. De mesmo modo é feita a segunda regra. Por fim, o resultado dessa inferência é o conjunto *C*, que consiste na operação *máx* entre *C1* e *C2*, ou seja, *C* é *C1 ou C2* ($C = C1$ *união* $C2 = C1 \vee C2$). A Figura 6 ilustra esse exemplo do método de inferência utilizando *max-min*:

Figura 6 – Método de Inferência *Fuzzy* utilizando *max-min*



Fonte: Produção do próprio autor.

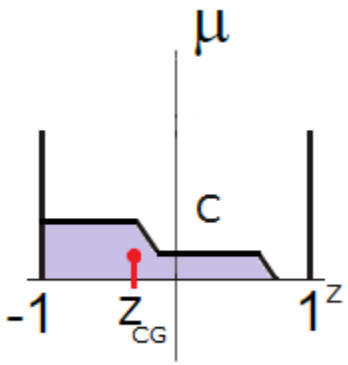
A Figura 6 reforça a ideia do Modelo *Fuzzy*, que apesar de uma informação parecer obedecer mais a uma certa regra, ela não deixa de ser influenciada no final pelas outras "menos adequadas", sendo no final uma composição de várias regras. Se o sistema tivesse mais do que duas entradas, seria necessário apenas adicionar mais uma variável nas regras, por exemplo: SE *A* e *B* e *C* ENTÃO *D*.

Finalmente, o Desfuzificador toma esse resultado encontrado pela Máquina de Inferências e utiliza um método para transformar essa informação composta por termos linguísticos em uma resposta numérica, representada na Figura 5 pela "Saída Precisa". Esse mapeamento *Fuzzy*-escalar pode ser definido como a produção de uma ação de controle *crisp* que melhor representa a distribuição de possibilidade de uma ação de controle *Fuzzy* inferida. (FERREIRA, 2009).

Dentre os métodos de desfuzificação, um dos mais comuns na literatura é o método do centro de gravidade (CG). Esse método busca o valor no universo de discurso que divide a área sob a curva da função de pertinência em duas partes iguais. Para demonstrar essa estratégia de

desfuzificação, pode-se aproveitar o exemplo da Figura 6, e mostrar na Figura 7 o resultado desse método:

Figura 7 – Desfuzificação pelo método do centro de gravidade



Fonte: Produção do próprio autor.

As Equações (4) e (5) descrevem esse método nos domínios contínuo e discreto, respectivamente:

$$z_{CG} = \frac{\int_{z \in U} z.\mu_C(z).dz}{\int_{z \in U} \mu_C(z).dz} \tag{4}$$

$$z_{CG} = \frac{\sum_{z \in U} z.\mu_C(z)}{\sum_{z \in U} \mu_C(z)} \tag{5}$$

Com a estrutura de um sistema *Fuzzy* completa, sua capacidade de modelar sistemas não lineares complexos, em ambientes com imprecisão, incertezas e ruídos, também possibilitando ao projetista adicionar conhecimento especialista, além da modelagem pelo levantamento de dados numéricos da planta, os sistemas *Fuzzy* aparecem como uma boa opção no controle de sistemas lineares e não lineares, ou seja, uma interessante alternativa aos controladores clássicos.

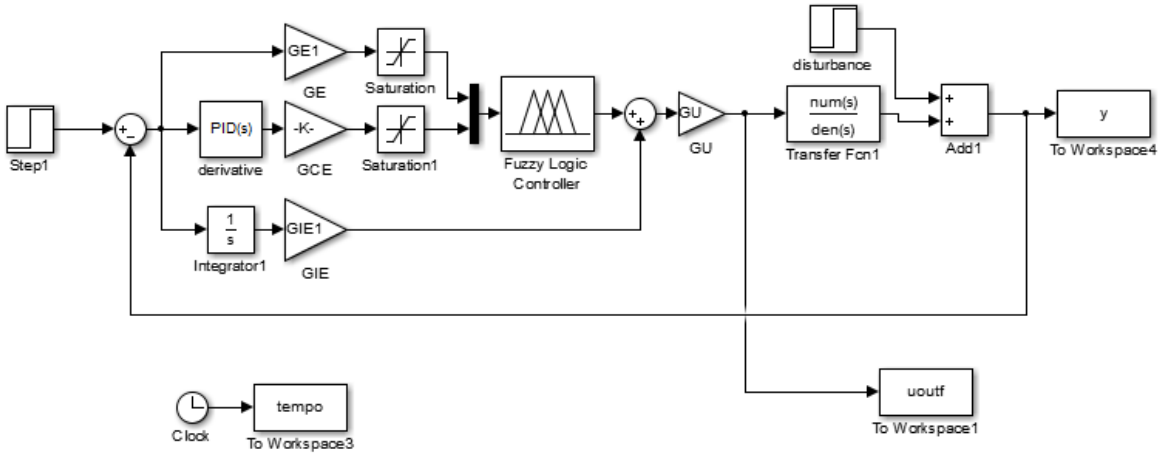
É possível perceber o número de aplicações e propostas citadas neste trabalho, no entanto, o trabalho realizado por Jan Jantzen em Jantzen (1998a) e Jantzen (1998b), que propõe uma sistemática de síntese de controladores *Fuzzy* baseada nos conhecimentos acerca da sintonia de controladores PID clássicos, merece uma atenção especial. A proposta de Jantzen consegue

substituir controladores lineares através de uma equivalência de ganhos (de *Kp*, *Kd* e *Ki*, para *GE*, *GCE*, *GIE* e *GU*), e da inserção de um conjunto de regras num Sistema de Inferência *Fuzzy* de uma forma bem especificada pelo próprio autor.

Essa estratégia motivou a escolha de seu modelo como ponto de partida para o desenvolvimento dessa pesquisa. Dessa forma, como a qualidade de resposta de qualquer sistema, tanto para PIDs clássicos quanto para inteligentes, depende basicamente da sua sintonia, a ideia deste projeto foi, portanto, implementar um sistema baseado em algoritmos evolutivos para realizar a síntese de ganhos e termos de controladores PID *Fuzzy*, a fim de preencher a lacuna deixada por Jan Jantzen na escolha de métodos para realizá-la.

A estrutura do controlador inteligente apresentada por Jantzen, e utilizada neste trabalho de graduação, foi reproduzida no Simulink (MATLAB®) e pode ser conferida na Figura 8. Nesta figura é possível perceber que o PID *Fuzzy* é composto por um PD *Fuzzy* (FPD) mais uma parte integral somada na saída do controlador *Fuzzy*.

Figura 8 – Estrutura de controlador *Fuzzy* FPD+I proposta por J. Jantzen



Fonte: Produção do próprio autor.

Na Figura 8, percebe-se a presença de quatro ganhos: três que escalonam o erro, a derivada do erro e a integral do erro (*GE*, *GCE* e *GIE*, respectivamente) para o universo *Fuzzy*, e um, *GU*, que compatibiliza em termos de escala a saída do controlador com a entrada da planta, não sendo necessário um saturador após este último ganho. Esses quatro ganhos se relacionam com os valores de *Kp*, *Kd* (*Kd = Kp*Td*) e *Ki* (*Ki = Kp/Ti*) como mostra na Figura 9:

Figura 9 – Relação entre os ganhos do PID *Fuzzy* e do clássico

Controller	K_p	$1/T_i$	T_d
FP	$GE * GU$		
FInc	$GCE * GCU$	GE/GCE	
FPD	$GE * GU$		GCE/GE
FPD + I	$GE * GU$	GIE/GE	GCE/GE

Fonte: JANTZEN, 1998b.

Nota-se na Figura 10, que mesmo o controlador proposto possuindo um núcleo *Fuzzy*, sua estrutura *Fuzzy* funciona como um somador simples, ou seja, representa somente a soma do erro com sua derivada compensados pelos seus respectivos ganhos (Erro + dErro = Saida, NP + PP = ZE).

Figura 10 – Base de regras de um FPD+I com 49 regras

Erro \ Derivada do Erro	NG	NM	NP	ZE	PP	PM	PG
NG	NG	NG	NG	NG	NM	NP	ZE
NM	NG	NG	NG	NM	NP	ZE	PP
NP	NG	NG	NM	NP	ZE	PP	PM
ZE	NG	NM	NP	ZE	PP	PM	PG
PP	NM	NP	ZE	PP	PM	PG	PG
PM	NP	ZE	PP	PM	PG	PG	PG
PG	ZE	PP	PM	PG	PG	PG	PG

Fonte: Produção do próprio autor.

É certo que, apesar da simplicidade na formação de seu núcleo *Fuzzy*, esse controlador inteligente é uma estrutura extremamente interessante, visto que este possibilita a configuração do controlador de maneira intuitiva sem que sejam necessários mais cálculos, já que a forma como as regras são escolhidas depende do projetista e do próprio projeto. Além disso, a ferramenta de síntese proposta nesta pesquisa poderá realizar o ajuste fino de termos e ganhos sempre que preciso sem a necessidade de maiores esforços do pesquisador.

2.2 Algoritmos Evolutivos

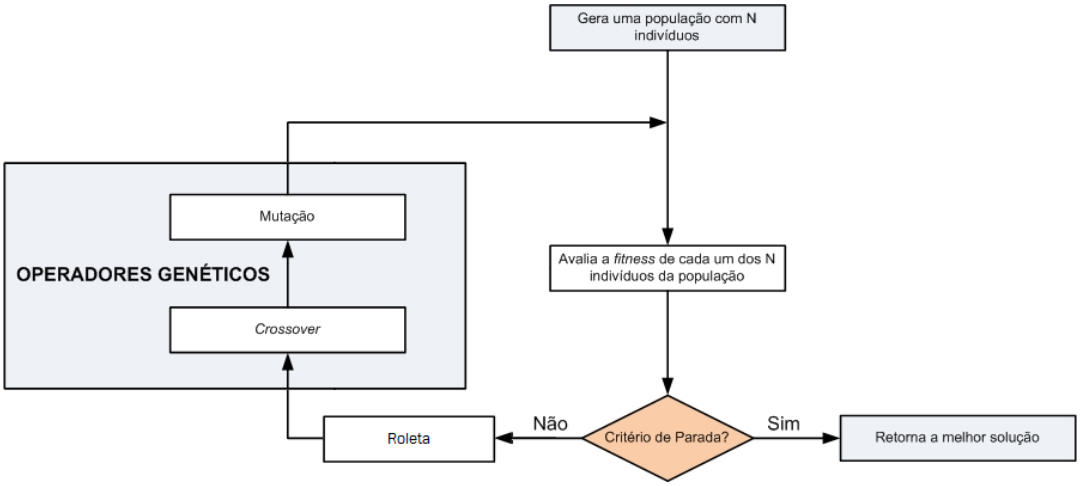
Os algoritmos evolutivos são inspirados nos mecanismos adaptativos utilizados pela natureza para a superação de problemas. A criação desse tipo de estratégia computacional motivou o desenvolvimento de muitas outras variações, como: o algoritmo genético, o algoritmo de evolução diferencial, e a nuvem de partículas, que imita a movimentação dos pássaros. O uso destas técnicas justifica-se bem pelo fato de que a otimização de um sistema através dos algoritmos evolutivos não exige muito conhecimento da planta, sendo ainda assim possível encontrar uma ótima solução.

2.2.1 Algoritmo Genético e sua aplicação na síntese de controladores

O algoritmo genético baseia-se no processo de evolução natural das espécies. Este foi desenvolvido por John Holland (HOLLAND, 1975) na Universidade de Michigan, e seu aluno David Goldberg (GOLDBERG, 1989) popularizou o método, apresentando sua utilização para resolver problemas complexos de engenharia. Hoje, essa técnica é um dos métodos mais utilizados para a otimização de sistemas, dando ênfase ao que se propõe com esta pesquisa, na síntese de controladores clássicos e avançados. (FERNANDES, 2006).

Assim como as espécies, o AG utiliza técnicas inspiradas na biologia evolutiva, como por exemplo: reprodução, mutação, recombinação (*crossover*) e exclusão, chamadas de operadores genéticos. Seu processo busca o melhor indivíduo de uma população, como no princípio *darwiniano*, o mais adaptado para sobreviver. As etapas do funcionamento básico do AG podem ser conferidas no fluxograma da Figura 8:

Figura 11 – Fluxograma de funcionamento básico do algoritmo genético



Fonte: BERTONALI, 2013 (modificado pelo autor).

No início do algoritmo, a primeira população de *N* indivíduos é criada aleatoriamente, com o objetivo de aumentar sua diversidade genética, garantindo um maior alcance do espaço de busca. O tamanho *N* dessa população possui um compromisso com o processamento e com a convergência do algoritmo. Uma população grande exige um alto esforço computacional, entretanto, uma população pequena tem mais chances de convergir e encontrar uma solução que não é a melhor para o problema estudado. (BERTOLANI, 2013).

Em seguida, cada indivíduo é avaliado em relação a sua proximidade do ideal. A função responsável por fazer essa avaliação é chamada de função *fitness* (aptidão), e após esse processo de análise, atribui-se um valor de *fitness* a cada indivíduo. Essa função em uma aplicação para síntese de controladores pode ser, por exemplo, uma resposta com sobressinal de 5%, tempo de subida de meio segundo e com erro estacionário de 10%. Quanto mais próximo desta resposta ideal estiver um determinado indivíduo, maior será sua *fitness*. Portanto, mais adaptado e melhor será o indivíduo.

O algoritmo, posteriormente, confere se o seu critério de parada foi obedecido, caso o resultado da verificação seja negativo, a próxima etapa é a criação de uma nova população para a próxima geração. Os critérios de parada podem ser o momento em que a geração corrente chegar ao seu valor máximo, um sinal externo (em programas com interface, o sinal poderia ser gerado por um botão) que interrompa o processo do algoritmo, o instante em que um indivíduo obtenha uma resposta aceitável comparada à função *fitness*, etc.

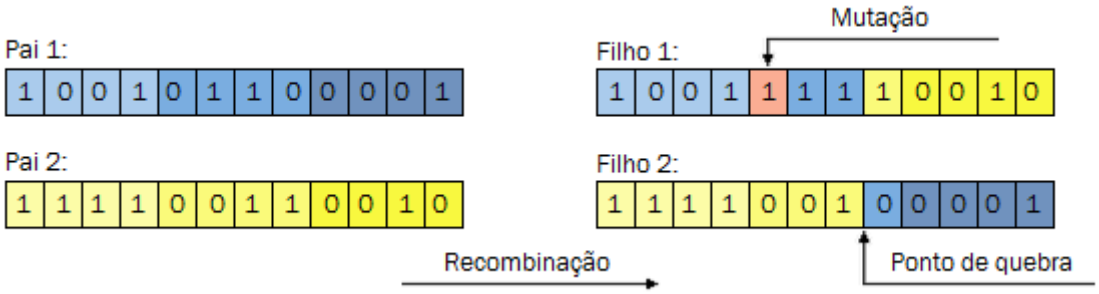
Caso o processo não tenha chegado ao fim na etapa do critério de parada, o próximo passo é a seleção dos indivíduos que poderão formar a nova geração. Um dos mais utilizados métodos de seleção é a clássica técnica da roleta. Neste método, cada indivíduo ocupa, em uma "roleta", uma área proporcional a sua aptidão (*fitness*). Desta forma, quanto maior a sua aptidão, mais chances de ser selecionado terá o indivíduo. Depois, uma simulação do giro da roleta é realizada e *N* indivíduos são selecionados. (FERNANDES, 2006).

A partir disso, os indivíduos realizam a recombinação e mutação aos pares. Estes são chamados de indivíduos pais, e após a implicação desses operadores genéticos, geram dois novos indivíduos que farão parte da próxima geração.

A recombinação, ou *crossover*, realiza a troca de informações genéticas dos dois pais para a criação de seus filhos, na quebra de seus cromossomos em um ou mais pontos aleatórios. Os cromossomos são a combinação de toda informação de um indivíduo que se deseja realizar a busca no algoritmo. Por exemplo, essa informação poderia ser a representação binária do conjunto dos ganhos proporcional, derivativo e integral de um controlador (*indivíduo = [Kp Kd Ki]*) que se busca a sintonia. Não há comprovação de que alguma maneira de utilização do operador de cruzamento apresente um desempenho superior às demais, entretanto, é possível se verificar que cada maneira é particularmente eficiente para uma determinada classe de problemas. (FERNANDES, 2006).

A etapa da mutação tem como objetivo manter e proporcionar a diversidade genética da população, alterando aleatoriamente um ou mais genes do cromossomo de um indivíduo. A probabilidade de ocorrer mutação deve ser pequena nesse algoritmo, normalmente de 1% a 2%, para que seja possível convergir e mesmo assim haver chances de procurar melhores soluções em outros locais do espaço de busca. A Figura 12, a seguir, ilustra o funcionamento dos operadores de recombinação e de mutação em dois indivíduos pais, gerando seus filhos. (FERNANDES, 2006).

Figura 12 – Processo de reprodução do algoritmo genético



Fonte: Produção do próprio autor.

Após esse processo de reprodução gerar os *N* filhos que compõem a população da nova geração, a aptidão (*fitness*) de cada novo indivíduo é encontrada, e caso o critério de parada não seja alcançado, esse ciclo continuará. Quando o critério for obedecido, o algoritmo, finalmente, retornará a melhor solução encontrada.

Atualmente, é comum, encontrar aplicações e trabalhos que utilizam essa estratégia genética na otimização de processos e sintonia de controladores. O algoritmo genético normalmente entra no ajuste de termos *Fuzzy* ou na sintonia de ganhos de um controlador puramente linear. Este método pode, então, realizar a síntese de um controlador PID *Fuzzy*, tanto no aspecto do ajuste de termos, quanto na síntese de ganhos, como realizado nessa pesquisa. Exemplos na literatura podem ser vistos em Bertolani (2013) e em Sharkawy (2010).

2.2.2 Evolução Diferencial e seu uso na síntese de controladores

O algoritmo ED, tal como o genético, se baseia no processo evolutivo natural das espécies, e foi desenvolvido em 1995 por Rainer Storn e Kenneth Price, quando tentavam resolver o problema polinomial de Chebyshev (STORN; PRICE, 1995). Apesar de apresentarem conceitos simples, o algoritmo pode ser implementado facilmente, além de ser robusto e eficiente para a minimização de funções não-lineares e não-diferenciáveis no espaço contínuo. (COELHO; MARIANI, 2006).

Assim como no algoritmo genético, o de evolução diferencial também utiliza técnicas inspiradas na biologia evolutiva. Os operadores genéticos que compõem esse algoritmo são:

mutação, recombinação (*crossover*) e seleção. Seu processo segue o mesmo princípio que o do algoritmo genético.

Inicialmente, a primeira população de *N* indivíduos é criada aleatoriamente dentro de seus limites previamente especificados. O tamanho *N* dessa população possui, sim, um compromisso com o processamento, porém, quanto maior a população, mais o universo poderá ser varrido, ou seja, mais chances o algoritmo terá de encontrar a melhor solução para o problema (ótimo global) e não encontrar um ótimo local.

Em seguida, cada indivíduo é avaliado em relação a função *fitness* e é levantada, portanto, sua *fitness*. Depois, é conferido se as condições de parada foram respeitadas, por exemplo, o indivíduo de melhor *fitness* ter um sobressinal menor que um valor pré-determinado, ou como no caso do AEFuzzy, se a geração chegou em seu valor máximo. Quanto maior a *fitness*, melhor é o indivíduo.

Caso os critérios de parada não tenham sido obedecidos, a população sofrerá alterações realizadas pelos operadores de mutação, *crossover* e seleção. Cada indivíduo ($x_i{}^g, i = 1, 2, 3 \dots N$) da população atual ($g$), representado por um vetor de tamanho M dentro do espaço de busca, como pode ser visto na Equação (6), passa por essa etapa para gerar seu sucessor na nova população ($x_i{}^{g+1}$). Os componentes (a_1, a_2, a_3, até a_M) podem representar ganhos, limites de universo *Fuzzy* (por exemplo, $\text{Erro}_{\text{mín}}$ e $\text{Erro}_{\text{máx}}$) e até mesmo termos dos controladores *Fuzzy*.

$$x_i{}^g = [a_1\ a_2\ a_3 \dots a_M] \tag{6}$$

O primeiro operador a afetar os indivíduos é a mutação. Diferentemente do algoritmo genético, a mutação no algoritmo de evolução diferencial escolhe aleatoriamente três indivíduos diferentes (inclusive diferentes do indivíduo corrente, x_i): o vetor *base*, o r_2 e o r_3, como pode-se conferir nas Equações (7), (8) e (9). Os vetores r_2 e r_3 são responsáveis por gerar o vetor *diferenças* (r_d), como pode-se ver na Equação (10):

$$base = x_{j1}{}^g, j1 \neq i \tag{7}$$

$$r_2 = x_{j2}{}^g, j2 \neq i, j1 \tag{8}$$

$$r_3 = x_{j3}{}^{g}, j3 \neq i, j1, j2 \tag{9}$$

$$r_d = r_2 - r_3 \tag{10}$$

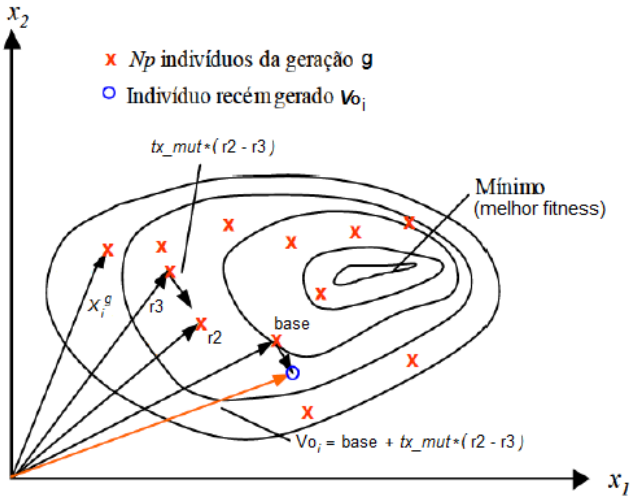
O vetor *base*, juntamente com o vetor *diferenças*, forma o vetor *mutação* (Vo), como pode ser conferido na Equação (11):

$$Vo_i = base + tx\_mut * r_d \tag{11}$$

Onde, tx_mut é o fator de mutação ($tx\_mut \in [0,2]$), normalmente representado pela letra *F*, assim como em Costa (2010), que controla a amplificação do vetor *diferenças*.

A Figura 13 mostra um exemplo de obtenção do vetor Vo_i para uma determinada função *fitness* com universo de busca bidimensional:

Figura 13 – Processo de obtenção do Vetor Vo_i para uma função bidimensional



Fonte: COSTA, 2010 (modificado pelo autor).

A posteriori, esse vetor *mutação* (Vo_i) será utilizado em conjunto com o indivíduo corrente ($x_i{}^{g}$) na etapa de recombinação. O vetor *prova* é, então, gerado através da operação de *crossover*, como pode-se conferir a seguir na Equação (12):

$$Vp = Vp(k) = \begin{cases} Vo_i(k)\ se\ rand[0,1] < tx\_cross \\ x_i{}^{g}(k)\ Caso\ contrário \end{cases} \tag{12}$$

No qual, $k \in [1, M]$, e tx_cross é a probabilidade de cruzamento ($tx\_cross \in [0,1]$, 1 equivale a 100% e 0 a 0%) e é representada normalmente na literatura por *Cr*. (COSTA, 2010).

Se depois do cruzamento, algum componente do vetor de *prova* estiver fora do universo de busca, seu valor deve ser limitado nos extremos da região de procura (região esta definida pelo usuário). Confira na Equação (13):

$$\begin{cases} Vp(i,k) = Amín(k)\ se\ Vp(i,k) < Amín(k) \\ Vp(i,k) = Amáx(k)\ se\ Vp(i,k) > Amáx(k) \end{cases} \quad (13)$$

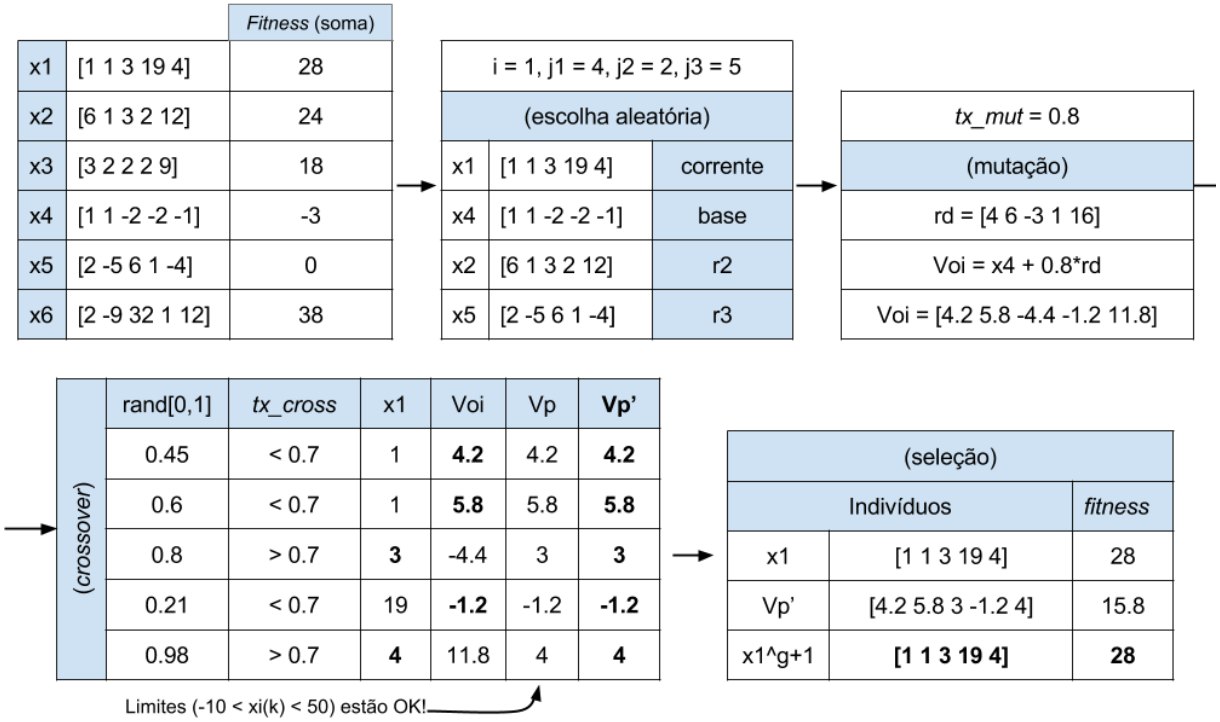
Onde, $Amín(k)$ e $Amáx(k)$ são os limites inferior e superior, respectivamente, de cada componente k do vetor Vp. Para o caso de controladores clássicos, onde os vetores ${x_i}^g$ são compostos por três ganhos (${x_i}^g = [Kp\ Kd\ Ki]$), só essa correção já é suficiente. Entretanto, no projeto desenvolvido, ${x_i}^g$ pode ser composto pelas localizações dos conjuntos das variáveis *Fuzzy* do controlador inteligente. Portanto, a ordem dos conjuntos para cada variável linguística importa, sendo então, também, necessário ordená-los.

Por fim, a *fitness* desses possíveis novos indivíduos (Vp) é calculada e assim dá-se início a seleção. Esse último operador compara a *fitness* dos indivíduos sem o efeito das operações de mutação e crossover (x^g) com a dos gerados em Vp, o individuo com maior *fitness* será selecionado para a nova geração, e o outro será descartado. Esse processo pode ser visto na Equação (14):

$$\begin{cases} {x_i}^{g+1} = Vp(i)\ se\ fitness\big(Vp(i)\big) > fitness({x_i}^g) \\ {x_i}^{g+1} = {x_i}^g\ Caso\ contrário \end{cases} \quad (14)$$

Na Figura 14 pode-se conferir um exemplo do funcionamento do algoritmo de evolução diferencial com todas as etapas acima citadas:

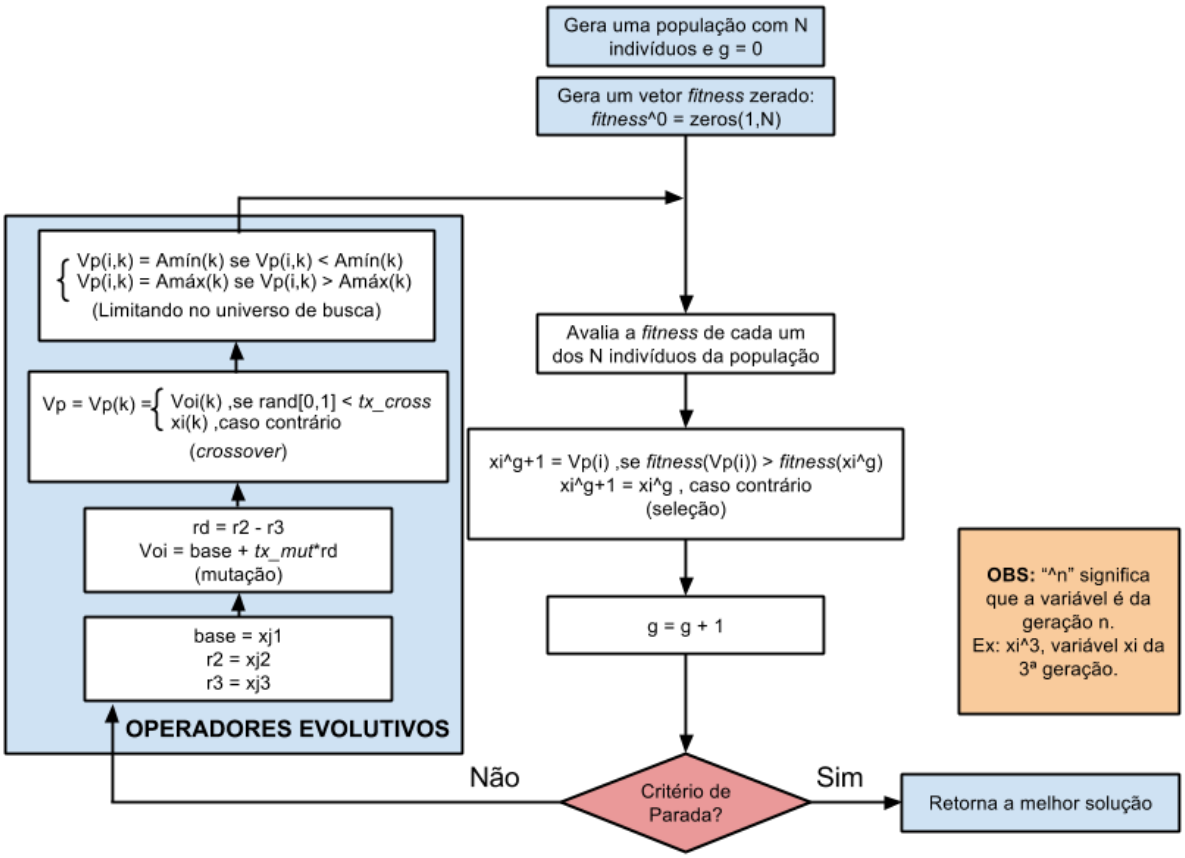
Figura 14 – Exemplo de funcionamento do algoritmo de evolução diferencial



Fonte: Produção do próprio autor.
Nota: Na etapa de *crossover* e de seleção, os números em negrito equivalem ao valores que farão parte do vetor prova e do indivíduo da nova geração, respectivamente

Esse processo de criação de novas populações entre gerações continua até algum dos critérios de parada ser obedecido. Como os algoritmos evolutivos utilizados possuem dinâmicas de funcionamento semelhantes, os critérios de parada válidos para o algoritmo genético também podem ser usados no algoritmo de evolução diferencial. Quando o critério for alcançado, o programa, finalmente, retornará a melhor solução encontrada e o ciclo chegará ao fim. As etapas do funcionamento básico do algoritmo de evolução diferencial podem ser resumidos pelo fluxograma da Figura 15:

Figura 15 – Fluxograma de funcionamento básico do algoritmo de evolução diferencial



Fonte: Produção do próprio autor.
Nota: g representa a geração corrente

Muitos trabalhos científicos, como Costa (2010) e Saad, Jamaluddin e Darus (2012), trazem o algoritmo da evolução diferencial como uma solução eficiente de otimização. Costa (2010) utiliza tanto os algoritmos genético e de evolução diferencial quanto métodos analíticos para realizar a estimação dos parâmetros de módulos fotovoltaicos. Já Saad, Jamaluddin e Darus (2010) mostram comparativamente os dois algoritmos utilizados por Costa além do método de Ziegler-Nichols para sintonizar controladores PID para três plantas de características distintas, a fim de testar a eficácia de cada técnica.

Normalmente, o algoritmo de evolução diferencial é utilizado na busca de valores para parâmetros de processos industriais ou na sintonia de ganhos de um controlador puramente linear. Entretanto, assim como a ideia, já comentada, de otimização de termos e ganhos *Fuzzy* via genético, este método também poderia ser utilizado para realizar a síntese de um controlador PID com núcleo *Fuzzy*.

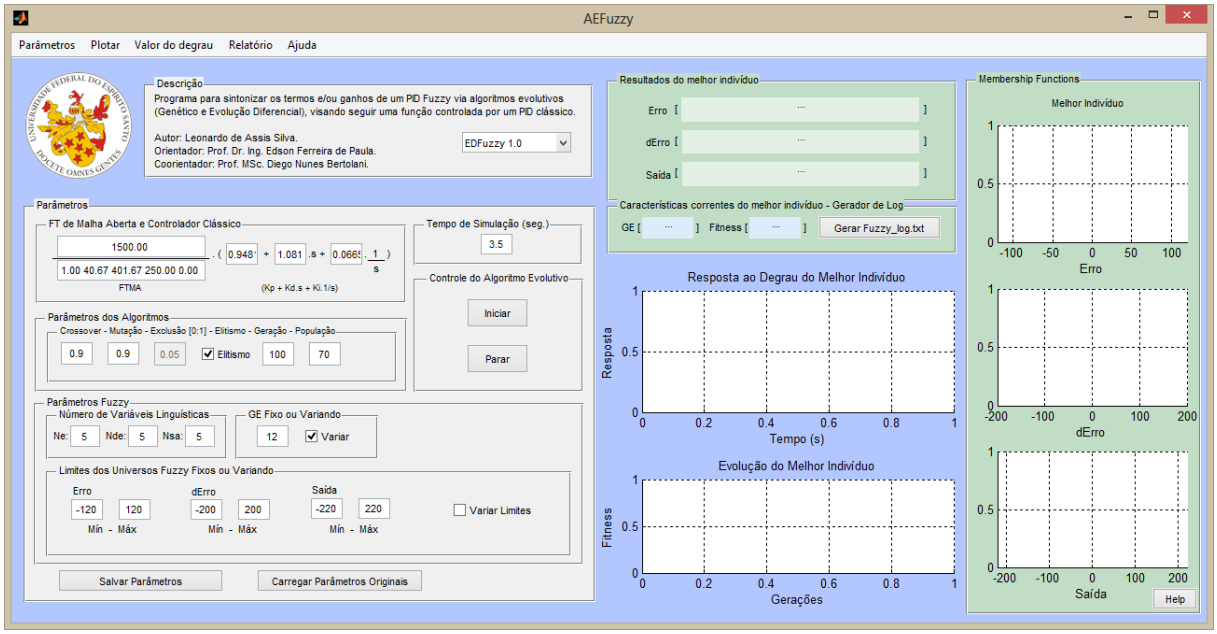
3 *SOFTWARE* AEFUZZY

Essa seção é destinada a explanação da interface e do funcionamento da ferramenta desenvolvida AEFuzzy (Algoritmos Evolutivos para síntese de controladores *Fuzzy*). Este programa foi implementado no MATLAB® e sua interface elaborada através do *toolbox* (pacote de ferramentas) gráfico GUIDE.

3.1 Descrição da interface

O desenvolvimento da interface foi norteada para a obtenção de alguns atributos de qualidade, como a facilidade de configuração e utilização do programa, além de prover uma condução adequada do usuário nos passos do programa, evitando erros e desvios desnecessários. Uma visão global da interface pode ser conferida abaixo na Figura 16:

Figura 16 – Interface da ferramenta desenvolvida, AEFuzzy

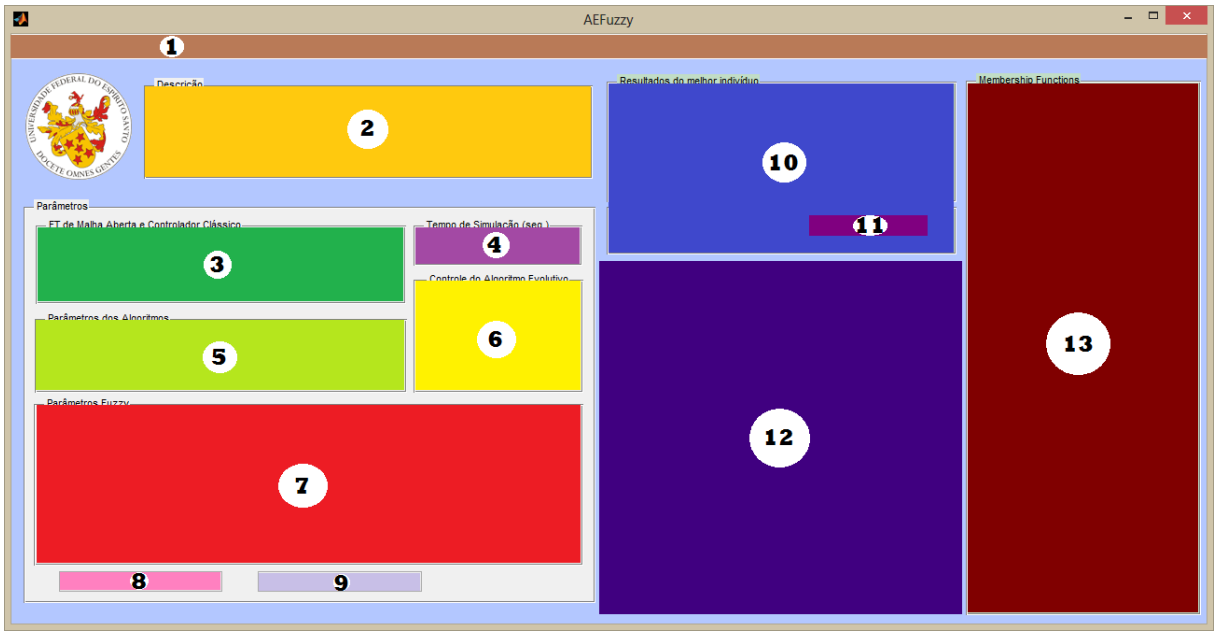


Fonte: Produção do próprio autor.

A tela inicial do programa é dividida em regiões bem definidas, conforme mostra a Figura 17, identificadas por números que vão de 1 a 13. No espaço 1 se encontra o menu do topo, que disponibiliza algumas opções extras do programa. Nessa área o usuário pode salvar ou carregar os parâmetros de configuração (salvo em um documento de texto). Também é possível escolher os pontos inicial e final da entrada em degrau da simulação, traçar os

gráficos normalmente desejados na síntese de controladores: resposta ao degrau da referência e do melhor indivíduo obtido, com ou sem saturação do esforço de controle; resposta ao degrau dos melhores indivíduos de cada geração; esforço de controle da referência e do melhor indivíduo, e gerar uma pasta com os arquivos essenciais para documentação dos resultados obtidos: arquivos de simulação do Simulink; o arquivo do Sistema de Inferência *Fuzzy* (extensão .FIS); e um relatório completo das gerações em um documento de texto, que será explicado melhor na área 11. Uma aba de ajuda está disponível para dar exemplos de configuração e contatos para suporte ao usuário.

Figura 17 – Identificação das regiões do programa AEFuzzy



Fonte: Produção do próprio autor.

Na área 2 o AEFuzzy permite que o usuário escolha entre duas opções de algoritmos evolutivos: AGFuzzy 2.0 e EDFuzzy 1.0 (segunda versão e primeira versão, nessa ordem, indicando o número de vezes que o código foi alterado visando a diminuição do tempo de processamento), que correspondem, respectivamente, ao algoritmo genético e ao algoritmo de evolução diferencial. Além disso, neste mesmo espaço encontra-se uma breve descrição das funcionalidades do *software* e os nomes do autor e dos professores que orientaram no projeto.

Na região 3 existe um espaço reservado para que sejam colocados o numerador e o denominador da função de transferência de malha aberta da planta, e os parâmetros do controlador PID clássico a ser copiado.

Na área 4, o programa dá ao usuário a possibilidade de escolher o tempo de simulação. Esse parâmetro está ligado diretamente à dinâmica do sistema compensado. Isso é, caso existam comportamentos que se deseja copiar para o controlador *Fuzzy* que ocorram, por exemplo, até 2 segundos de simulação, nesse campo, deverá ser colocado, então, pelo menos 2 segundos, para que a curva da resposta ao degrau do sistema englobe essa dinâmica.

No espaço 5 encontram-se os parâmetros dos algoritmos a serem escolhidos pelo usuário, como: taxa de *crossover*, taxa de mutação, taxa de exclusão (exclusiva do algoritmo genético), elitismo, que implica que o melhor indivíduo terá 100% de chances de aparecer na próxima geração, o número de gerações, e o tamanho da população. Em 6 o usuário pode iniciar o processo de síntese dos algoritmos e também parar seu funcionamento no instante desejado.

A área 7 possibilita o usuário configurar os parâmetros *Fuzzy* do sistema: número de variáveis linguísticas das entradas (erro e derivada do erro) e da saída (saída), o valor do ganho *GE* da estrutura de controle *Fuzzy*, se esse ganho pode variar até o valor digitado, e o valor dos limites do universo de cada variável linguística do sistema *Fuzzy*. Como para o ganho *GE*, também é possível escolher se os limites do universo *Fuzzy* do erro, da derivada do erro e da saída podem variar de 0 até os valores digitados.

As região 8 e 9 são atalhos para opções da área 1, onde é possível, respectivamente, salvar e carregar os parâmetros de configuração dos espaços 3, 4, 5 e 7, já citados. Durante o funcionamento do algoritmo, o programa disponibiliza na região 10 o resultado do melhor indivíduo até o momento (parâmetros *Fuzzy*, ganho *GE* e *fitness*), e quando chega ao fim, a melhor solução encontrada.

Na área 11 encontra-se um botão responsável por gerar um relatório em um documento de texto contendo o algoritmo utilizado, os parâmetros do algoritmo, da função de transferência de malha aberta, do controlador clássico, e da resposta ao degrau do melhor indivíduo encontrado e de cada geração, como pode-se conferir em um exemplo mostrado nas Figuras 18 e 19 que seguem:

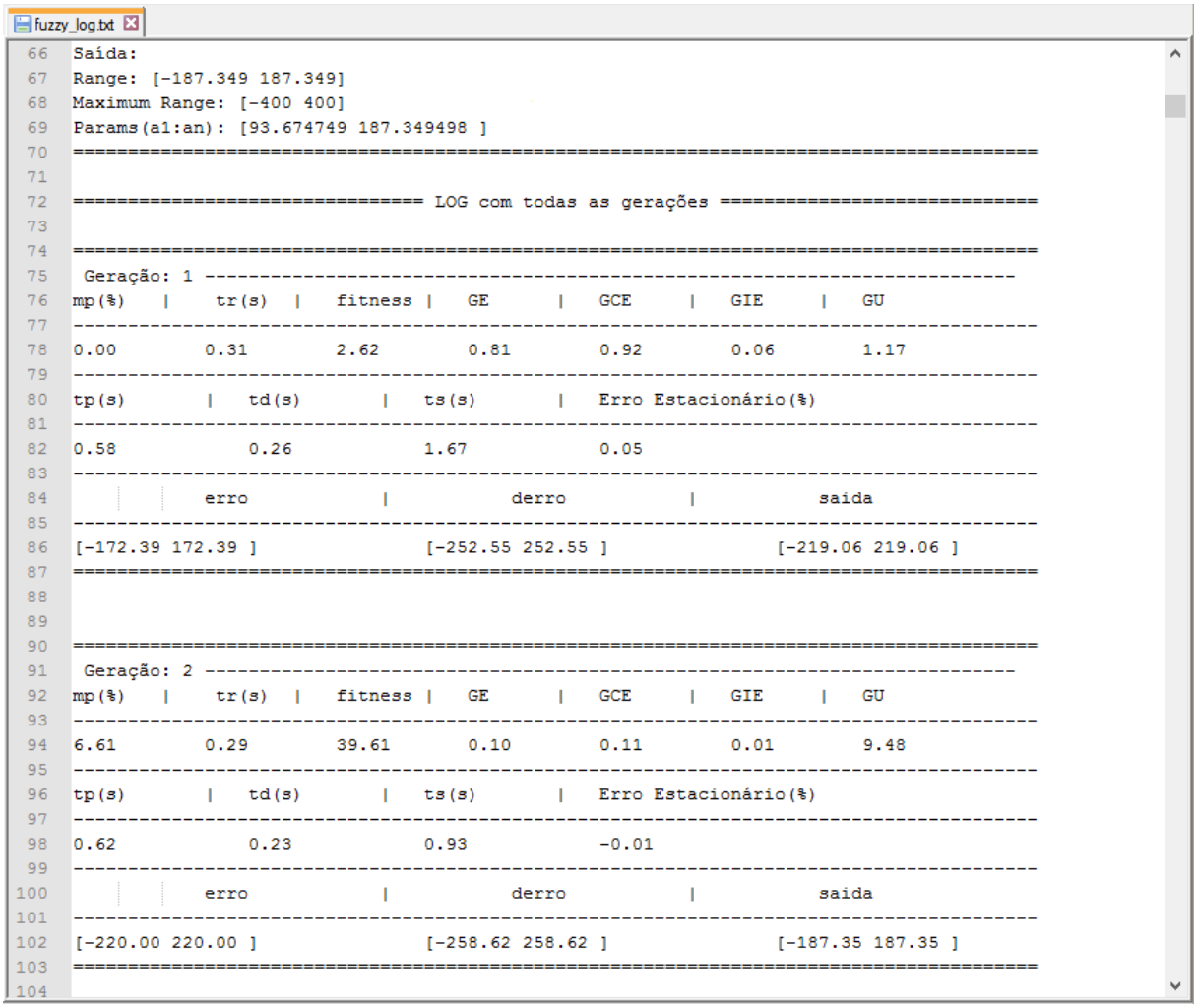
Figura 18 – Relatório gerado pela ferramenta AEFuzzy

fuzzy_log.txt

```
======================== Resultados obtidos do melhor indivíduo ========================

-------------------------------------------------------------------------------------------
- Parâmetros do ALGORITMO DE EVOLUÇÃO DIFERENCIAL utilizado (EDFuzzy 1.0) -

O algoritmo é ELITISTA; o ganho GE VARIA; são os LIMITES QUE VARIAM nessa otimização, ou seja,
modificam o universo fuzzy enquanto os termos ficam igualmente espaçados. Além disso alguns
parâmetros importantes do algoritmo evolutivo, do sistema fuzzy e da simulação seguem abaixo:

CROSSOVER: 90.00%, MUTAÇÃO: 90.00%, EXCLUSÃO: 5.00%, nº GERAÇÕES: 100, tamanho da POPULAÇÃO: 70.
TEMPO DE SIMULAÇÃO: 3.50 s, DEGRAU INICIAL: 0.00, DEGRAU FINAL: 1.00.
Número de variáveis linguísticas - Ne: 5, Nde: 5, Nsa: 5.

(Kd*s^2 + Kp*s + Ki) = (1.0810*s^2 + 0.9481*s + 0.0665)
---------------------   ---------------------------------
        s                           s

FTMA:
sys =

             1500
  ---------------------------------
  s^4 + 40.67 s^3 + 401.7 s^2 + 250 s

Continuous-time transfer function.



-------------------------------------------------------------------------------------------
- Parâmetros da resposta do melhor indivíduo -

Overshoot: 6.61%
Fitness: 39.6149
Tempo de Subida: 0.29 s
Tempo de Pico: 0.62 s
Tempo de Atraso: 0.23 s
Tempo Estacionário: 0.93 s
Erro Estacionário: -0.01

-------------------------------------------------------------------------------------------
- Parâmetros Fuzzy do melhor indivíduo -

Última Geração: 3

GE:
Range: [0 20.000]
Value: 0.1000

GCE:
Value: 0.1140

GIE:
Value: 0.0070

GU:
Value: 9.4810

Erro:
Range: [-220.000 220.000]
Maximum Range: [-220 220]
Params(a1:an): [110.000000 220.000000 ]

dErro:
Range: [-258.621 258.621]
Maximum Range: [-300 300]
Params(a1:an): [129.310547 258.621093 ]
```

Fonte: Produção do próprio autor.

Figura 19 – Relatório gerado pela ferramenta AEFuzzy



fuzzy_log.txt

```
Saída:
Range: [-187.349 187.349]
Maximum Range: [-400 400]
Params(a1:an): [93.674749 187.349498 ]
================================================================================

================================ LOG com todas as gerações ================================

================================================================================
 Geração: 1 ------------------------------------------------------------------------
mp(%)    |    tr(s)   |   fitness |   GE      |   GCE     |   GIE     |   GU
--------------------------------------------------------------------------------
0.00        0.31        2.62        0.81        0.92        0.06        1.17
--------------------------------------------------------------------------------
tp(s)       |   td(s)       |   ts(s)       |   Erro Estacionário(%)
--------------------------------------------------------------------------------
0.58            0.26            1.67            0.05
--------------------------------------------------------------------------------
        erro            |           derro           |           saida
--------------------------------------------------------------------------------
[-172.39 172.39 ]               [-252.55 252.55 ]               [-219.06 219.06 ]
================================================================================


================================================================================
 Geração: 2 ------------------------------------------------------------------------
mp(%)    |    tr(s)   |   fitness |   GE      |   GCE     |   GIE     |   GU
--------------------------------------------------------------------------------
6.61        0.29        39.61       0.10        0.11        0.01        9.48
--------------------------------------------------------------------------------
tp(s)       |   td(s)       |   ts(s)       |   Erro Estacionário(%)
--------------------------------------------------------------------------------
0.62            0.23            0.93            -0.01
--------------------------------------------------------------------------------
        erro            |           derro           |           saida
--------------------------------------------------------------------------------
[-220.00 220.00 ]               [-258.62 258.62 ]               [-187.35 187.35 ]
================================================================================

```

Fonte: Produção do próprio autor.

Na seção 12, o usuário pode conferir em tempo de execução a resposta ao degrau do melhor indivíduo, além de um gráfico mostrando a evolução do indivíduo através de uma curva relacionando as gerações com a *fitness* dos melhores membros.

A área 13, assim como a 12, mostra informações em tempo de execução do melhor indivíduo, mas dessa vez, apresentando a configuração dos conjuntos *Fuzzy* das variáveis observadas: erro (*Erro*), derivada do erro (*dErro*) e da saída (*Saída*).

3.2 Funcionamento interno

Esta seção está destinada a apresentação do funcionamento de partes do código da ferramenta desenvolvida, mais especificamente no processo dos algoritmos evolutivos utilizados até a geração de seus resultados e na estrutura do controlador *Fuzzy* adotado.

Após o programa ser aberto, a interface coleta os valores de todas as variáveis importantes para o funcionamento do algoritmo. Em seguida, o programa espera o usuário apertar o botão "Iniciar" que se encontra na região 6 da interface para que o processo de síntese possa começar.

Opções como a geração de relatórios do espaço 11 e traçar gráficos da área 1, que dependam dos resultados da ferramenta após o processo de sintonia do controlador, não farão nada, ou para o caso da área 11, gerará um documento bem simples apenas com as informações obtidas na interface caso o programa não tenha sido usado nenhuma vez desde sua inicialização.

Após começado o ciclo do algoritmo evolutivo pelo botão "Iniciar", as outras áreas serão desabilitadas para uso, visando impedir que parâmetros sejam alterados durante o funcionamento da ferramenta, salve a própria área 6 que precisa que seu botão "Parar" esteja habilitado para que seja possível encerrar a execução do programa a qualquer momento. Um texto em azul aparece na parte inferior do programa, escrito "Working..." ("funcionando..."), para sinalizar o início e todo o tempo de funcionamento do programa. Pode ser necessário esperar algum tempo para que o programa funcione após o botão "Iniciar" ser pressionado, pois a interface, por ter muitas funcionalidades, acaba processando um número grande de informações.

Nesta aplicação, o primeiro passo compreende a criação da primeira população. Cada indivíduo é formado por um conjunto de cromossomos, podendo estes ser o ganho *GE*, caso tenha sido escolhido que seu valor varie até o digitado, e termos referentes ao núcleo *Fuzzy*. Este último depende também de como foi configurada a interface, pois se o usuário marcar na região 7 que deseja que os limites do universo *Fuzzy* de cada variável varie, os indivíduos serão como segue nas Equações 15 e 16, com *erromax*, *derromax*, *saidamax* e *gemax* sendo os valores máximos que as variáveis *Erro*, *dErro*, *Saida* e *GE* podem alcançar. A variável

gevaria possui o valor de 1 caso tenha sido marcado na área 7 da interface para que *GE* tivesse seu valor variando, e 0 caso contrário.

$$individuo = [erromax * rand(1)\ derromax * rand(1)\ saidamax * rand(1)]; \quad (15)$$

$$\begin{aligned} &if(gevaria == 1) \\ &\quad individuo = [gemax * rand(1)\ individuo]; \\ &end \end{aligned} \quad (16)$$

Com os limites variando, os conjuntos *Fuzzy* ficam igualmente espaçados, se espalhando pelo universo de forma uniforme. Entretanto, caso a opção de variar os limites não seja marcada, os conjuntos *Fuzzy* terão as suas posições variando, portanto, os indivíduos serão criados como pode ser visto nas Equações 17 e 18, onde, *Ne*, *Nde* e *Nsa* correspondem ao número de variáveis linguísticas a serem sintonizadas do *Erro*, *dErro* e *Saida*, respectivamente. Para o correto funcionamento do bloco *Fuzzy* do Simulink em conjunto com o programa AEFuzzy, os termos de cada variável são colocados em ordem crescente pela função *sort()* do MATLAB®.

$$\begin{aligned} individuo = [&sort(erromax * rand(1, Ne), 2)\ sort(derromax \\ &* rand(1, Nde), 2)\ sort(saidamax * rand(1, Nsa), 2)]; \end{aligned} \quad (17)$$

$$\begin{aligned} &if(gevaria == 1) \\ &\quad individuo = [gemax * rand(1)\ individuo]; \\ &end \end{aligned} \quad (18)$$

Nota-se que o número de variáveis linguísticas escolhidos na área 7 equivale ao número de conjuntos *Fuzzy* que cada variável de entrada e de saída terá. Como o sistema se comporta de forma análoga para valores negativos e positivos do erro e de sua derivada, o número de termos a serem sintonizados pode ser reduzido pela metade. Os conjuntos utilizados são trapezoidais nos extremos e triangulares nos demais.

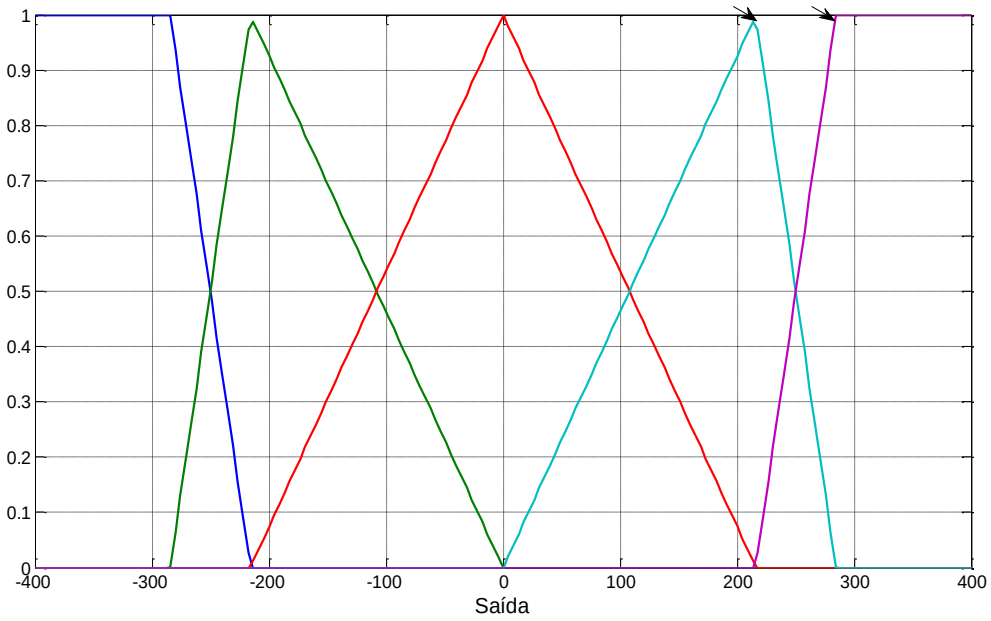
Para que os conjuntos do lado positivo e do negativo se posicionem de forma simétrica, o termo triangular central deve ter sua ponta (ponto de máxima pertinência) posicionada em 0 (posição central do universo *Fuzzy* da variável). É também importante frisar que os conjuntos devem se cruzar com os adjacentes em um valor de pertinência igual a 0.5, visto que uma regra deve na soma de todas as possibilidades ter peso 100%. Com o cruzamento em 0.5,

nesse ponto, os dois conjuntos tem peso 50%, fazendo com que em qualquer outro ponto isso também ocorra (posto que, por serem triângulos e trapézios, seus lados são retas).

Isso limita o número de possíveis configurações para a solução, diminuindo o tamanho dos indivíduos: o primeiro ponto do triângulo tem pertinência nula e, por isso, deve se posicionar no mesmo lugar que o ponto de máximo do conjunto anterior (pertinência 100%); o segundo ponto tem pertinência máxima, portanto, fica na mesma posição que o ponto de mínimo dos conjuntos a volta; o terceiro e último ponto do triângulo também possui pertinência 0% e fica no ponto de máxima do próximo conjunto. Os conjuntos de trapézio são formados por um ponto a mais, que por ter pertinência 1, se encontra em um extremo, onde os outros conjuntos possuem pertinência nula, somando 100%.

Pode-se concluir disso que os únicos pontos a serem sintonizados pelo AEFuzzy, ou seja, que devem entrar na formação dos indivíduos, são as pontas dos triângulos e dos trapézios que possuem pertinência 1 (como o trapézio possui dois pontos de formação com pertinência 1, somente o ponto que não se localiza no extremo do universo será usado). Uma ilustração disso pode ser conferida na Figura 20, e a maneira como calcular *Ne*, *Nde* e *Nsa* exibida nas Equações 19, 20 e 21:

Figura 20 – Marcação dos pontos a serem sintonizados para a variável Saída



Fonte: Produção do próprio autor.

$$Ne = \frac{ne - 1}{2} \tag{19}$$

$$Nde = \frac{nde - 1}{2} \tag{20}$$

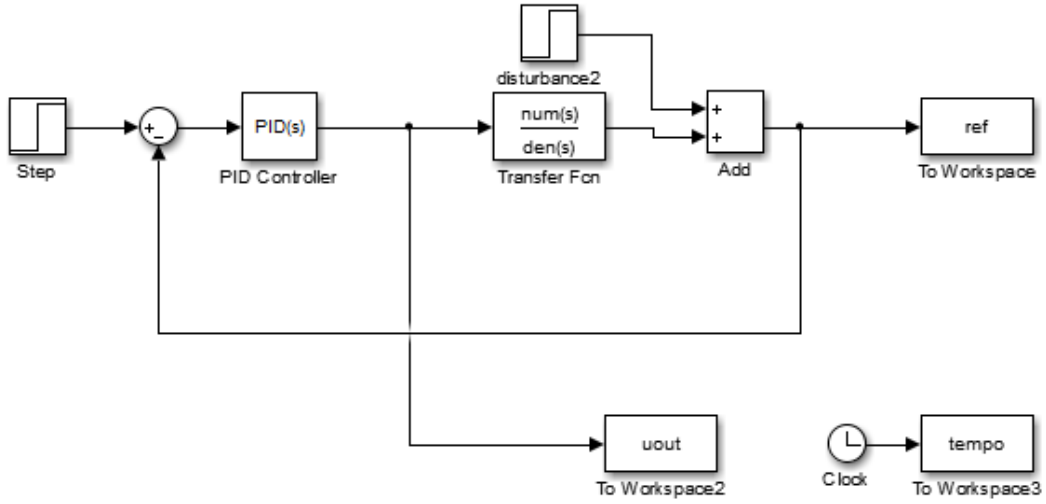
$$Nsa = \frac{nsa - 1}{2} \tag{21}$$

Nas Equações 19, 20 e 21, as variáveis *ne*, *nde* e *nsa* são os números de variáveis linguísticas escolhidos na região 7 da interface. Esses valores, como já explicado, são reduzidos à metade, retirando também o triângulo central, de forma a otimizar o processo de busca. Assim sendo, como exemplo, os pontos que formarão o indivíduo na Figura 20 (pontos a serem sintonizados) estão destacados por duas setas (*nsa* igual a 5 conjuntos, *Nsa* = 2).

Esses indivíduos, independente do algoritmo escolhido, serão avaliados de acordo com uma função *fitness*. Após avaliados, os indivíduos são colocados em ordem na população do melhor para o pior de acordo com suas respectivas aptidões através de uma função chamada *quicksort()* (implementada pelo autor) que ordena de forma otimizada uma matriz. A função *fitness* é calculada no programa como é mostrado na Equação 22. A Figura 21 mostra como a função *fitness* (*ref*) é gerada pelo Simulink. O sinal *y* na Equação 22 é gerado pela simulação da malha apresentada na Figura 8.

$$fitness = \frac{1}{\sum_{k=1}^{N}(ref[k] - y[k])^2} \tag{22}$$

Figura 21 – Estrutura da função de referência para o cálculo da fitness



Fonte: Produção do próprio autor.

Analisando o cálculo da função *fitness*, pode-se notar que se busca um indivíduo de aptidão máxima, já que se tenta minimizar a diferença quadrática entre a resposta ao degrau do indivíduo (*y*) e a função *fitness* (*ref*). É possível ver também que a função *fitness* do programa é a própria função de transferência de malha aberta já compensada, que é passada para o algoritmo pela região 3 da interface.

Depois de calculadas as aptidões de todos os indivíduos e do ordenamento da população, os resultados do melhor são mostrados nas áreas 10, 12 e 13 da interface. O programa, então, confere se o botão "Parar" foi pressionado ou se a geração já chegou a seu valor máximo. Se nenhum dos critérios de parada forem obedecidos, o algoritmo escolhido começa a criar uma nova população para a geração seguinte.

Para o caso de o algoritmo genético ter sido selecionado, o programa gerará a nova população através da mutação e do *crossover*. O cruzamento foi implementado como segue nas Equações 23 até 26:

$$m = population(1{:}\,2{:}\,end-1,:);$$
$$f = population(2{:}\,2{:}\,end,:); \tag{23}$$

$$xss = rand(size(m,1),1);$$
$$e = round\Big(2 + (size(m,2)-2) * rand\big(1, size(m,1)\big)\Big); \tag{24}$$

$$
\begin{aligned}
&for\ k = 1{:}\ size(m,1)\\
&\ \ if\ xss(k) \le tx\_cross\\
&\ \ \ \ m(k,:) = \left[m\left(k,1{:}\left(e(k)-1\right)\right) f(k,e(k){:}\,end)\right];\\
&\ \ \ \ f(k,:) = \left[f\left(k,1{:}\left(e(k)-1\right)\right) m(k,e(k){:}\,end)\right];\\
&\ \ end\\
&end
\end{aligned}
\tag{25}
$$

$$
\begin{aligned}
&new\_population\ = zeros(size(population));\\
&new\_population(1{:}\,2{:}\,end-1,:)\ = m;\\
&new\_population(2{:}\,2{:}\,end,:)\ = f;
\end{aligned}
\tag{26}
$$

Percebe-se que a operação de cruzamento é realizada a cada par de macho (*m*) e fêmea (*f*). Os machos equivalem aos indivíduos de índice ímpar da população, e as fêmeas de índice par (Equação 23). Dois vetores são criados com valores randômicos de 0 a 1 (*xss*), e com números de 2 ao tamanho do indivíduo (*e*) na Equação 24. O vetor *xss* será comparado com a taxa de *crossover* (*tx_cross*) escolhida pelo usuário na região 5 para cada par de indivíduos. Caso seja menor que a taxa de cruzamento, haverá a troca de genes entre os pais (*m* e *f*), sendo os vetores macho e fêmea quebrados para o *crossover* na posição definida por *e*. Isso é conferido na Equação 25, onde um *loop* (ciclo de iterações, que neste caso foi representado por um *for*) realiza a varredura dos indivíduos. Na Equação 26 a nova população, antes dos efeitos da mutação, é construída com os novos machos e fêmeas.

Esses operadores genéticos não realizam suas funcionalidades sobre valores binários. O objetivo desta pesquisa não entra no mérito de otimização de operações próprias do AG Algorítmo Genético) ou da ED (Evolução Diferencial), sendo o mais importante utilizar as ferramentas de modo correto. Portanto, foi escolhido trabalhar com valores decimais. É interessante citar que foram efetuados testes com valores binários apenas no momento de realizar as operações genéticas, porém, não foi observada qualquer vantagem em relação ao modo adotado. A mutação implementada pode ser conferida nas Equações 27, 28 e 29 que seguem:

$$
\begin{aligned}
taxa\_mutacao &= rand\left(size(population)\right);\\
mut &= taxa\_mutacao\ <= tx\_mut;
\end{aligned}
\tag{27}
$$

$$
\begin{aligned}
&if\ gevaria == 1\\
&\ mut\_popu = [erromax * ones(size\_of\_population, Ne)\\
&\qquad derromax * ones(size\_of\_population, Nde)\\
&\qquad saidamax * ones(size\_of\_population, Nsa)].* rand(size(\\
&\qquad population));\\
&else\\
&\ mut\_popu = [gemax * ones(size\_of\_population, 1)\\
&\qquad erromax * ones(size\_of\_population, Ne)\\
&\qquad derromax * ones(size\_of\_population, Nde)\\
&\qquad saidamax * ones(size\_of\_population, Nsa)].* rand(size(\\
&\qquad population));\\
&end
\end{aligned}
\tag{28}
$$

$$
new\_population(logical(mut)) = mut\_popu(logical(mut));
\tag{29}
$$

Na Equação 27, uma matriz (*taxa_mutacao*) de mesmas dimensões que a matriz população (*population*) é criada, preenchida com números randômicos que variam de 0 a 1. Isso significa que todos os componentes de todos os indivíduos podem sofrer mutação, ou seja, caso algum componente da matriz *taxa_mutacao* seja menor ou igual a taxa de mutação (*tx_mut*) definida pelo usuário na região 5, o valor dessa posição na matriz *population* será recriado do zero. Pode-se conferir essa substituição dos valores na Equação 29. Os valores criados estão mostrados na Equação 28 na variável *mut_popu*.

Como já foi visto na seção 2.2.1, antes de acontecer a reprodução dos indivíduos, estes são selecionados dentro da população através de uma roleta (método utilizado no programa, mais opções na literatura são: por campeonato, por classificação, por Boltzman, entre outras), o que quer dizer que nem todos os indivíduos passarão seus genes para a próxima geração, apenas os mais aptos. Estes indivíduos selecionados são colocados dentro de uma matriz chamada *population*, que já foi apresentada nas Equações de 23 a 29. O código da roleta é apresentado nas Equações 30, 31 e 32 abaixo:

$$
\begin{aligned}
&sft = sum(fitness);\\
&psel = fitness/sft;\\
&cpsel = cumsum(psel);\\
&N = size(population, 1);
\end{aligned}
\tag{30}
$$

$$\begin{aligned} &II = zeros(1, N); \\ &for\ ii = 1{:}N \\ &\quad r = rand(1); \\ &\quad\ R = cpsel > r; \\ &\quad\ I = find(R, 1); \\ &end \end{aligned} \tag{31}$$

$$population = population(II, :) \tag{32}$$

Nota-se na Equação 30 que a implementação da roleta é feita transformando o vetor que contém a *fitness* de todos os indivíduos em ordem decrescente (*fitness*) em valores percentuais da soma de todas as aptidões, ou seja, do total (*psel*). Em seguida, um vetor (*cpsel*) é criado contendo a soma acumulativa do vetor *psel*, que contém as quantias percentuais. Isso é feito para que este represente, de um indivíduo para o próximo, o quanto já foi acumulado na roleta. Um exemplo pode ser conferido na Figura 22:

Figura 22 – Exemplo do funcionamento do princípio do código da roleta

Indivíduo	**Fitness** (*fitness*)	**Fitness (%/100)** (*psel*)	**Soma acumulativa** (*cpsel*)
x1	8.1261	0.4841	0.4841
x2	7.0695	0.4211	0.9052
x3	0.3079	0.0183	0.9236
x4	0.2271	0.0135	0.9371
x5	0.2097	0.0125	0.9496
x6	0.2044	0.0122	0.9618
x7	0.1719	0.0102	0.9720
x8	0.1616	0.0096	0.9816
x9	0.1558	0.0093	0.9909
x10	0.1528	0.0091	1.0000
Total:	16.7867 (*sft*)	**1 (100%)**	

Fonte: Produção do próprio autor.

A Equação 31 demonstra como os valores contidos em *cpsel* podem escolher os indivíduos que participarão da reprodução. Percebe-se que subtrair um componente do vetor *cpsel* do seu anterior significa dizer o quanto esse componente adicionou a formação da roleta para

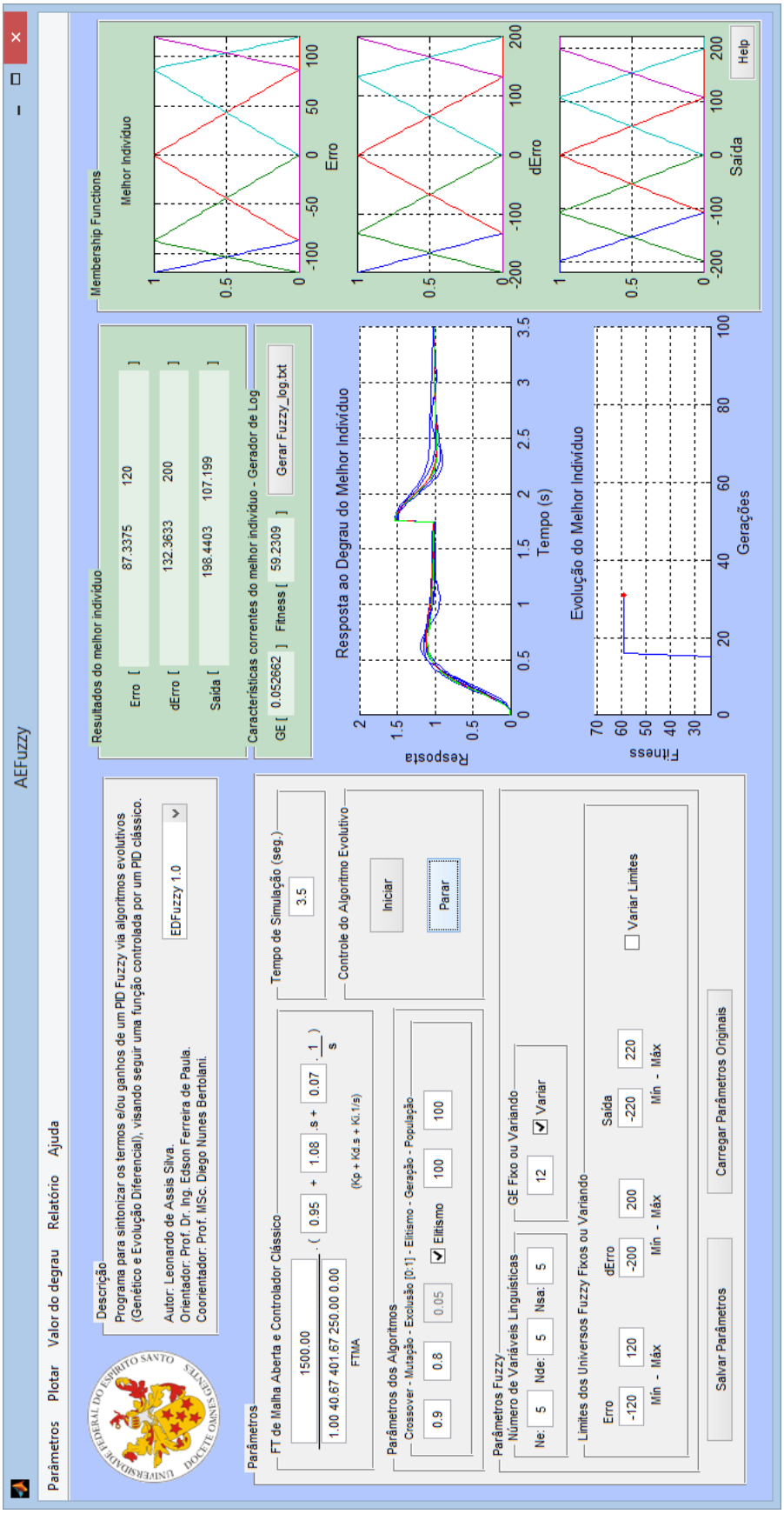
totalizar 100%, logo, o primeiro valor de *cpsel* maior que um determinado número randômico (*r*) representa a região da roleta em que este número se encontra. Pode-se fazer uma analogia com o lançamento um dardo em um círculo girante. A maior região (indivíduo x1 da Figura 22 com 41.48%) teria mais chances de ser acertada, enquanto as demais possuiriam menos possibilidades. Um valor aleatório de 0 a 0.4840 indicaria que o primeiro indivíduo foi escolhido. Já um valor de 0.4841 a 0.9051 (42.11% de chances) significaria que o indivíduo x2 foi o eleito, e um valor entre 0.9909 a 1 (0.91% de chances) indicaria o indivíduo x10. Esta análise é válida para todas as vezes que a roleta for girada.

Desta forma, na Equação 31, um vetor *R* é preenchido com as posições obtidas no funcionamento da roleta, e na Equação 32, o vetor *population* é refeito com os N indivíduos escolhidos. Nota-se que o tamanho da população não diminui em momento algum, pois mesmo que esta perca indivíduos pela sua taxa de exclusão (definida pelo usuário), que elimina os indivíduos menos aptos, a roleta será rodada N vezes, ou seja, um número igual ao tamanho da população.

Para o algoritmo de evolução diferencial, como suas operações são basicamente somas, subtrações e multiplicações, seu código não será detalhado. A representação do algoritmo foi feita exatamente como explicado na seção 2.2.2. O código completo pode ser conferido no Apêndice A.

A Figura 23 ilustra um teste da ferramenta AEFuzzy que, após executada, o botão “Parar” foi pressionado para apresentar os resultados finais. É possível notar na área 10 os valores do posicionamento dos conjuntos *Fuzzy* de cada variável sintonizada, o valor do ganho GE obtido, e o valor da aptidão do melhor indivíduo encontrado. Na região 12, pode ser conferida a resposta ao degrau da função *fitness* em verde tracejado, do melhor indivíduo de cada geração em azul e do melhor indivíduo encontrado em vermelho. Além disso, também está disponível o andamento da aptidão do melhor indivíduo de cada geração em azul e do melhor indivíduo encontrado em vermelho. No espaço 13, são apresentados os posicionamentos dos conjuntos *Fuzzy* de cada variável do melhor indivíduo encontrado.

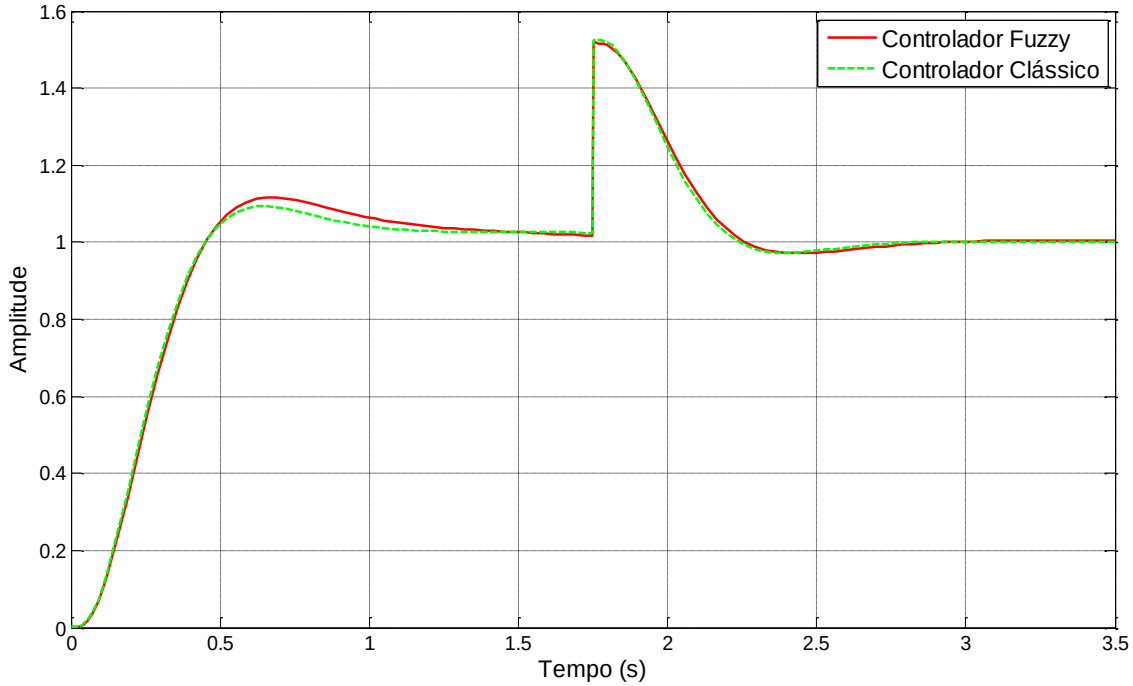
Figura 23 – Exemplo de funcionamento do programa AEFuzzy



Fonte: Produção do próprio autor.

Como exemplo, uma figura mostrando a resposta ao degrau do melhor indivíduo encontrado em vermelho e da função *fitness* em verde tracejado é gerada em uma nova janela e pode ser conferida na Figura 24. Na metade do tempo de simulação, uma perturbação é adicionada ao sistema.

Figura 24 – Resposta ao degrau do melhor indivíduo obtido pelo exemplo



Fonte: Produção do próprio autor.

4 RESULTADOS E DISCUSSÕES

A validação de uma ferramenta deve ser feita a partir de um conjunto de dados suficiente a colocar a prova todas as funcionalidades do programa desenvolvido. Por isso, 3 sistemas com diferentes características foram escolhidos para a realização dos testes: uma planta de ordem elevada, uma com atraso, e um sistema de fase não mínima. Estas podem ser conferidas nas Equações 33, 34 e 35, respectivamente, que seguem:

$$G_1(s) = \frac{1500}{s^4 + 40.67s^3 + 401.67s^2 + 250s} \tag{33}$$

$$G_2(s) = \frac{10e^{-1.0s}}{(1+8s)(1+2s)} = \frac{10 - 5s}{8s^3 + 21s^2 + 10.5s + 1} \tag{34}$$

$$G_3(s) = \frac{1 - 10s}{(1+s)^3} = \frac{1 - 10s}{s^3 + 3s^2 + 3s + 1} \tag{35}$$

A planta 1, apresentada na Equação 33, possui ordem 4 e tipo 1. Um sistema com um número considerável de polos, sendo que um polo encontra-se na origem, pode trazer certas complicações para o processo de sintonia de controladores. Quanto maior o número de polos, maiores podem ser as possibilidades de ocorrer instabilidade. O eixo imaginário separa o eixo real em dois: o esquerdo, onde o sistema será estável caso todos os polos estejam desse lado, e direito, instável para o caso de pelo menos um polo se localizar lá. Portanto, o eixo imaginário pode ser considerado o limiar de estabilidade do lugar geométrico das raízes.

A localização dos polos também afeta a velocidade de resposta do sistema. Quanto mais próximo da origem está o polo, mais lento fica o sistema. Isso acontece pelo fato de o polo aumentar a taxa de decaimento do módulo do sistema, diminuindo portanto a banda passante, caso se localize perto da origem. A vantagem deste fato é que uma banda passante menor evita que o sistema seja interferido por ruídos em frequências maiores. Outro ponto de vista seria na representação no domínio do tempo, onde quanto mais longe da origem estiver o polo, menor será seu efeito na dinâmica transitória do sistema com o passar do tempo, como pode-se conferir na Equação 36:

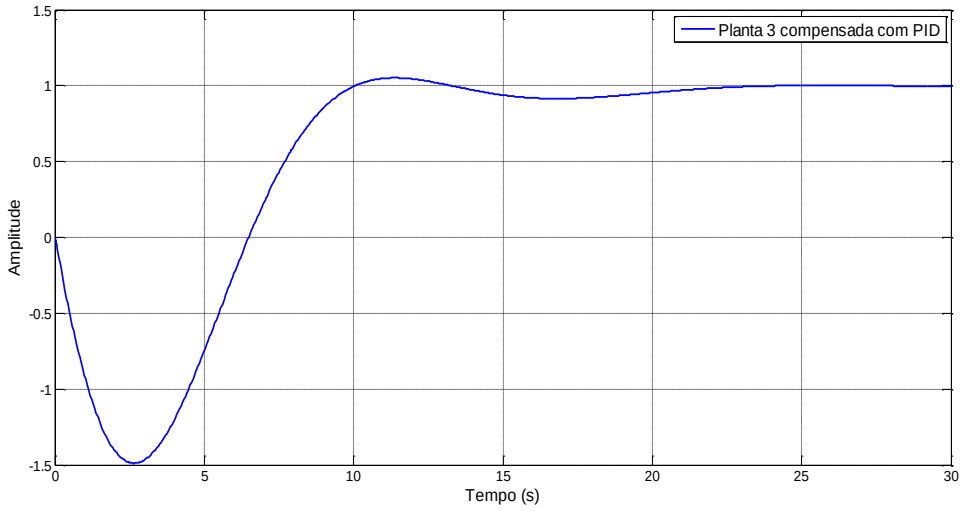
$$G_1(s) = \frac{Y(s)}{X(s)} = \frac{1500}{s(s+15)(s+\frac{10}{15})(s+25)}, X(s) = \frac{1}{s}$$
$$\downarrow L^{-1} \tag{36}$$
$$y(t) = 6t - 9.64u(t) - 0.046e^{-15t} + 9.676e^{-\frac{10t}{15}} + 0.098e^{-25t}$$

Nota-se ainda que além da planta 1 possuir um polo próximo da origem ($s = -\frac{10}{15}$), ela também tem um integrador ($s = 0$), ou seja, um polo que não só está no limiar de estabilidade como também sobre a origem. Em malha fechada, o projetista terá que ajustar a posição dos polos através de um controlador pensando nesses problemas citados.

A planta 2, apresentada na Equação 34, possui atraso, que é substituído por uma aproximação de Padé de primeira ordem. Na prática, atrasos podem ser encontrados em muitos tipos de sistema, como sistemas que envolvam hidráulica, em controle de velocidade de motores, pneumática, ou quando o controle é feito a partir de um computador, que precisa de certo tempo para realizar as operações matemáticas. O problema do atraso no sistema é intuitivo, pois atuar no sistema para resolver um problema quantificado certo tempo atrás vai introduzir oscilações piorando a estabilidade transitória. Dependendo da demora de resposta da planta, um sistema mesmo estável pode chegar a instabilidade.

A planta 3, representada na Equação 35, tem um zero real positivo. O zero, quando à direita no eixo real, adiciona mais atraso dinâmico ao sistema, além do já existente pelos 3 polos ($s = -1$). Em malha fechada gera um caminho de ganho positivo para um dos polos até o lado instável do eixo real. Mesmo com os devidos cuidados para que o sistema seja estável, a dinâmica da planta 3 realimentada possui uma resposta inversa inicialmente que aumenta o erro à entrada, exigindo mais esforço do controlador. Esse tipo de resposta indesejada, devida ao zero real positivo, pode ser conferida na Figura 25:

Figura 25 – Resposta ao degrau unitário da planta 3 compensada com PID em malha fechada



Fonte: Produção do próprio autor.

Na Figura 25 é apresentada a resposta ao degrau da planta 3 compensada por um PID com ganhos *Kp*, *Kd* e *Ki* iguais a 0.19, 0.11 e 0.07, respectivamente. Esse controlador foi o resultado do uso do algoritmo de evolução diferencial para a sintonia desses três ganhos na pesquisa de Saad, Jamaluddin e Darus (2012), bem como o controlador da planta 3. Como pode ser observado, mesmo com um sobressinal de 7.01% e estabilidade relativa adequada, a planta 3 apresenta uma oscilação negativa, retrocesso, maior que o próprio degrau de entrada do sistema. Isso pode causar desgaste na planta e em outras áreas do sistema completo, além de gastos desnecessários de energia.

As plantas 2 e 3 foram compensadas com controladores obtidos a partir de algoritmos evolutivos (SAAD; JAMALUDDIN; DARUS, 2012), e a primeira através do projeto de sintonia via lugar geométrico das raízes. Os controladores referentes as plantas 1, 2 e 3, respectivamente, podem ser conferidos nas Equações 37, 38 e 39:

$$C_1(s) = 0.95 + 1.08s + \frac{0.07}{s} \tag{37}$$

$$C_2(s) = 0.68 + 1.34s + \frac{0.07}{s} \tag{38}$$

$$C_3(s) = 0.19 + 0.11s + \frac{0.07}{s} \tag{39}$$

A três plantas e seus devidos controladores foram testados pelo AEFuzzy. Os parâmetros de configuração dos algoritmos ED e AG e do sistema *Fuzzy* pela interface podem ser conferidos nas Tabelas 1 e 2 que seguem:

Tabela 1 – Configuração dos parâmetros dos algoritmos ED e AG

ED	AG
Tamanho população = 100	Tamanho população = 100
Taxa de cruzamento = 0.9	Taxa de cruzamento = 0.9
Taxa de mutação = 0.6	Taxa de mutação = 0.02
-	Taxa de exclusão = 0.05
Número de gerações = 100	Número de gerações = 100
Elitista	Elitista

Fonte: Produção do próprio autor.

Tabela 2 – Configuração dos parâmetros *Fuzzy*

	Sistema de Ordem Elevada		Sistema com Atraso		Sistema de Fase Não-mínima	
Limites	Fixos	Variáveis	Fixos	Variáveis	Fixos	Variáveis
Nº conjuntos	5	5	5	5	5	5
Erro mín	-100	-150	-100	-150	-100	-150
Erro máx	100	150	100	150	100	150
dErro mín	-200	-250	-200	-250	-200	-250
dErro máx	200	250	200	250	200	250
Saída mín	-300	-400	-300	-400	-300	-400
Saída máx	300	400	300	400	300	400
GE mín	0	0	0	0	0	0
GE máx	12	12	12	12	12	12

Fonte: Produção do próprio autor.

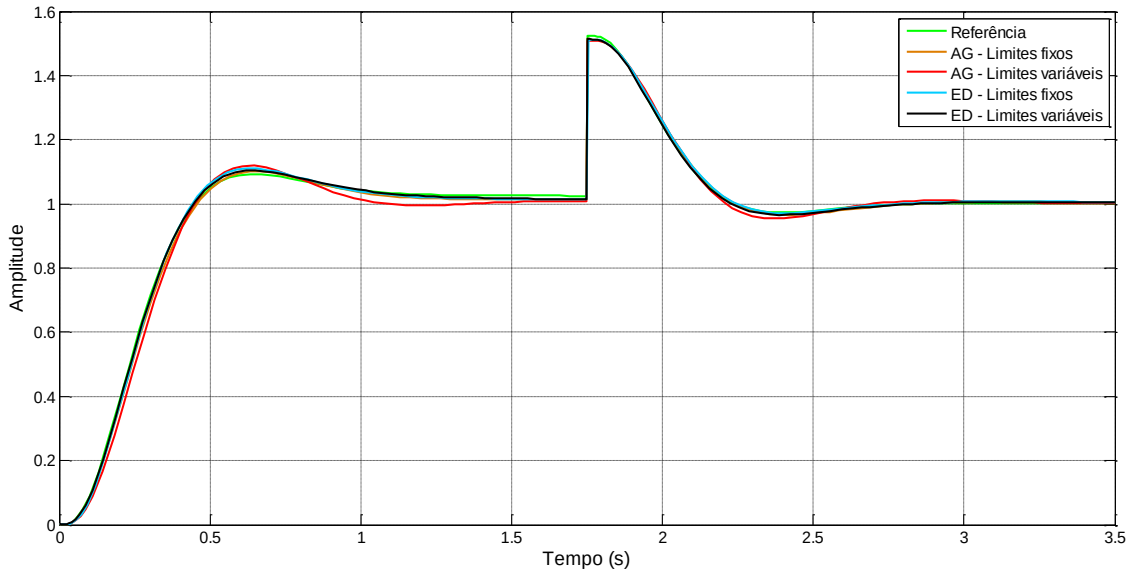
Os testes ocorreram tanto para o caso de limites fixos quanto para o caso de limites variando, como pode-se conferir na Tabela 2. Além disso, o ganho GE foi configurado para fazer parte do conjunto de cromossomos, ou seja, variar de 0 a 12 (valores próximos de 0, GE nulo zera GIE e GCE). Os resultados podem ser conferidos nas Figuras 26 a 37 e nas Tabelas 3 a 8, nas seções deste capítulo, bem como os sistemas e seus compensadores, que são retomados nas Equações 40 a 42. As curvas denominadas de Referência representam a saída do sistema compensado pelo controlador clássico de cada planta (função *fitness*), sendo a entrada do

sistema um degrau unitário, e tr o tempo de subida, tp o tempo de pico, td o tempo de atraso, ts o tempo de estabelecimento, MP o valor máximo percentual do sobressinal, t simulação o tempo total de simulação, e Undershoot o valor mínimo percentual (máximo módulo) da oscilação negativa inicial.

4.1 Sistema de ordem elevada

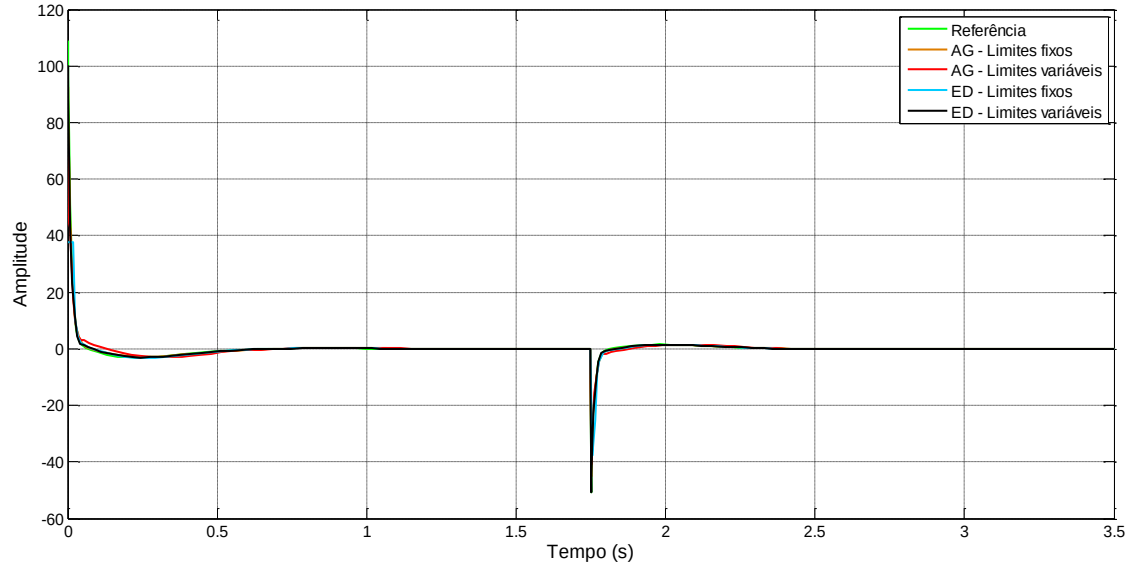
$$G_1(s) = \frac{1500}{s^4 + 40.67s^3 + 401.67s^2 + 250s}$$
$$C_1(s) = 0.95 + 1.08s + \frac{0.07}{s} \quad (40)$$

Figura 26 – Resposta ao degrau unitário dos melhores indivíduos encontrados para planta de ordem elevada



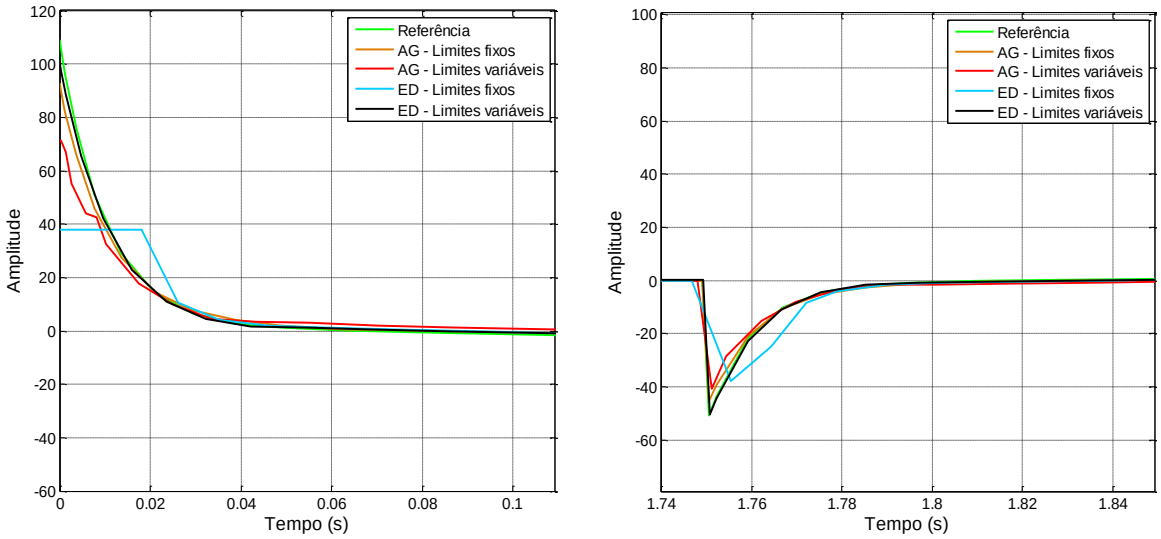
Fonte: Produção do próprio autor.

Figura 27 – Esforço de controle dos melhores indivíduos encontrados para planta de ordem elevada



Fonte: Produção do próprio autor.

Figura 28 – Detalhes do esforço de controle dos melhores indivíduos encontrados para planta de ordem elevada

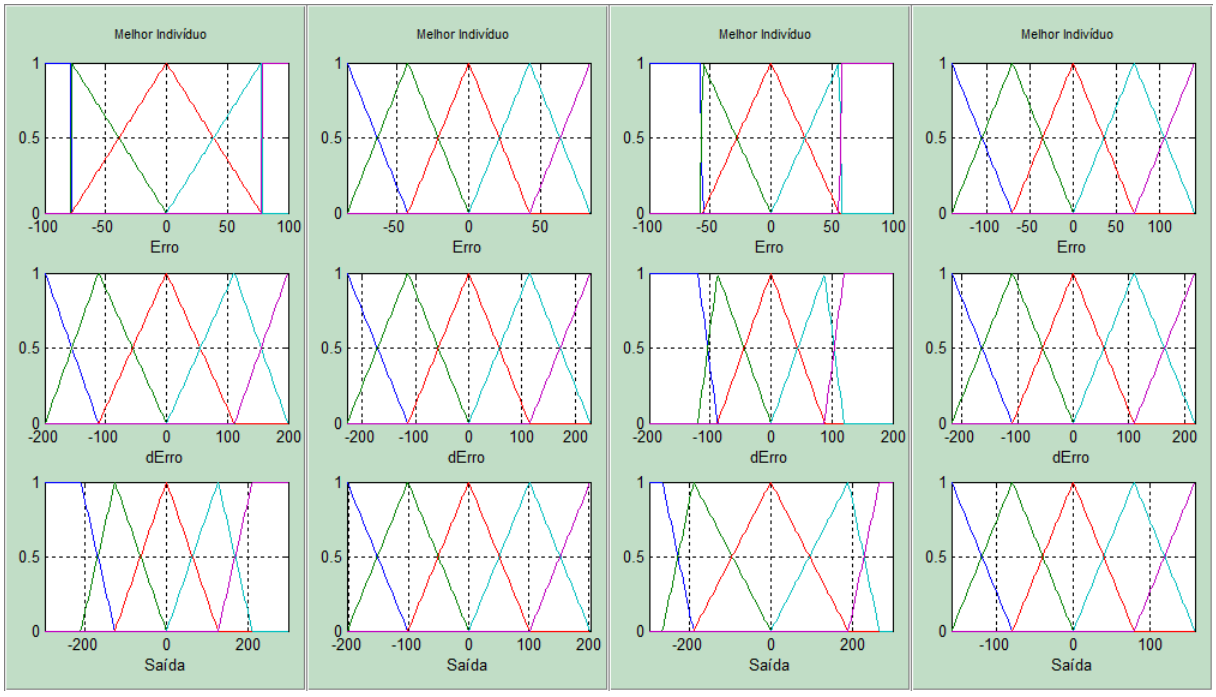


(a) (b)

Fonte: Produção do próprio autor.

Legenda: (a) Figura 27 ampliada no transitório inicial para mostrar melhor os detalhes
(b) Figura 27 ampliada no transitório causado pela perturbação para mostrar melhor os detalhes

Figura 29 – Variáveis linguísticas dos melhores indivíduos encontrados para planta de ordem elevada



(a) (b) (c) (d)

Fonte: Produção do próprio autor.

Legenda: (a) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* fixos usando AG
(b) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* variáveis usando AG
(c) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* fixos usando ED
(d) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* variáveis usando ED

Tabela 3 – Resultados do regime transitório para o sistema de ordem elevada

	Ref.	AG		ED	
Limites	-	Fixos	Variáveis	Fixos	Variáveis
tr (s)	0.28	0.29	0.28	0.27	0.28
tp (s)	0.65	0.67	0.65	0.63	0.65
td (s)	0.23	0.25	0.26	0.24	0.23
ts (s)	1.35	1.16	0.97	1.24	1.35
MP (%)	9.30	10.15	11.87	10.99	10.38
t simulação (s)	3.5	3.5	3.5	3.5	3.5

Fonte: Produção do próprio autor.

Tabela 4 – Parâmetros *Fuzzy* do melhor indivíduo encontrado para o sistema de ordem elevada

	AG		ED	
Limites	Fixos	Variáveis	Fixos	Variáveis
GE	0.51	2.27	6.57	0.13
GCE	0.58	2.58	7.47	0.15
GIE	0.04	0.17	0.48	0.01
GU	1.86	0.42	0.14	7.31
Erro Limites	[-100 100]	[-84.65 84.65]	[-100 100]	[-140.70 140.70]
Erro Conjuntos	[77.85 77.85]	[42.33 84.65]	[56.13 57.75]	[70.35 140.70]
dErro Limites	[-200 200]	[-229.00 229.00]	[-200 200]	[-218.99 218.99]
dErro Conjuntos	[111.52 199.38]	[114.50 229.00]	[88.12 119.64]	[109.49 218.99]
Saída Limites	[-300 300]	[-203.36 203.36]	[-300 300]	[-158.33 158.33]
Saída Conjuntos	[127.30 211.21]	[101.68 203.36]	[190.57 267.45]	[79.17 158.33]

Fonte: Produção do próprio autor.

Pode-se conferir na Figura 26 e na Tabela 3, que todos os casos obtiveram êxito em copiar a dinâmica da planta 1. O teste para limites variáveis utilizando o algoritmo de evolução diferencial se igualou completamente a referência, salve o sobressinal (10.38%) que se diferiu em 11.6% do valor da referência (9.30%). O resultado mais distinto visualmente pela resposta ao degrau unitário, na Figura 26, foi conseguido para a condição de limites variáveis utilizando o algoritmo genético, onde apesar de mesmo tempo de subida (0.28 s) e mesmo tempo de pico (0.65 s), o sobressinal foi de 11.87% e ainda assim, conseguiu se estabilizar antes, com um tempo de estabilização de 0.97 s.

Na Figura 27 pode-se observar os pontos em que o esforço dos controladores é mais relevante. Isso se deve a diferença inicial alta entre a saída da planta e o sinal de entrada (degrau unitário). Na Figura 28 é possível ter uma visão melhor do sinal de saída do controlador. Nota-se que o caso em azul claro (ED para limites fixos) não acompanha os

outros controladores no sinal do esforço de controle transitório (Figura 28), mas mesmo assim, sua resposta ao degrau unitário (Figura 26) copia a referência tanto quanto os outros casos.

Pela Figura 29 e Tabela 4 é possível conferir a disposição dos conjuntos *Fuzzy* dentro do universo *Fuzzy* de cada teste realizado. O melhor indivíduo encontrado para limites fixos utilizando AG possui sobreposição de conjuntos *Fuzzy* na variável linguística Erro, onde um ponto de máxima e um de mínima pertinência do trapézio se localizam sobre o mesmo ponto, já que o ponto de máxima pertinência do triângulo adjacente também se localiza nessa mesma posição. Esta situação vai contra o que foi comentado na seção 3.2, onde a soma de todas as possibilidades devem totalizar sempre 100% (neste caso, chegaria a 200%). Além disso, este seria um ponto de inconsistência, já que um conjunto apresenta valores de 100% e 0% no mesmo ponto. Entretanto, o MATLAB® trata este caso, tanto para valores positivos de erro, quanto para negativos, como sendo 100% para o conjunto trapezoidal, e 0% para o triangular, não deixando ocorrer a inconsistência. Quando o erro diminui em módulo, mesmo que infinitesimalmente, a pertinência referente ao trapézio atinge valor nulo e a pertinência do triângulo adjacente valor de 1 (100%).

Verifica-se também, pelas Figuras 26 e 29 e pela Tabela 4, que a dinâmica transitória de cada indivíduo foi semelhante a da função *fitness*, apesar de apresentarem diferenças no posicionamento dos conjuntos *Fuzzy* e ganhos de cada solução encontrada pelos algoritmos. Além disso, a disposição dos conjuntos pode ser explicada pela relação dos limites do universo de cada variável linguística, dos seus respectivos ganhos (*GE*, *GCE*, *GIE* e *GU*) e da magnitude do erro, da sua derivada e da sua integral. É possível citar, como exemplo, os resultados para limites fixos e variáveis utilizando ED.

No caso de limites variáveis, pelos conjuntos estarem espalhados uniformemente pelo universo *Fuzzy* e pelos ganhos *GE* e *GCE* serem pequenos em relação aos limites do universo, os valores de erro e de sua derivada serão mapeados, mesmo no início onde o sinal derivativo é elevado, próximos do centro do universo (valores próximos de 0, longe dos extremos). Isso implica na geração de uma regra dominante durante quase todo período de simulação, ou seja, dentre os conjuntos que compõem a resposta para a desfuzificação, um se sobressairá em

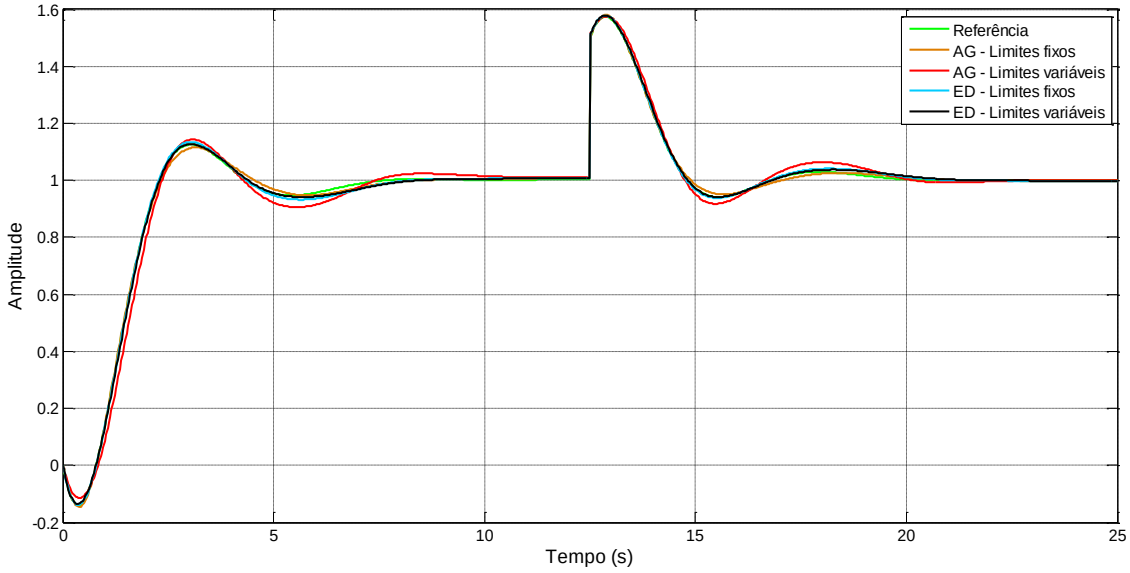
relação aos outros, apresentando uma pertinência máxima maior, trazendo o centro de gravidade para perto dele.

Já no caso de limites fixos, os ganhos são maiores, o que espalha por todo universo *Fuzzy* o sinal de derivada do erro. Contudo, pelo fato de o erro chegar no máximo a 1, o ganho *GE* mapeia o sinal de erro apenas até seu valor (*GE* = 6.57). Isso resulta em uma saída diferente do caso de limites variáveis, não obtendo nesta situação um conjunto dominante durante toda a simulação, assumindo vários formatos de saída, consequentemente variando bastante seu centro de gravidade.

4.2 Sistema com atraso

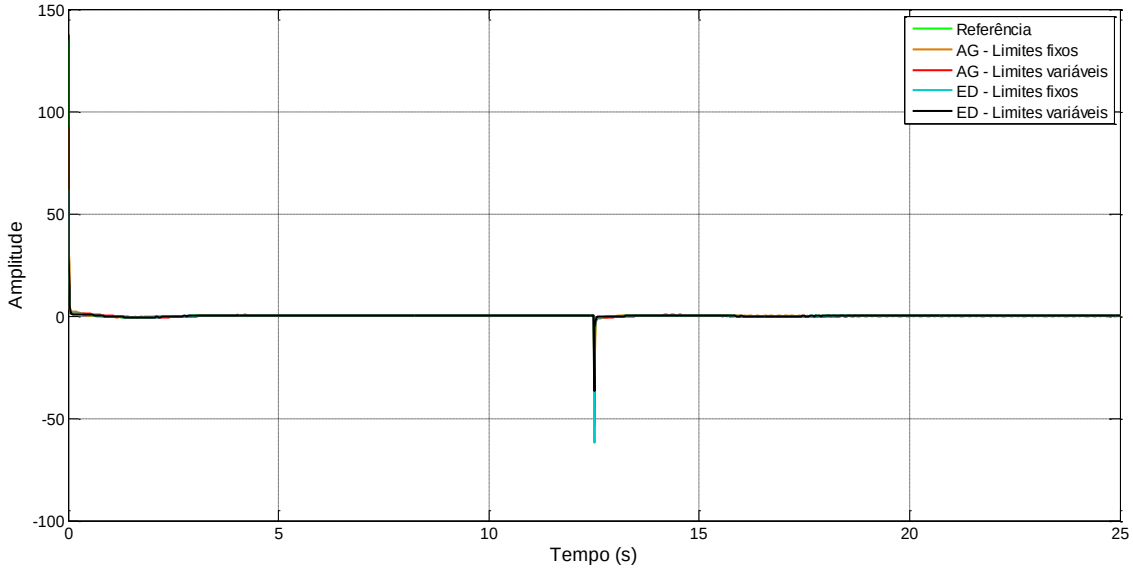
$$G_2(s) = \frac{10e^{-1.0s}}{(1+8s)(1+2s)}$$
$$C_2(s) = 0.68 + 1.34s + \frac{0.07}{s} \quad (41)$$

Figura 30 – Resposta ao degrau unitário dos melhores indivíduos encontrados para planta com atraso



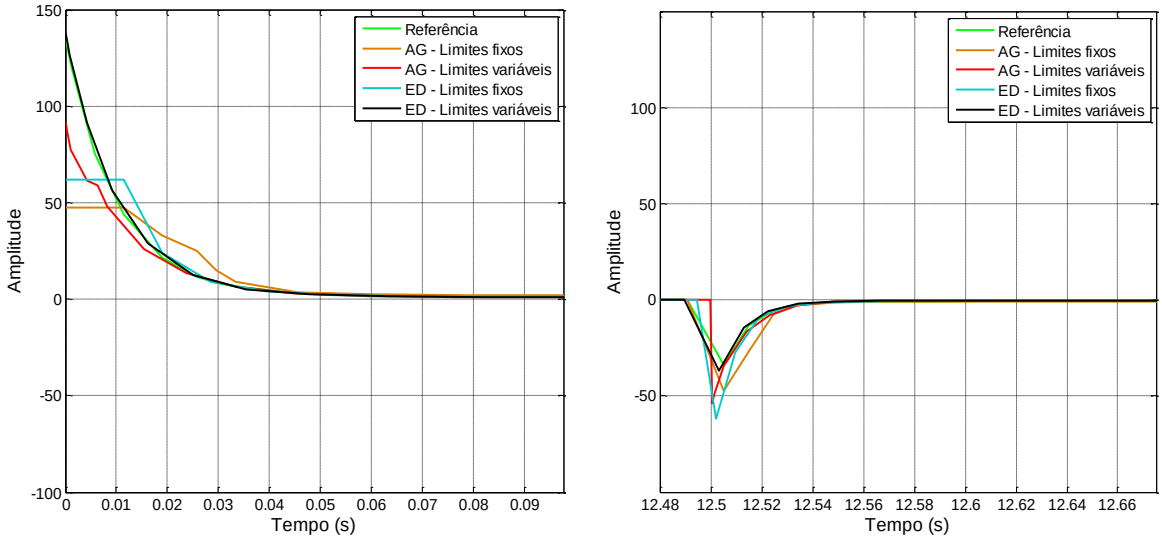
Fonte: Produção do próprio autor.

Figura 31 – Esforço de controle dos melhores indivíduos encontrados para planta com atraso



Fonte: Produção do próprio autor.

Figura 32 – Detalhes do esforço de controle dos melhores indivíduos encontrados para planta com atraso

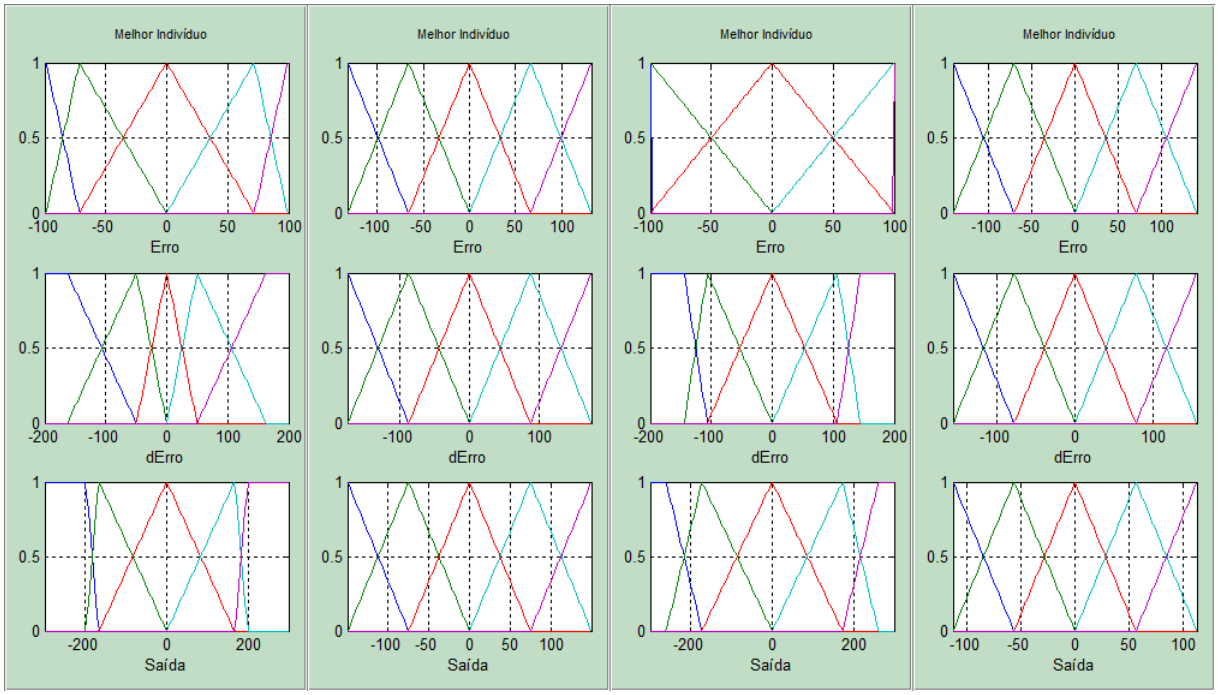


(a) (b)

Fonte: Produção do próprio autor.

Legenda: (a) Figura 31 ampliada no transitório inicial para mostrar melhor os detalhes
(b) Figura 31 ampliada no transitório causado pela perturbação para mostrar melhor os detalhes

Figura 33 – Variáveis linguísticas dos melhores indivíduos encontrados para planta com atraso



(a) (b) (c) (d)

Fonte: Produção do próprio autor.

Legenda: (a) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* fixos usando AG
(b) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* variáveis usando AG
(c) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* fixos usando ED
(d) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* variáveis usando ED

Tabela 5 – Resultados do regime transitório para o sistema com atraso

	Ref.	AG		ED	
Limites	-	Fixos	Variáveis	Fixos	Variáveis
tr (s)	1.14	1.14	1.14	1.11	1.12
tp (s)	3.03	3.16	3.03	3.02	3.05
td (s)	1.45	1.44	1.57	1.46	1.48
ts (s)	6.73	7.24	9.38	7.25	7.29
MP (%)	12.60	11.60	14.30	13.43	12.71
t simulação (s)	25	25	25	25	25

Fonte: Produção do próprio autor.

Tabela 6 – Parâmetros *Fuzzy* do melhor indivíduo encontrado para o sistema com atraso

	AG		ED	
Limites	Fixos	Variáveis	Fixos	Variáveis
GE	3.48	0.86	2.81	0.016
GCE	6.86	1.70	5.54	0.032
GIE	0.36	0.09	0.29	0.002
GU	0.19	0.79	0.24	41.99
Erro Limites	[-100 100]	[-132.44 132.44]	[-100 100]	[-140.62 140.62]
Erro Conjuntos	[71.61 99.20]	[66.22 132.44]	[100 100]	[70.31 140.62]
dErro Limites	[-200 200]	[-174.38 174.38]	[-200 200]	[-156.19 156.19]
dErro Conjuntos	[49.83 162.93]	[87.19 174.78]	[106.35 144.55]	[78.10 156.19]
Saída Limites	[-300 300]	[-148.77 148.77]	[-300 300]	[-112.73 112.73]
Saída Conjuntos	[166.70 199.40]	[74.38 148.77]	[172.92 260.96]	[56.36 112.73]

Fonte: Produção do próprio autor.

Na Figura 30 e na Tabela 5, os resultados levam a mesma conclusão que apresentaram os testes para a planta de ordem elevada. Como era esperado, todos as condições obtiveram uma dinâmica muito semelhante a da planta 2. Somente um caso (AG para limites variáveis) se diferiu, nos parâmetros do regime transitório apresentados na Tabela 5, mais que 8.5% dos valores retirados da resposta transitória de referência. Isso é refletido em sua resposta ao degrau unitário, na Figura 30, no qual seu sobressinal chega a 14.3% e tempo de estabilização aos 9,38 s. Entretanto, como já dito, apesar de ser 13.5% mais oscilatório que a referência, suas diferenças podem ser consideradas muito pequenas, portanto, aceitáveis.

Na Figura 32 pode-se observar detalhadamente o sinal de saída dos controladores. Nota-se que os casos em azul claro (ED para limites fixos) e em marrom (AG para limites fixos) foram os que mais se destoaram dos demais. O esforço de controle gerado por estes dois

compensadores se assemelha ao resultado de um saturador limitando os sinais em aproximadamente 62.5 e 39, respectivamente. Isso se deve ao efeito do Sistema de Inferência *Fuzzy* que, pelo particionamento de cada variável linguística, pelos dados iniciais da realimentação e pelas regras definidas como somador simples do controlador *Fuzzy*, resulta num mesmo conjunto de saída que, pela desfuzificação, corresponde inicialmente aos mesmos valores, os obtidos na simulação.

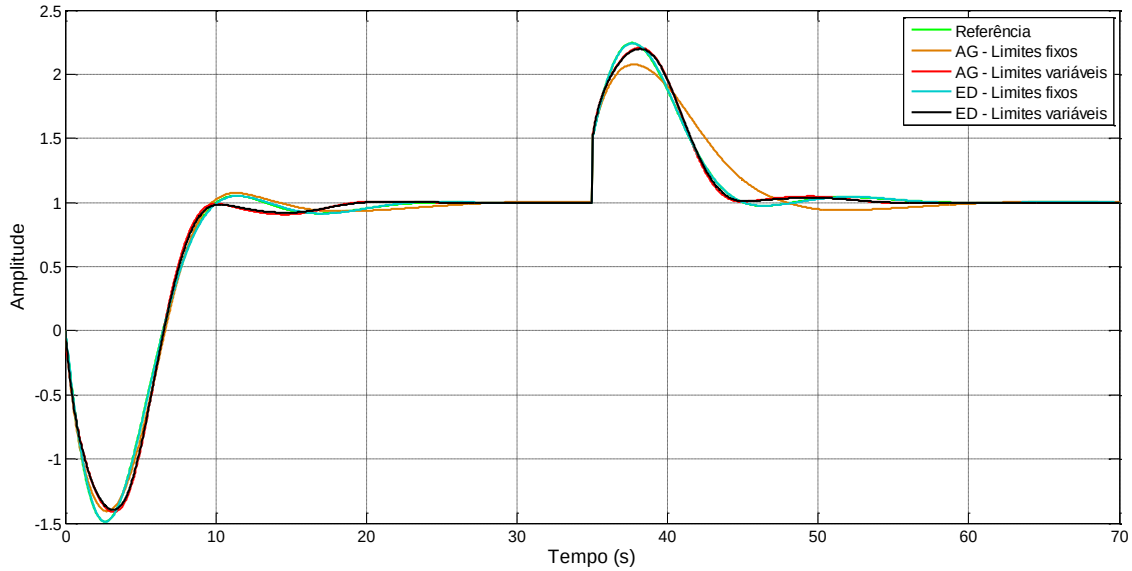
Pelas Figuras 30 e 32 e pela Tabela 5, é possível conferir que, assim como para os testes da planta de ordem elevada, o algoritmo de evolução diferencial (ED) para limites variáveis obteve um indivíduo que apresentou, em sua resposta ao degrau unitário, uma dinâmica transitória muito parecida com a da função *fitness*, validando a ferramenta de busca desenvolvida (AEFuzzy). Na Figura 33 e Tabela 6 é possível conferir a disposição dos conjuntos *Fuzzy* dentro do universo *Fuzzy* de cada teste realizado.

Assim como na seção 4.1, os resultados apresentados na Figura 33 e na Tabela 6 mostram que os ganhos obtidos para os casos de limites variáveis são maiores que para limites fixos e que, juntamente da disposição dos conjuntos *Fuzzy*, levam às mesmas conclusões apresentadas na seção 4.1.

4.3 Sistema de fase não-mínima

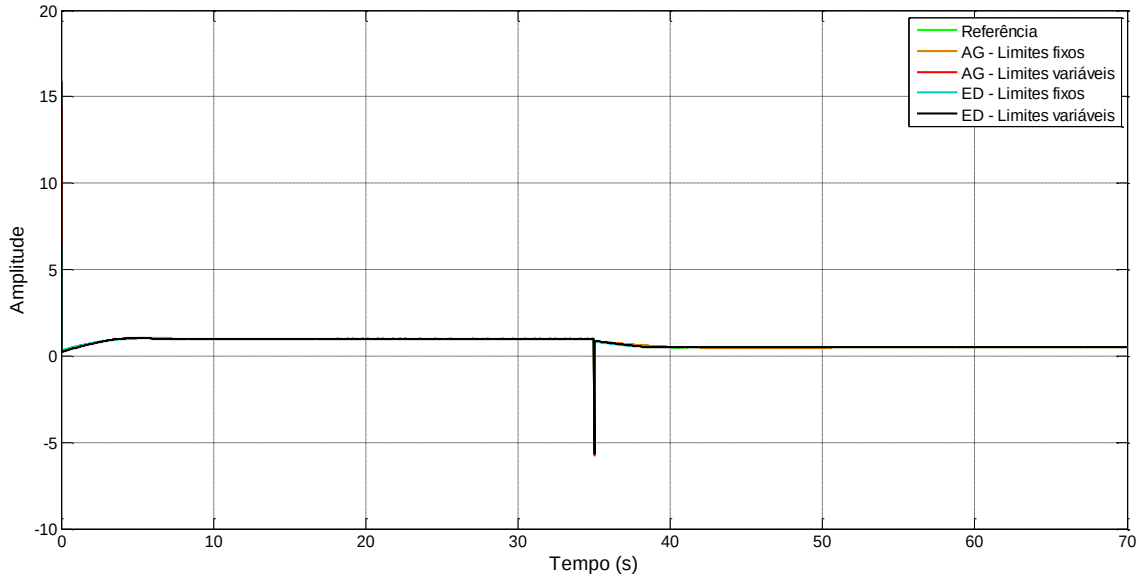
$$G_3(s) = \frac{1 - 10s}{(1+s)^3}$$
$$C_3(s) = 0.19 + 0.11s + \frac{0.07}{s} \quad (42)$$

Figura 34 – Resposta ao degrau unitário dos melhores indivíduos encontrados para planta de fase não-mínima



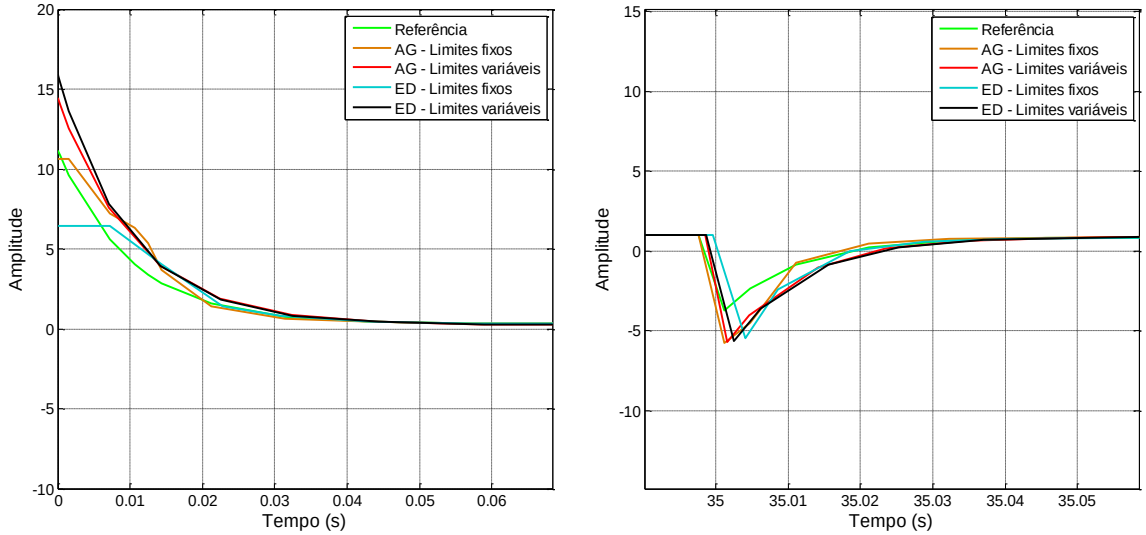
Fonte: Produção do próprio autor.

Figura 35 – Esforço de controle dos melhores indivíduos encontrados para planta de fase não-mínima



Fonte: Produção do próprio autor.

Figura 36 – Detalhes do esforço de controle dos melhores indivíduos encontrados para planta de fase não-mínima

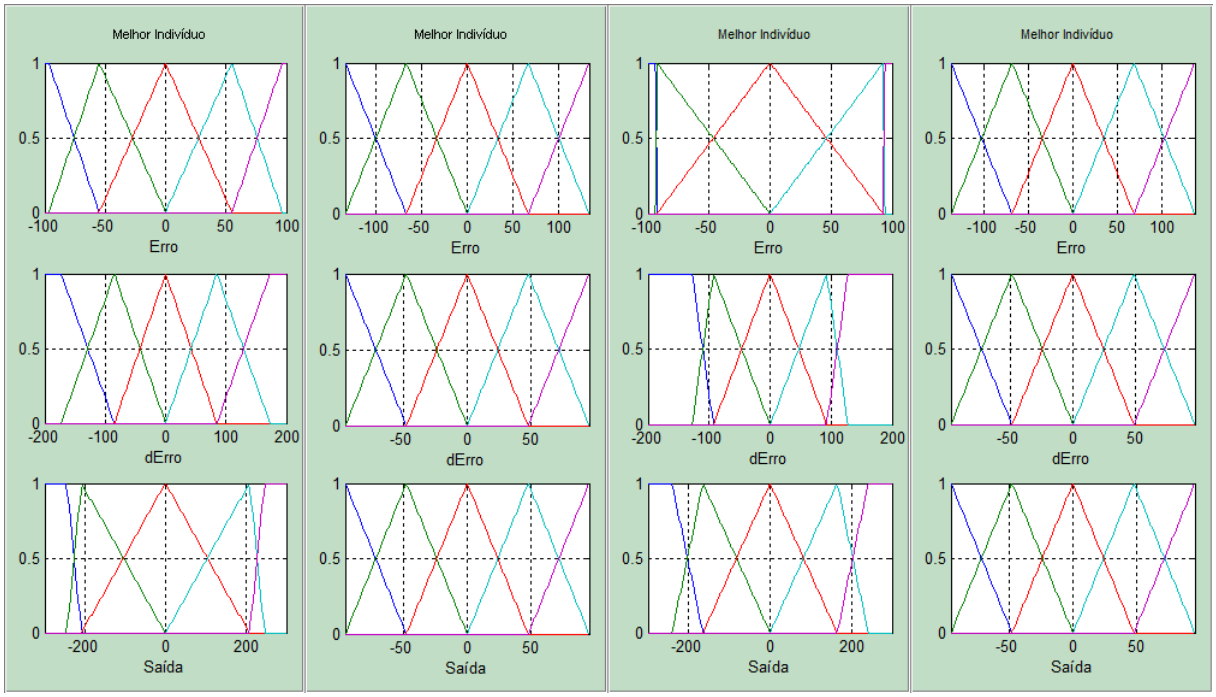


(a) (b)

Fonte: Produção do próprio autor.

Legenda: (a) Figura 35 ampliada no transitório inicial para mostrar melhor os detalhes
(b) Figura 35 ampliada no transitório causado pela perturbação para mostrar melhor os detalhes

Figura 37 – Variáveis linguísticas dos melhores indivíduos encontrados para planta de fase não-mínima



(a) (b) (c) (d)

Fonte: Produção do próprio autor.

Legenda: (a) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* fixos usando AG
(b) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* variáveis usando AG
(c) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* fixos usando ED
(d) Variáveis linguísticas do melhor indivíduo encontrado para limites *Fuzzy* variáveis usando ED

Tabela 7 – Resultados do regime transitório para o sistema de fase não-mínima

	Ref.	AG		ED	
Limites	-	Fixos	Variáveis	Fixos	Variáveis
tr (s)	2.55	2.24	2.12	2.58	2.24
tp (s)	11.3	11.33	20.65	11.3	21.29
td (s)	7.70	7.70	7.47	7.72	7.55
ts (s)	19.94	24.61	17.19	19.84	18.21
tus (s)	2.64	2.79	3.19	2.66	3.19
MP (%)	5.20	7.46	0.66	5.12	0.44
Undershoot (%)	148.80	140.70	141.30	149.00	139.50
t simulação (s)	70	70	70	70	70

Fonte: Produção do próprio autor.

Tabela 8 – Parâmetros *Fuzzy* do melhor indivíduo encontrado para o sistema de fase não-mínima

	AG		ED	
Limites	Fixos	Variáveis	Fixos	Variáveis
GE	4.72	0.094	7.34	0.0001
GCE	2.73	0.055	4.25	0.00005
GIE	1.74	0.035	2.70	0.00003
GU	0.04	2.013	0.026	2312.94
Erro Limites	[-100 100]	[-133.97 133.97]	[-100 100]	[-136.55 136.55]
Erro Conjuntos	[54.99 96.62]	[66.98 133.97]	[92.64 93.42]	[68.27 136.55]
dErro Limites	[-200 200]	[-95.09 95.09]	[-200 200]	[-98.46 98.46]
dErro Conjuntos	[84.83 172.82]	[47.54 95.09]	[92.95 127.71]	[49.23 98.46]
Saída Limites	[-300 300]	[-95.38 95.38]	[-300 300]	[-96.18 96.18]
Saída Conjuntos	[208.54 245.99]	[47.69 95.38]	[164.86 241.27]	[48.09 96.18]

Fonte: Produção do próprio autor.

Verifica-se, através da Figura 34 e da Tabela 7, que para a condição da planta de fase não-mínima, o melhor indivíduo, ou seja, cuja dinâmica seja mais semelhante à da referência, não foi conseguido através de limites variáveis, como obervado para os dois casos anteriores, do sistema de ordem elevada e da planta com atraso. Contudo, esse indivíduo também foi obtido a partir do algoritmo de evolução diferencial (ED).

A melhor solução encontrada apresentou uma dinâmica muito parecida à da planta 3 compensada pelo PID clássico (referência) e pode ser observada pelas curvas azul claro (ED para limites fixos) nas Figuras 34, 35 e 36. Apesar disso, pode-se verificar conforme os resultados apresentados pela Figura 34 e Tabela 7, que outros resultados que não tenham

dinâmica semelhante à curva de referência podem também ser aproveitados. Soluções que apresentem respostas menos oscilatórias e com tempo de resposta inferior ao da função *fitness*, que pela lógica dos algoritmos possuiriam menor aptidão, também podem ser exploradas, afinal, é o que se busca em qualquer sistema controlado. Isso fica evidente nas soluções geradas a partir dos dois algoritmos para limites variáveis. Estas soluções ainda, juntamente com a de limites fixos utilizando algoritmo genético, tiveram um *undershoot* (oscilação de retrocesso inicial) menor que a da referência.

Na Figura 36 pode-se observar em detalhes o sinal de saída dos controladores. Como pode-se notar, a curva azul claro (ED para limites fixos) não chega ao pico inicial da referência (curva verde). Efeito semelhante ao de adicionar um saturador na saída do controlador da curva azul clara, não o permitindo ultrapassar certo valor (6.25 e -6.25). Verifica-se que em uma situação dessas, mesmo com a utilização de um saturador, ou seja, mesmo o controlador não podendo atingir valores tão grandes quanto os da referência, pode-se encontrar uma solução que apresente resposta transitória e de regime semelhantes aos desejados.

Mesmo que para o caso da perturbação o valor absoluto máximo do esforço de controle da solução encontrada pelo algoritmo ED para limites fixos tenha sido superior ao da referência, para efeitos de estudo da suposição de utilização de um saturador, seu valor não ultrapassaria o limite de 6.25, não sendo, portanto, penalizado em sua resposta transitória devida a perturbação.

Pela Figura 37 e Tabela 8 é possível conferir a disposição dos conjuntos *Fuzzy* dentro do universo *Fuzzy* de cada teste realizado para o sistema de fase não-mínima. Nota-se que para o caso de limites variáveis utilizando ED os ganhos *GE*, *GCE* e *GIE* apresentaram valores muito pequenos (valores relevantes na quarta ou quinta casa decimal). Isso afeta o centro de gravidade do conjunto de saída, afinal, mesmo valores elevados da derivada do erro serão atenuados e mapeados em posições próximas de 0 (centro do universo *Fuzzy* da variável linguística). Com o erro e a sua derivada mapeados próximos de 0, o conjunto de saída praticamente será formado pelo conjunto *Fuzzy* ZE (zero), sendo este, portanto um conjunto dominante.

5 CONCLUSÕES E TRABALHOS FUTUROS

Neste projeto de graduação foi apresentada uma contribuição para o desenvolvimento de sistemas *Fuzzy* na área de controle e modelagem de plantas. Foi proposta uma nova metodologia de ajuste de controladores *Fuzzy* utilizando algoritmos evolutivos (Algoritmo Genético ou Evolução Diferencial).

O algoritmo genético e de evolução diferencial são comumente utilizados no ajuste de ganhos de um controlador puramente linear. Porém, conseguiu-se comprovar, através dos resultados obtidos com o AEFuzzy, que essas aplicações podem ser eficientes métodos de otimização para sistemas *Fuzzy*, tanto no aspecto de síntese de termos, quanto na síntese de ganhos.

Mostrou-se também que é comum encontrar trabalhos que utilizam a técnica *Fuzzy* por ser possível simular o conhecimento de um especialista em uma base de regras e alterar estas sempre que desejado, com o objetivo de ajustar parâmetros de controladores lineares para diferentes situações que possam vir a acontecer, como, por exemplo, degraus de amplitudes variadas na entrada do sistema.

Apesar dessa vantagem do emprego do *Fuzzy* em sistemas de controle, é necessário ter cuidado com as alterações da base de regras. A alteração de apenas uma regra não significa que esta não interferirá no resto da base de regras. O que ocorre é justamente o contrário. Como o *Fuzzy* interpola as regras, a alteração feita não ficará contida apenas para a condição modificada, ela alterará os resultados formados da junção desta com as regras adjacentes, podendo acarretar em respostas inesperadas e até mesmo indesejadas do sistema.

A ferramenta AEFuzzy desenvolvida realiza a síntese de um controlador PID *Fuzzy*. Entretanto, a base de regras é fixada para funcionar como um somador simples e, então, a ferramenta de síntese proposta realiza o ajuste fino de termos e ganhos sempre que preciso sem a necessidade de maiores esforços do pesquisador. As regras, portanto, possuem comportamento previsto e, como foi possível observar pelos resultados, o AEFuzzy conseguiu ajustar as variáveis linguísticas *Fuzzy* tal que o FPD+I de Jantzen obteve mesma dinâmica transitória que o sistema compensado por um PID clássico utilizado como referência.

O algoritmo de evolução diferencial (ED) gerou sempre os melhores resultados. Em 12 simulações foi possível ver o quão eficiente foram os dois algoritmos. É importante frizar que a eficiência em gerar um bom indivíduo depende, para os dois casos, da população inicial, executando mais vezes o programa, resultados melhores poderiam ser obtidos.

Pode-se afirmar isto, pois como as funções de referência são as próprias plantas já compensadas, a resposta a ser copiada existe de fato para o sistema, portanto, o *Fuzzy* só deve ser sintonizado de forma a apresentar a mesma dinâmica. Mesmo assim, é importante apontar, que a melhor forma de utilização de um sistema *Fuzzy* é como um interpolador, usando conhecimento especialista para gerar e sintonizar o sistema *Fuzzy*. Desta maneira, poder-se-ia utilizar a verdadeira potencialidade da lógica *Fuzzy*: sua propriedade de interpolar dados e regras, não sendo necessário o projeto de controladores para cada situação. Uma ferramenta como o AEFuzzy entraria para realizar a sintonia fina deste interpolador.

Foi observado também que alguns resultados obtidos, como, por exemplo, para a planta de fase não-mínima, os algoritmos retornaram como solução indivíduos nem tão parecidos com a referência, mas menos oscilatórios (menor sobressinal e menor *undershoot*) e com tempo de resposta (tempo de subida, de atraso e de estabilização) inferior às soluções alcançadas na literatura apresentada. Logo, mesmo buscando obter a função *fitness*, resultados de melhor desempenho também podem ser encontrados através desta ferramenta.

Por fim, apesar das plantas utilizadas para a realização das simulações não serem necessariamente modelos existentes, suas características englobam situações recorrentes em plantas reais, resultando em conclusões suficientes para qualquer caso, ou seja, funcionando para esses cenários, espera-se que funcione para qualquer sistema que apresente tais características, real ou não.

Para trabalhos futuros, seria interessante analisar formas de relacionar mais os parâmetros dos limites do modelo *Fuzzy*, para que o sistema de busca fique mais otimizado (por exemplo, ligando o valor de GE com os limites do universo *Fuzzy* do Erro). Seria possível, também, explorar mais o conjunto de regras de definição do PD *Fuzzy* (FPD), procurando alternativas para somas *Fuzzy* que ponderem de maneira diferenciada o valor do erro e a derivada do erro, ou até mesmo o que cabe à componente integral no que se refere à elaboração do conjunto.

Entretanto, talvez não seja necessária uma síntese diferenciada de regras, podendo ser sintonizados os outros dois graus de liberdade do controlador *Fuzzy*: os ganhos de escalonamento e os próprios parâmetros dos conjuntos *Fuzzy*.

Os algoritmos implementados são eficientes e seu estudo é de suma importância, haja vista sua vasta utilização. Portanto, outra contribuição possível para as próximas pesquisas seria de adicionar outros algoritmos, como nuvem de partículas e colônia de formigas, e outros tipos de fitness, como a adição do esforço de controle em seu cálculo, visando encontrar meios melhores de sintonia e deixar mais conhecimento através de documentação e implementações para os trabalhos posteriores.

Além disso, seria interessante realizar testes com plantas com características diferentes das já utilizadas para a validação do *software* AEFuzzy, como sistemas não lineares e instáveis em malha aberta (exemplo: pêndulo invertido e sistema de levitação magnética). Além disso, talvez fosse melhor copiar a dinâmica da planta compensada pelo controlador clássico através da utilização de um sinal binário pseudo aleatório (PRBS) no lugar do degrau na entrada do sistema, buscando uma melhor excitação na entrada.

Uma continuação direta a este trabalho seria de implementar um controlador encontrado pelo AEFuzzy em uma planta real. Entretanto, a ideia de utilizar um Sistema de Inferência *Fuzzy* para a resolução de problemas de controle não precisa necessariamente de ser no núcleo do controlador. Não foi por acaso que o sistema *Fuzzy* foi empregado inúmeras vezes como um interpolador na literatura. É importante relembrar que mesmo que um controlador de núcleo *Fuzzy* seja utilizado, sua configuração é feita para um caso de entrada. Caso a entrada seja diferente, novos parâmetros devem ser calculados.

Isso traz a necessidade de um processamento a mais no controlador, algo que analise a entrada, o erro, ou até sua derivada, para escolher tais parâmetros do PID utilizado. Portanto, uma boa contribuição de trabalho futuro seria utilizar a ideia da ferramenta AEFuzzy para sintonizar um interpolador *Fuzzy* usando algoritmos evolutivos. Essa proposta poderia ser utilizada em robôs móveis multiarticulados, mesclando esta pesquisa com outras na mesma área de estudo.

REFERÊNCIAS

ABEYWARDENA, D. M. W. et al. A velocity feedback Fuzzy Logic controller for stable hovering of a quad rotor UAV. In: INTERNATIONAL CONFERENCE ON INDUSTRIAL AND INFORMATION SYSTEMS, ICIIS, 4., 2009, Peradeniya, Sri Lanka. **IEEE Xplore CD.** Peradeniya, Sri Lanka: [s.n.], 2009. p. 558-562. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5429800&isnumber=5429790>. Acesso em 06 mar. 2014.

BERTOLANI, D. N. **Sintonia do controle de configuração de robôs móveis multiarticulados via algoritmo genético.** 2013. Dissertação (Mestrado em Engenharia Elétrica) — Programa de Pós Graduação em Engenharia Elétrica, Universidade Federal do Espírito Santo, Vitória, 2013.

CARLI, A. D.; LIGUORI, P.; MARRONI, A. A Fuzzy-PI control strategy. **Control Engineering Practice**, v. 2, n. 1, p. 147-153, feb. 1994.

COELHO, L. dos S.; MARIANI, V. C. Evolução diferencial híbrida com programação quadrática aplicada ao problema de despacho econômico de energia elétrica. **Revista Controle & Automação**, v. 17, n. 4, p. 409-423, out./dez. 2006. Disponível em: <http://www.scielo.br/pdf/ca/v17n4/a03v17n4>. Acesso em: 05 jul. 2014.

COSTA, W. T. da. **Modelagem, estimação de parâmetros e métodos MPPT para módulos fotovoltaicos.** 2010. Tese (Doutorado em Engenharia Elétrica) — Programa de Pós Graduação em Engenharia Elétrica, Universidade Federal do Espírito Santo, Vitória, 2010.

FÉO, A. E. **Implementação de um estabilizador de sistemas de potência utilizando compensador Fuzzy com características Lead/Lag.** 2004. Dissertação (Mestrado em Engenharia Elétrica) — Programa de Pós Graduação em Engenharia Elétrica, Universidade Federal de Itajubá, Itajubá, MG, 2004. Disponível em: <http://juno.unifei.edu.br/bim/0031002.pdf>. Acesso em: 05 mar. 2014.

FERNANDES, M. R. **Algoritmo Genético.** Vitória: 2006. Disponível em: <http://moodle.ele.ufes.br/file.php/274/AlgoritmoGeneticoMRF.pdf>. Acesso em: 05 mar. 2014.

FERREIRA, E. P. **Controle Inteligente & Fundamentos e Aplicações da Lógica Fuzzy ao Controle de Sistemas.** Vitória: 2009. Disponível em: <http://moodle.ele.ufes.br/file.php/274/Controle_Inteligente_Logica_Fuzzy.pdf>. Acesso em: 04 mar. 2014.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning.** 1st ed. Reading, MA: Addison-Wesley, 1989.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems.** 1st ed. Cambridge, MA: MIT Press, 1975.

JANTZEN, J. **Design of Fuzzy controllers.** 1998a. Technical Report — Department of Automation, Technical University of Denmark, Kongens Lyngby, 1998a. Disponível em: <http://moodle.ele.ufes.br/file.php/274/JJantsen_design_PIDfuzzy.pdf>. Acesso em: 05 mar. 2014.

_____. **Tuning of Fuzzy PID controllers.** 1998b. Technical Report — Department of Automation, Technical University of Denmark, Kongens Lyngby, 1998b. Disponível em: <http://moodle.ele.ufes.br/file.php/274/JJantsen_tunning_PIDfuzzy.pdf>. Acesso em: 05 mar. 2014.

KUO, B. C.; GOLNARAGHI, F. **Automatic Control Systems.** 9th ed. New Jersey: John Wiley & Sons, Inc. 2009.

LIN, C.; LEE, C. S. G. **Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems.** New Jersey: Prentice-Hall, Inc. 1996.

MAMDANI, E. H. Application of Fuzzy Logic to approximate reasoning using linguistic synthesis. **IEEE Transactions on Computers**, v. 26, n. 12, p. 1182-1191, dec. 1977. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1674779>. Acesso em: 05 mar. 2014.

PANDOLFI, F. **Modelagem e controle de robôs móveis multiarticulados no espaço de configurações: soluções não- lineares e Fuzzy**. 2012. Dissertação (Mestrado em Engenharia Elétrica) — Programa de Pós Graduação em Engenharia Elétrica, Universidade Federal do Espírito Santo, Vitória, 2012.

SAAD, M. S; JAMALUDDIN, H.; DARUS, I. Z. M. PID controller tuning using evolutionary algorithms. **WSEAS Transactions on Systems and Control**, v. 7, n. 4, p. 139-149, oct. 2012. Disponível em: <http://www.wseas.org/multimedia/journals/control/2012/54-890.pdf>. Acesso em: 13 jul. 2014.

SHARKAWY, A. B. Genetic Fuzzy self-tuning PID controllers for Antilock Braking Systems. **Engineering Applications of Artificial Intelligence**, v. 23, n. 7, p. 1041-1052, oct. 2010. Disponível em: <http://www.sciencedirect.com/science/article/pii/S095219761000134X#>. Acesso em: 05 mar. 2014.

SIVANANDAM, S. N.; SUMATHI, S.; DEEPA, S. N. **Introduction to Fuzzy Logic using MATLAB.** 1st ed. New Jersey: Springer-Verlag New York, Inc. 2007. Disponível em: <http://books.google.com.br/books?id=a_J2P3FT9sgC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 06 mar. 2014.

STORN, R.; PRICE, K. **Differential evolution: a simple and efficient adaptive scheme for global optimization over continuous spaces.** 1995. Technical Report — International Computer Science Institute, Berkeley, 1995. Disponível em: <http://www1.icsi.berkeley.edu/ftp/pub/techreports/1995/tr-95-012.pdf>. Acesso em: 05 jul. 2014.

TAKAGI, T.; SUGENO, M. Fuzzy identification of systems and its applications to modeling and control. **IEEE Transactions on Systems, Man and Cybernetics**, v.15, n. 1, p. 116-132, jan./feb. 1985. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6313399>. Acesso em: 06 mar. 2014.

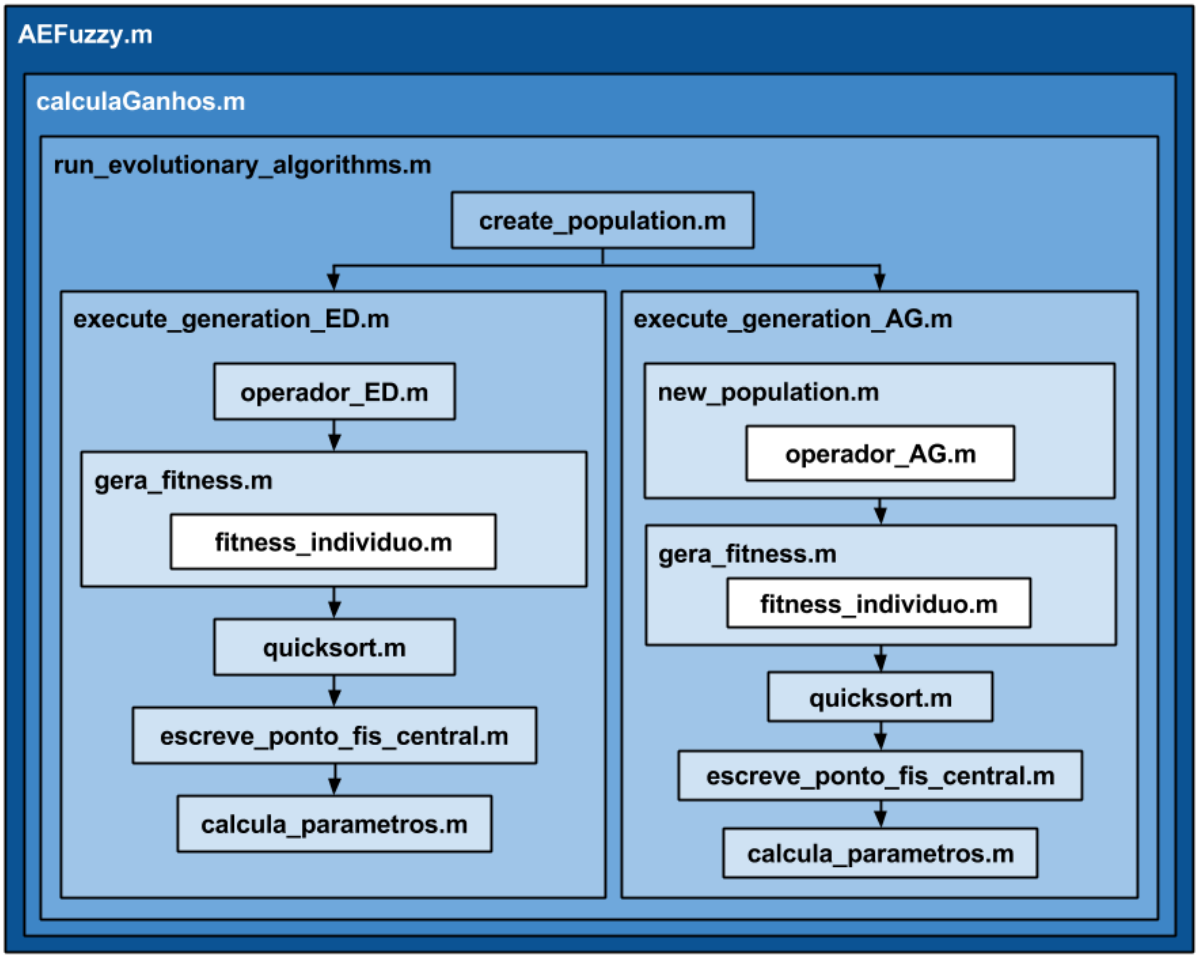
WANG, L.-X.; MENDEL, J. M. Generating Fuzzy rules by learning from examples. **IEEE Transactions on Systems, Man and Cybernetics**, v. 22, n. 6, p. 1414-1427, nov./dec. 1992. Disponível em: <http://moodle.ele.ufes.br/file.php/274/Wang_e_Mendel_artigo_geracao_de_regras.pdf>. Acesso em: 05 mar. 2014.

ZADEH, L. A. Fuzzy Sets*. **Information and Control**, v. 8, n. 3, p. 338-353, jun. 1965. Disponível em: <http://www.sciencedirect.com/science/article/pii/S001999586590241X#>. Acesso em: 06 mar. 2014.

APÊNDICE A – CÓDIGO FONTE DO AEFUZZY

A Figura 38 apresenta o fluxograma de funções do AEFuzzy. Nesta, um bloco se localizar no interior de outro significa que a função do bloco de fora chama a função do bloco de dentro. Além disso, um bloco estar ligado a outro através de uma seta representa a sequência em que as funções, dentro de outra, são chamadas. Por exemplo: as funções *create_population.m*, *execute_generation_ED.m* e *execute_generation_AG.m* são todas chamadas pela função *run_evolutionary_algorithms.m*, porém, a função *create_population.m* é executada primeiro, e depois, dependendo de qual algoritmo foi escolhido pelo usuário, a função *execute_generation_ED.m* ou *execute_generation_AG.m* é executada.

Figura 38 – Fluxograma de funções do programa AEFuzzy



Fonte: Produção do próprio autor.

Nas subseções a seguir, pode-se conferir os código de cada bloco mostrado na Figura 38. Nem todos se encontram em íntegra. Devido à extensão do código *AEFuzzy.m*, apenas a parte que

se refere a configuração inicial e ao funcionamento do algoritmo descrito no fluxograma da Figura 38 foi anexada às subseções. Partes ligadas à coleta de dados e funcionalidades (traçar gráficos, gerar relatório, salvar e carregar parâmetros) da interface, no código *AEFuzzy.m*, foram retiradas.

A.1 – AEFuzzy.m

```
function varargout = AEFuzzy(varargin)
% AEFuzzy
% Orientador: Prof. Dr. Ing. Edson Ferreira de Paula.
% Coorientador: Prof. MSc. Diego Nunes Bertolani.
% Autor: Leonardo de Assis Silva.
% Programa baseado no AGPID 2.0 feito por: Clebson Joel Mendes de Oliveira
% e no A4G feito por: Diego Nunes Bertolani.
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @AEFuzzy_OpeningFcn, ...
                   'gui_OutputFcn',  @AEFuzzy_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end

% --- Função de abertura. Executa antes de a interface ser visível.
function AEFuzzy_OpeningFcn(hObject,eventdata,handles,varargin)
% Inicialização do programa:
```

```
% ------- Escolhe o Algoritmo Genetico como Default ----------
handles.variaveis.aged = 1;
% ---------------------------------------------------------------------------
% Parâmetros:
% ------- Numerador e Denominador da FT de malha aberta ------------
% Carrega a função de transferência que está armazenada em dois arquivos
% e exporta a mesma para o workspace:
Num = importdata('num.txt');
set(handles.num,'String',sprintf('%.2f ',Num));
Den = importdata('den.txt');
set(handles.den,'String', sprintf('%.2f ',Den));
assignin('base', 'Num', Num);
assignin('base', 'Den', Den);
% ---------------------------------------------------------------------------

% ----- Carregando Variáveis ------------------------------------------
handles = carrega_var('var.txt',handles);
% ---------------------------------------------------------------------------

% ------- Kd Kd e Ki ----------------------------------------------------
set(handles.Kp,'String',handles.variaveis.Kp);
set(handles.Kd,'String',handles.variaveis.Kd);
set(handles.Ki,'String',handles.variaveis.Ki);
% Exporta os ganhos para o workspace:
assignin('base', 'Kp', handles.variaveis.Kp);
assignin('base', 'Kd', handles.variaveis.Kd);
assignin('base', 'Ki', handles.variaveis.Ki);
% ---------------------------------------------------------------------------

% ------- Crossover - Mutação [0;1] -------------------------------------
set(handles.cross,'String',handles.variaveis.tx_cross);
set(handles.mut,'String',handles.variaveis.tx_mut);
% ---------------------------------------------------------------------------
```

```
% ------- Exclusão [0;1] - Elitismo ----------------------------------------
set(handles.exc,'String',handles.variaveis.tx_exclui);
% Programa abre configurado como Elitista:
set(handles.elite,'value',1);
handles.variaveis.elite = 1;
% --------------------------------------------------------------------------

% ------- Número de Variáveis Linguísticas -----------------------------
set(handles.Ne,'String',handles.variaveis.Ne);
set(handles.Nde,'String',handles.variaveis.Nde);
set(handles.Nsa,'String',handles.variaveis.Nsa);
% --------------------------------------------------------------------------

% ------- Limites dos Parâmetros ---------------------------------------
set(handles.erromin,'String',handles.variaveis.erromin);
set(handles.erromax,'String',handles.variaveis.erromax);
set(handles.derromin,'String',handles.variaveis.derromin);
set(handles.derromax,'String',handles.variaveis.derromax);
set(handles.saidamin,'String',handles.variaveis.saidamin);
set(handles.saidamax,'String',handles.variaveis.saidamax);
% Programa abre configurado sem variação dos limites:
set(handles.limites,'value',0);
handles.variaveis.limites = 0;
% --------------------------------------------------------------------------

% ------- Geração - População ------------------------------------------
set(handles.ger,'String',handles.variaveis.max_gen);
set(handles.pop,'String',handles.variaveis.max_pop);
% --------------------------------------------------------------------------

% ------- Tempo de Simulação (seg.) ------------------------------------
set(handles.tsimu,'String',handles.variaveis.tsimu);
```

```
% Define tempo da simulação em segundos:
tempo_simulacao = handles.variaveis.tsimu;
assignin('base','tempo_simulacao',tempo_simulacao);
% --------------------------------------------------------------------------

% ------- GE (GEmax) ---------------------------------------------------
set(handles.gemax,'String',handles.variaveis.gemax);
% Programa inicia configurado com variação do ganho GE (sendo gemax o valor máximo):
set(handles.gevaria,'value',1);
handles.variaveis.gevaria = 1;
% --------------------------------------------------------------------------

% ------- Configurações Gerais -----------------------------------------
% Esconde o texto que informa se o programa está executando o algoritmo:
set(handles.wrkng,'Visible','off');
% Cria e exporta os Valores do Degrau:
degrauini = 0;
degraufim = 1;
assignin('base','degrauini',degrauini);
assignin('base','degraufim',degraufim);
% --------------------------------------------------------------------------

% Atualiza a variável handles ----------------------------------------------
guidata(hObject, handles);

% Campos de Resultados:
% ------- Inicia os campos de erro derro e saida -------------------------
set(handles.erro,'String','...');
set(handles.derro,'String','...');
set(handles.saida,'String','...');
set(handles.ge,'String','...');
set(handles.ge,'BackgroundColor',[0.87 0.92 0.98]);
set(handles.fitness,'String','...');
```

```
set(handles.fitness,'BackgroundColor',[0.87 0.92 0.98]);
% --------------------------------------------------------------------

% ------- Inicia as configurações dos gráficos ----------------------------
% Adiciona grade e títulos aos eixos das resposta ao degrau e fitness
hold(handles.axes1,'on');
hold(handles.axes2,'on');
grid(handles.axes1,'on');
grid(handles.axes2,'on');
title(handles.axes1,'Resposta ao Degrau do Melhor Indivíduo');
title(handles.axes2,'Evolução do Melhor Indivíduo');
xlabel(handles.axes1, 'Tempo (s)');
ylabel(handles.axes1, 'Resposta');
xlabel(handles.axes2, 'Gerações');
ylabel(handles.axes2, 'Fitness');
% Adiciona grade, títulos aos eixos e determina o alcance dos gráficos das variáveis
linguísticas
grid(handles.mferro,'on');
xlabel(handles.mferro, 'Erro');
axis(handles.mferro,[-handles.variaveis.erromax handles.variaveis.erromax 0 1]);
grid(handles.mfderro,'on');
xlabel(handles.mfderro, 'dErro');
axis(handles.mfderro,[-handles.variaveis.derromax handles.variaveis.derromax 0 1]);
grid(handles.mfsaida,'on');
xlabel(handles.mfsaida, 'Saída');
axis(handles.mfsaida,[-handles.variaveis.saidamax handles.variaveis.saidamax 0 1]);
% --------------------------------------------------------------------

% ------- Logo UFES ------------------------------------------------------
% Adiciona o logo da UFES no programa:
logoufes = imread('logo_ufes.png');
imshow(logoufes,'Parent',handles.logo);
% --------------------------------------------------------------------
```

```
% ------- Controla a quantidade de inicializações simultâneas ----------
handles.count = 0;
% --------------------------------------------------------------------------

% ------- Inicia LOG ---------------------------------------------------------
handles.variaveis.log = 0;
% --------------------------------------------------------------------------

% Atualiza a variável handles ----------------------------------------------
guidata(hObject, handles);
% --------------------------------------------------------------------------

% ========================================================================
function start_Callback(hObject, eventdata, handles)
% --- Executa quando se pressiona o botão start.
% Início do Algoritmo Evolutivo:
handles.count = handles.count + 1; % Para que caso o botao INICIAR seja
                                   % apertado novamente apos o inicio do
                                   % algoritmo, nao inicie o programa
                                   % novamente. Clicar mais de uma vez
                                   % impossibilitara o usuario de entrar
                                   % mais de uma vez no if abaixo.

% Mostra "Working...", dizendo que o programa está funcionando:
set(handles.wrkng,'Visible','on');

% Desabilita interações, para usuário não modificar nada durante execução
set(handles.num,'Enable','off');
set(handles.den,'Enable','off');
set(handles.Kp,'Enable','off');
set(handles.Kd,'Enable','off');
set(handles.Ki,'Enable','off');
```

```
set(handles.cross,'Enable','off');
set(handles.mut,'Enable','off');
set(handles.exc,'Enable','off');
set(handles.elite,'Enable','off');
set(handles.Ne,'Enable','off');
set(handles.Nde,'Enable','off');
set(handles.Nsa,'Enable','off');
set(handles.ger,'Enable','off');
set(handles.pop,'Enable','off');
set(handles.tsimu,'Enable','off');
set(handles.gemax,'Enable','off');
set(handles.gevaria,'Enable','off');
set(handles.salvarParametros,'Enable','off');
set(handles.restaurarParametros,'Enable','off');
set(handles.aged,'Enable','off');
set(handles.erromin,'Enable','off');
set(handles.erromax,'Enable','off');
set(handles.derromin,'Enable','off');
set(handles.derromax,'Enable','off');
set(handles.saidamin,'Enable','off');
set(handles.saidamax,'Enable','off');
set(handles.limites,'Enable','off');
set(handles.gerartxt,'Enable','off');
set(handles.help,'Enable','off');

% ================================================================
% Campos de Resultados:
% ------- Inicia os campos das variáveis observadas ----------------------
set(handles.ge,'BackgroundColor',[0.87 0.92 0.98]);
set(handles.fitness,'BackgroundColor',[0.87 0.92 0.98]);
if handles.variaveis.limites == 1
   set(handles.erro,'BackgroundColor',[0.87 0.92 0.98]);
   set(handles.derro,'BackgroundColor',[0.87 0.92 0.98]);
```

```
    set(handles.saida,'BackgroundColor',[0.87 0.92 0.98]);
end
% --------------------------------------------------------------------------
% Atualiza a variável handles
guidata(hObject, handles);

if( handles.count == 1)
    % ------- Inicia as configuracoes dos graficos ---------------------------
    cla(handles.axes1);
    cla(handles.axes2);
    cla(handles.mferro);
    cla(handles.mfderro);
    cla(handles.mfsaida);
    grid(handles.mferro,'on');
    xlabel(handles.mferro, 'Erro');
    axis(handles.mferro,[-handles.variaveis.erromax handles.variaveis.erromax 0 1]);
    grid(handles.mfderro,'on');
    xlabel(handles.mfderro, 'dErro');
    axis(handles.mfderro,[-handles.variaveis.derromax handles.variaveis.derromax 0 1]);
    grid(handles.mfsaida,'on');
    xlabel(handles.mfsaida, 'Saída');
    axis(handles.mfsaida,[-handles.variaveis.saidamax handles.variaveis.saidamax 0 1]);
    % --------------------------------------------------------------------------

    % ---------- Área de Resultados do Algoritmo -------------------------
    set(handles.erro,'String','...');
    set(handles.derro,'String','...');
    set(handles.saida,'String','...');
    set(handles.ge,'String','...');
    set(handles.fitness,'String','...');
    % Atualiza a variável handles
    guidata(hObject, handles);
    % --------------------------------------------------------------------------
```

```
% Número de variáveis linguísticas de erro
Ne  = (handles.variaveis.Ne - 1)/2;
% Número de variáveis linguísticas de derro
Nde = (handles.variaveis.Nde - 1)/2;
% Número de variáveis linguísticas de saida
Nsa = (handles.variaveis.Nsa - 1)/2;
GE  = handles.variaveis.gemax;

% Cria a variável de log do sistema, onde o melhor indivíduo de cada
% geração será salvo
if handles.variaveis.limites == 0
   handles.variaveis.log = zeros(handles.variaveis.max_gen,11+Ne+Nde+Nsa);
else
   handles.variaveis.log = zeros(handles.variaveis.max_gen,11+6);
end

% Algoritmo Evolutivo:
[ganhos handles] = calculaGanhos(GE,Ne,Nde,Nsa,handles);

% Após a execução, retorna o melhor indivíduo encontrado:
gevaria = handles.variaveis.gevaria;
if gevaria == 1
   set(handles.ge,'String',num2str(ganhos(1)));
else
   set(handles.ge,'String',num2str(GE));
end
set(handles.ge,'BackgroundColor',[0.89 0.94 0.9]);
if handles.variaveis.limites == 0
   set(handles.erro,'String',num2str(ganhos(1+gevaria:Ne+gevaria)));
   set(handles.derro,'String',num2str(ganhos(Ne+1+gevaria:Ne+Nde+gevaria)));

set(handles.saida,'String',num2str(ganhos((Ne+Nde)+1+gevaria:Ne+Nde+Nsa+gevaria)));
else
```

```
    set(handles.erro,'BackgroundColor',[0.89 0.94 0.9]);
    set(handles.derro,'BackgroundColor',[0.89 0.94 0.9]);
    set(handles.saida,'BackgroundColor',[0.89 0.94 0.9]);
    set(handles.erro,'String',num2str([-ganhos(1+gevaria) ganhos(1+gevaria)]));
    set(handles.derro,'String',num2str([-ganhos(2+gevaria) ganhos(2+gevaria)]));
    set(handles.saida,'String',num2str([-ganhos(3+gevaria) ganhos(3+gevaria)]));
end
% ================================================================

% Habilita novamente as interações com a interface:
set(handles.num,'Enable','on');
set(handles.den,'Enable','on');
set(handles.Kp,'Enable','on');
set(handles.Kd,'Enable','on');
set(handles.Ki,'Enable','on');
set(handles.cross,'Enable','on');
set(handles.mut,'Enable','on');
if handles.variaveis.aged == 1
    set(handles.exc,'Enable','on');
end
set(handles.elite,'Enable','on');
set(handles.Ne,'Enable','on');
set(handles.Nde,'Enable','on');
set(handles.Nsa,'Enable','on');
set(handles.ger,'Enable','on');
set(handles.pop,'Enable','on');
set(handles.tsimu,'Enable','on');
set(handles.gemax,'Enable','on');
set(handles.gevaria,'Enable','on');
set(handles.salvarParametros,'Enable','on');
set(handles.restaurarParametros,'Enable','on');
set(handles.aged,'Enable','on');
set(handles.erromin,'Enable','on');
```

```
    set(handles.erromax,'Enable','on');
    set(handles.derromin,'Enable','on');
    set(handles.derromax,'Enable','on');
    set(handles.saidamin,'Enable','on');
    set(handles.saidamax,'Enable','on');
    set(handles.limites,'Enable','on');
    set(handles.gerartxt,'Enable','on');
    set(handles.help,'Enable','on');
    % Devolve a possibilidade de usar o botão Iniciar
    handles.count = 0;

end

% Atualiza a variável handles
guidata(hObject, handles);
% ================================================================

function stop_Callback(hObject, eventdata, handles)
% Atualiza Flag de Parada:
assignin('base', 'stop_flag', true);
% Atualiza a variável handles
guidata(hObject, handles);
% --------------------------------------------------------------------------------
```

A.2 – calculaGanhos.m

```
function [individuo handles] = calculaGanhos(GE,Ne,Nde,Nsa,handles)
% Executa o algoritmo evolutivo para encontrar o melhor indivíduo:
[individuo handles] =
run_evolutionary_algorithms(GE,Ne,Nde,Nsa,handles.variaveis.max_gen,handles.variaveis.m
ax_pop,handles.variaveis.erromin,handles.variaveis.erromax,handles.variaveis.derromin,hand
les.variaveis.derromax,handles.variaveis.saidamin,handles.variaveis.saidamax,handles.variave
```

```
is.tx_cross,handles.variaveis.tx_mut,handles.variaveis.elite,handles.variaveis.tx_exclui,handle
s.variaveis.gevaria,handles.variaveis.aged,handles);
end
```

A.3 – run_evolutionary_algorithms.m

```
function [y handles] =
run_evolutionary_algorithms(GE,Ne,Nde,Nsa,maximum_generations,size_of_population,erro
min,erromax,derromin,derromax,saidamin,saidamax,tx_cross,tx_mut,elite,tx_exclui,gevaria,a
ged,handles)
% Função para iniciar o algoritmo
% maximum_generations => quantidade de gerações criadas.
% size_of_population => quantidade de indivíduos por população.
% erromax => tamanho máximo inicial do erro.
% derromax => tamanho máximo inicial da derivada do erro.
% saidamax => tamanho máximo inicial da saída.
% tx_cross => propabilidade de ocorrer crossover. Por exemplo: 0.9 = 90%.
% tx_mut => probalidade de ocorrer mutação. Por exemplo: 0.05 = 5%.
% elite => define se haverá ou não elitismo.
% tx_exclui => porcentagem de indivíduos excluídos entre gerações. Por exemplo: 0.05 =
5%.

% Verificação dos parâmetros:
a = maximum_generations;
b = round(maximum_generations);
c = a - b;
if c ~=0 || (maximum_generations < 0)
   error('Maximum_Generation deve ser um inteiro e positivo');
end
d = size_of_population;
e = round(size_of_population);
f = d - e;
if f ~= 0 || (size_of_population < 0)
```

```
  error('Size_of_population deve ser um inteiro positivo');
end

% Iniciando o algoritmo:
[population] =
create_population(GE,Ne,Nde,Nsa,size_of_population,erromin,erromax,derromin,derromax,s
aidamin,saidamax,gevaria,handles.variaveis.limites);       % Cria a população inicial

% Definicao de qual algoritmo:
if aged == 2 % Evolução Diferencial
   [y handles] =
execute_generation_ED(GE,Ne,Nde,Nsa,population,maximum_generations,tx_cross,tx_mut,e
lite,gevaria,handles.variaveis.limites,handles);              % Executa o loop de geração por ED
else        % Algoritmo Genético
   [y handles] =
execute_generation_AG(GE,Ne,Nde,Nsa,population,maximum_generations,tx_cross,tx_mut,
elite,tx_exclui,gevaria,handles.variaveis.limites,handles); % Executa o loop de geração por
AG
end
```

A.4 – create_population.m

```
function population = create_population
(GE,Ne,Nde,Nsa,size_of_population,erromin,erromax,derromin,derromax,saidamin,saidamax
,gevaria,limites)
% Função usada na criação da primeira população:
% Nparam - número de variáveis linguísticas;
% populacao = Matriz com indivíduos em linha:
% populacao = [individuo_1;
%             individuo_2;
%             individuo_3;
%             ...;...;...;
%             individuo_size_of_population]
```

```
% Caso limites = 1, os limites variam e as variáveis ficam fixas igualmente espaçadas.
if limites == 1
   Nparam = 3;
   raw_population = rand(size_of_population,Nparam);
   population = [erromax*raw_population(:,1) derromax*raw_population(:,2)
saidamax*raw_population(:,3)];
else
   Nparam = Ne+Nde+Nsa;
   raw_population = rand(size_of_population,Nparam);
   population = [sort(erromax*raw_population(:,1:Ne),2)
sort(derromax*raw_population(:,Ne+1:Ne+Nde),2)
sort(saidamax*raw_population(:,Ne+Nde+1:end),2)];
end

if gevaria == 1
population = [GE*rand(size_of_population,1) population];
end

end
```

A.5 – execute_generation_ED.m

```
function [best handles] =
execute_generation_ED(GEx,Ne,Nde,Nsa,population,maximum_generations,tx_cross,tx_mut,
elite,gevaria,limites,handles)
% Lê o valor do sinal stop exportado para o workspace no callback do botão stop
assignin('base', 'stop_flag', false);
current_generation = 0;        % Contador de gerações
best_fitness_antigo = 0;
ultima_alteracao = 1;
% Vetor fitness zerado somente para a primeira geração, onde acontecerá:
% fitness(nova) - fitness(antiga) = fitnessn - fitness
```

```
fitness = zeros(1,size(population,1));

% Para não escrever "handles.axes..." toda hora, faz-se ax apontar pra essa informação:
ax1 = handles.axes1;
ax2 = handles.axes2;
axe = handles.mferro;
axde = handles.mfderro;
axsa = handles.mfsaida;

population1 = population;       % Inicializando population1 para contas serem mais rápidas

% Loop de Gerações:
while (current_generation < maximum_generations && ~evalin('base','stop_flag'))
   current_generation = current_generation +1;

   % ------- Operadores -----------------------------------------
   % Passos 1 ao 5:
   if (current_generation > 1)
      population1 =
operador_ED(GEx,Ne,Nde,Nsa,population,tx_cross,tx_mut,elite,handles.variaveis.erromax,ha
ndles.variaveis.derromax,handles.variaveis.saidamax,gevaria,limites);
   end

   % Passo 6:
   [populationn, fitnessn] =
gera_fitness(GEx,Ne,Nde,Nsa,population1,gevaria,limites,handles); % Gera Fitness
   % fitness_nova > fitness_velha (troca-se o indivíduo pelo novo) Ex: auxn = [1 0 1 1 0];
         auxn = fitnessn>fitness;
         % auxn para a população
   auxnp = repmat(auxn',1,size(population,2));

   fitness(logical(auxn)) = fitnessn(logical(auxn));
   population(logical(auxnp)) = populationn(logical(auxnp));
```

```
      % Ordena a População e o vetor de Fitness em Ordem Decrescente:
[fitness,population] = quicksort(fitness,population);

% ---------------------------------------------------------------
% Atualiza as informações do melhor indivíduo:
best = population(1,:);
      best_fitness = fitness(1);
      handles.variaveis.fitness = best_fitness;
% Caso limites == 1 (varia), esses valores serão ignorados no escreve ponto fis central
if limites == 1
   Erro = 1/Ne:1/Ne:1;
   dErro = 1/Nde:1/Nde:1;
   Saida = 1/Nsa:1/Nsa:1;
   % Gerando os vetores com limite variáveis ====================
   Erro  = best(1+gevaria)*Erro;
   dErro = best(2+gevaria)*dErro;
   Saida = best(3+gevaria)*Saida;
   Erromax  = best(1+gevaria);
   dErromax = best(2+gevaria);
   Saidamax = best(3+gevaria);
else
   % Gerando os vetores para limites fixos ======================
   Erro = sort(best(1+gevaria:Ne+gevaria));
   dErro = sort(best(Ne+1+gevaria:Ne+Nde+gevaria));
   Saida = sort(best((Ne+Nde)+1+gevaria:Ne+Nde+Nsa+gevaria));
   Erromax  = handles.variaveis.erromax;
   dErromax = handles.variaveis.derromax;
   Saidamax = handles.variaveis.saidamax;
end

assignin('base', 'erro', Erro);
assignin('base', 'derro', dErro);
```

```
    assignin('base', 'saida', Saida);
    % Atualiza o .fis para ser executado no simulink (agfuzzyrules.fis):
    escreve_ponto_fis_central(Ne,Nde,Nsa,Erromax,dErromax,Saidamax,Erro,dErro,Saida);
    % Traz do Workspace os valores de Kp, Kd e Ki e calcula os valores de GE, GCE, GIE e
GU:
    Kp = evalin('base','Kp');
    Kd = evalin('base','Kd');
    Ki = evalin('base','Ki');

    if gevaria == 1
       GE = best(1);
    else
       GE = GEx;
    end
    set(handles.ge,'String',num2str(GE));
    set(handles.fitness,'String',num2str(handles.variaveis.fitness));

    if limites == 1
       set(handles.erro,'String',num2str([-Erromax Erromax]));
       set(handles.derro,'String',num2str([-dErromax dErromax]));
       set(handles.saida,'String',num2str([-Saidamax Saidamax]));
    else
       set(handles.erro,'String',num2str(Erro));
       set(handles.derro,'String',num2str(dErro));
       set(handles.saida,'String',num2str(Saida));
    end

    GU = Kp/GE;
    GCE = Kd/GU;
    GIE = Ki/GU;
    G05 = 1/GE;
    assignin('base','GE1',GE);
    assignin('base','GCE1',GCE);
```

```
    assignin('base','GIE1',GIE);
    assignin('base','GU',GU);
    assignin('base','G05',G05);
    assignin('base','erromin',-Erromax);
    assignin('base','erromax',Erromax);
    assignin('base','derromin',-dErromax);
    assignin('base','derromax',dErromax);
    assignin('base','saidamin',-Saidamax);
    assignin('base','saidamax',Saidamax);

    fisistema = readfis('agfuzzyrules.fis');
    assignin('base','fisistema',fisistema);
    sim('pidl');
    t = tempo;

    % Calcula os parâmetros para o LOG ----------------------------
    [mp_m,tr_m,tp_m,td_m,ts_m,erro_m] = calcula_parametros(t,y,ref);
    fit_m = best_fitness;
    assignin('base','mp_m',mp_m);
    assignin('base','fitness',fit_m);
    assignin('base','tr_m',tr_m);
    assignin('base','tp_m',tp_m);
    assignin('base','td_m',td_m);
    assignin('base','ts_m',ts_m);
    assignin('base','erro_m',erro_m);
    if limites == 1
       handles.variaveis.log(current_generation,:) = [mp_m tr_m fit_m GE GCE GIE GU tp_m
td_m ts_m erro_m -Erromax Erromax -dErromax dErromax -Saidamax Saidamax];
    else
       handles.variaveis.log(current_generation,:) = [mp_m tr_m fit_m GE GCE GIE GU tp_m
td_m ts_m erro_m Erro dErro Saida];
    end
    assignin('base','current_generation',current_generation);
```

```
% ------------------------------------------------------------

if (current_generation == maximum_generations)
   % Traçando o sistema:
   plot(ax1,t,y,'r',t,ref,'--g');
   drawnow
   assignin('base', 'yb', y);
   assignin('base', 'ref', ref);
   assignin('base', 'tb', t);
   assignin('base', 'uoutf', uoutf);
   assignin('base', 'uout', uout);
   % Traçando a fitness:
   plot(ax2,[ultima_alteracao current_generation],[best_fitness_antigo best_fitness], 'b');
   drawnow
   plot(ax2,current_generation, best_fitness, 'r.');
   drawnow
   figure,plot(t,y,'r',t,ref,'--g');
   h = legend('Controlador Fuzzy','Controlador Clássico',2,'Location','NorthEast');
   set(h,'Interpreter','none')
   title('Resposta ao Degrau');
   xlabel('Tempo (s)');
   ylabel('Amplitude');
   grid
   % Atualiza campo de fitness:
   set(handles.fitness,'String',num2str(handles.variaveis.fitness));
   set(handles.fitness,'BackgroundColor',[0.89 0.94 0.9]);
   % Fim da execução do programa:
   set(handles.wrkng,'Visible','off');
else
   % Traçando o sistema:
   plot(ax1,t,y,'b',t,ref,'--g');
                drawnow
   % Traçando a fitness:
```

```
        plot(ax2,[ultima_alteracao current_generation],[best_fitness_antigo best_fitness], 'b');
        drawnow
        limite_max = 1.2*best_fitness;
        axis(ax2,[0 maximum_generations (limite_max/3) limite_max]);
    end

    % Traçando as variáveis linguísticas nos gráficos:
    fisistema = readfis('agfuzzyrules');
    [xmf ymf] = plotmf(fisistema,'input',1);
    plot(axe,xmf,ymf);
    grid(handles.mferro,'on');
    xlabel(handles.mferro, 'Erro');
    axis(axe,[-Erromax Erromax 0 1]);
    drawnow
    [xmf ymf] = plotmf(fisistema,'input',2);
    plot(axde,xmf,ymf);
    grid(handles.mfderro,'on');
    xlabel(handles.mfderro, 'dErro');
    axis(axde,[-dErromax dErromax 0 1]);
    drawnow
    [xmf ymf] = plotmf(fisistema,'output',1);
    plot(axsa,xmf,ymf);
    grid(handles.mfsaida,'on');
    xlabel(handles.mfsaida, 'Saída');
    axis(axsa,[-Saidamax Saidamax 0 1]);
    drawnow
    ultima_alteracao = current_generation;
    best_fitness_antigo = best_fitness;
end

if evalin('base','stop_flag')
    % Traçando o sistema:
    plot(ax1,t,y,'r',t,ref,'--g');
```

```
drawnow
assignin('base', 'yb', y);
assignin('base', 'ref', ref);
assignin('base', 'tb', t);
assignin('base', 'uoutf', uoutf);
assignin('base', 'uout', uout);
% Traçando a fitness:
plot(ax2,[ultima_alteracao current_generation],[best_fitness_antigo best_fitness], 'b');
drawnow
limite_max = 1.2*best_fitness;
axis(ax2,[0 maximum_generations (limite_max/3) limite_max]);
plot(ax2,current_generation, best_fitness, 'r.');
drawnow
% Traçando as variáveis linguísticas:
fisistema = readfis('agfuzzyrules');
[xmf ymf] = plotmf(fisistema,'input',1);
plot(axe,xmf,ymf);
grid(handles.mferro,'on');
xlabel(handles.mferro, 'Erro');
axis(axe,[-Erromax Erromax 0 1]);
drawnow
[xmf ymf] = plotmf(fisistema,'input',2);
plot(axde,xmf,ymf);
grid(handles.mfderro,'on');
xlabel(handles.mfderro, 'dErro');
axis(axde,[-dErromax dErromax 0 1]);
drawnow
[xmf ymf] = plotmf(fisistema,'output',1);
plot(axsa,xmf,ymf);
grid(handles.mfsaida,'on');
xlabel(handles.mfsaida, 'Saída');
axis(axsa,[-Saidamax Saidamax 0 1]);
drawnow
```

```
   figure,plot(t,y,'r',t,ref,'--g');
   h = legend('Controlador Fuzzy','Controlador Clássico',2,'Location','NorthEast');
   set(h,'Interpreter','none')
   title('Resposta ao Degrau');
   xlabel('Tempo (s)');
   ylabel('Amplitude');
   grid
   % Atualiza campo de fitness:
   set(handles.fitness,'String',num2str(handles.variaveis.fitness));
   set(handles.fitness,'BackgroundColor',[0.89 0.94 0.9]);
   % Fim da execução do programa:
   set(handles.wrkng,'Visible','off');
end
```

A.6 – operador_ED.m

```
function [new_population] =
operador_ED(GE,Ne,Nde,Nsa,population,tx_cross,tx_mut,elite,erromax,derromax,saidamax,g
evaria,limites)
max_pop = size(population,1);                % Retorna o tamanho da população
max_ind = size(population,2);                % Retorna o tamanho do indivíduo
new_population = population;                 % Inicia a matriz new_population para agilizar
cálculos no Matlab

for k = 1+elite:max_pop                      % Elitismo ou não
   % 1 - Escolhe os outros 3 vetores para a operação (todos indivíduos
   % devem ser diferentes)
   vet = [0 0 0];                            % vet = [base r2 r3 xk]
   while (vet(1)==vet(2))||(vet(2)==vet(3))||(vet(3)==vet(1))||(vet(1)==k)||(vet(2)==k)||
(vet(3)==k)
      vet = round(1 + (max_pop-1)*rand(1,3));
   end
   base = population(vet(1),:);
```

```
   r2   = population(vet(2),:);
   r3   = population(vet(3),:);

   % 2 - Encontra o vetor diferenças (rd)
   rd = r2-r3;
   % 3 - Encontra o vetor mutação (Vo)
   Vo = tx_mut*rd + base;
   % 4 - Aplica o crossover
   vrand = rand(1,max_ind);
   xk = population(k,:);                    % Vetor alvo
   % Vetor de referência (Cr)
   auxo = vrand < tx_cross;                 % Cr = tx_cross, Cr > vrand (utiliza-se a informação do
vetor mutação) Ex: auxo = [1 0 1 1 0];

   Vp = xk;
   Vp(logical(auxo)) = Vo(logical(auxo));               % Gera o vetor prova Ex: Vp = [Vo(1)
xk(2) Vo(3) Vo(4) xk(5)];

   % 5 - Corrige o vetor prova dentro do seu universo
   if gevaria == 1
      Vpc = Vp(1);
      if (Vpc <= 0)          % Não se pode ter ganho negativo ou zero
         Vpc = abs(GE);
      end
      if (Vpc > GE)
         Vpc = GE;
      end
   else
      Vpc = [];
   end
   if limites == 1
      Vp(1+gevaria:end) = abs(Vp(1+gevaria:end));
          if (Vp(1+gevaria) > erromax)
```

```
            Vp(1+gevaria) = erromax;
        end
        if (Vp(2+gevaria) > derromax)
            Vp(2+gevaria) = derromax;
        end
        if (Vp(3+gevaria) > saidamax)
            Vp(3+gevaria) = saidamax;
        end
        new_population(k,:) = [Vpc Vp(1+gevaria:end)];
    else
        Erro  = Vp(1+gevaria:Ne+gevaria);
        dErro = Vp(Ne+1+gevaria:Ne+Nde+gevaria);
        Saida = Vp(Ne+Nde+1+gevaria:Ne+Nde+Nsa+gevaria);
        % Para corrigir o vetor, as variáveis são saturadas em seus limites
        Erro(logical(Erro < 0)) = 0;
        Erro(logical(Erro > erromax)) = erromax;
        dErro(logical(dErro < 0)) = 0;
        dErro(logical(dErro > derromax)) = derromax;
        Saida(logical(Saida < 0)) = 0;
        Saida(logical(Saida > saidamax)) = saidamax;
        new_population(k,:) = [Vpc Erro dErro Saida];
    end
end
```

A.7 – gera_fitness.m

```
function [individuos,fitness] =
gera_fitness(GE,Ne,Nde,Nsa,population,gevaria,limites,handles)
% Função que calcula a fitness de todos os indivíduos da população:
warning off;
nmax = size(population,1);
fitness = zeros(1,nmax);
for individual_index = 1:nmax
```

```
    fitness(individual_index) =
fitness_individuo(GE,Ne,Nde,Nsa,population(individual_index,:),gevaria,limites,handles);
end
individuos = population;
end
```

A.8 – fitness_individuo.m

```
function fitness = fitness_individuo(GEx,Ne,Nde,Nsa,individuo,gevaria,limites,handles)
% Função que calcula a fitness de cada indivíduo da população:
if limites == 1
    Erro = 1/Ne:1/Ne:1;
    dErro = 1/Nde:1/Nde:1;
    Saida = 1/Nsa:1/Nsa:1;
    % Gerando os vetores com limite variáveis ========================
    Erro  = individuo(1+gevaria)*Erro;
    dErro = individuo(2+gevaria)*dErro;
    Saida = individuo(3+gevaria)*Saida;
    Erromax  = individuo(1+gevaria);
    dErromax = individuo(2+gevaria);
    Saidamax = individuo(3+gevaria);
else
    % Gerando os vetores para limites fixos ==========================
    Erro = sort(individuo(1+gevaria:Ne+gevaria));
    dErro = sort(individuo(Ne+1+gevaria:Ne+Nde+gevaria));
    Saida = sort(individuo((Ne+Nde)+1+gevaria:Ne+Nde+Nsa+gevaria));
    Erromax  = handles.variaveis.erromax;
    dErromax = handles.variaveis.derromax;
    Saidamax = handles.variaveis.saidamax;
end

assignin('base', 'erro', Erro);
assignin('base', 'derro', dErro);
```

```
assignin('base', 'saida', Saida);
% Atualiza o .fis para ser executado no simulink (agfuzzyrules.fis)
escreve_ponto_fis_central(Ne,Nde,Nsa,Erromax,dErromax,Saidamax,Erro,dErro,Saida);

% Tras do Workspace os valores de Kp, Kd e Ki e calcula os valores de
% GE, GCE, GIE e GU:
Kp = evalin('base','Kp');
Kd = evalin('base','Kd');
Ki = evalin('base','Ki');

if gevaria == 1
   GE = individuo(1);
else
   GE = GEx;
end

GU = Kp/GE;
GCE = Kd/GU;
GIE = Ki/GU;
G05 = 1/GE;
assignin('base','GE1',GE);
assignin('base','GCE1',GCE);
assignin('base','GIE1',GIE);
assignin('base','GU',GU);
assignin('base','G05',G05);
assignin('base','erromin',-Erromax);
assignin('base','erromax',Erromax);
assignin('base','derromin',-dErromax);
assignin('base','derromax',dErromax);

fisistema = readfis('agfuzzyrules.fis');
assignin('base','fisistema',fisistema);
sim('pidl.mdl');                % Executa a simulação
```

```
% Forma de calcular a fitness
degrauini = evalin('base','degrauini');
degraufim = evalin('base','degraufim');
ref_n = (ref-degrauini)/(degraufim-degrauini);% Normalizando a referência como se fosse sempre para degrau unitário
y_n = (y-degrauini)/(degraufim-degrauini);% Normalizando a resposta ao degrau como se fosse sempre para degrau unitário
area = sum((ref_n-y_n).^2);
if( area == 0)
    area = 0.000000000000001;
end
fitness = 1/area;
end
```

A.9 – quicksort.m

```
function [x,y] = quicksort(x,y)
% Função que ordena os vetores x e y de acordo com o vetor x
% fazendo com que o vetor linha x fique em ordem decrescente:
% Ex:
% x = [0.7 0.5 10 2]; y = [1 2;3 4;5 6;7 8];
% [xn yn] = quicksort(x,y);
% xn = [10 2 0.7 0.5];
% yn = [5 6;7 8;1 2;3 4];
if (length(x) > 1)
   pos_pivot = 1+round((length(x)-1)*rand(1));
   pivot = x(pos_pivot);
   pivot_y = y(pos_pivot,:);
   x(pos_pivot) = [];
   y(pos_pivot,:) = [];
   maiores = x>pivot;
   [Ax,Ay] = quicksort(x(logical(maiores)),y(logical(maiores),:));
```

```
    [Bx,By] = quicksort(x(logical(~maiores)),y(logical(~maiores),:));
    x = [Ax pivot Bx];
    y = [Ay;pivot_y;By];
end
end
```

A.10 – escreve_ponto_fis_central.m

```
function [] =
escreve_ponto_fis_central(Ne,Nde,Nsa,Erromax,dErromax,Saidamax,Erro,dErro,Saida)
cont1=1;
ne  = (2*Ne+1);
nde = (2*Nde+1);
nsa = (2*Nsa+1);
Erro  = [-Erro(Ne:-1:1) 0 Erro];
dErro = [-dErro(Nde:-1:1) 0 dErro];
Saida = [-Saida(Nsa:-1:1) 0 Saida];

fid = fopen('agfuzzyrules.fis', 'wt+');
fprintf(fid,
'[System]\nName="agfuzzyrules"\nType="mamdani"\nVersion=2.0\nNumInputs=2\nNumOut
puts=1\nNumRules= %d
\nAndMethod="min"\nOrMethod="max"\nImpMethod="min"\nAggMethod="max"\nDefuzz
Method="centroid"\n\n',ne*nde);
% Escreve a entrada Erro
fprintf(fid,'[Input1]\nName="Erro"\nRange=[%f %f]\n',-Erromax,Erromax);
fprintf(fid, 'NumMFs=%d\nMF%d="A%d":"trapmf",[%f %f %f %f]\n',ne,cont1,cont1,-
Erromax,-Erromax,Erro(1),Erro(2));
for k = 1:ne-2
    cont1 = cont1+1;
    if k == (ne-rem(ne,2))/2
        fprintf(fid, 'MF%d="ZE":"trimf",[%f %f %f]\n',cont1,Erro(k),Erro(k+1),Erro(k+2));
    else
```

```
        fprintf(fid, 'MF%d="A%d":"trimf",[%f %f
%f]\n',cont1,cont1,Erro(k),Erro(k+1),Erro(k+2));
    end
end
cont1 = cont1+1;
fprintf(fid, 'MF%d="A%d":"trapmf",[%f %f %f %f]\n\n',cont1,cont1,Erro(ne-
1),Erro(ne),Erromax,Erromax);
% Escreve a entrada dErro
cont1 = 1;
fprintf(fid,'[Input2]\nName="dErro"\nRange=[%f %f]\n',-dErromax,dErromax);
fprintf(fid, 'NumMFs=%d\nMF%d="A%d":"trapmf",[%f %f %f %f]\n',nde,cont1,cont1,-
dErromax,-dErromax,dErro(1),dErro(2));
for k = 1:nde-2
    cont1 = cont1+1;
    if k == (nde-rem(nde,2))/2
        fprintf(fid, 'MF%d="ZE":"trimf",[%f %f %f]\n',cont1,dErro(k),dErro(k+1),dErro(k+2));
    else
        fprintf(fid, 'MF%d="A%d":"trimf",[%f %f
%f]\n',cont1,cont1,dErro(k),dErro(k+1),dErro(k+2));
    end
end
cont1 = cont1+1;
fprintf(fid, 'MF%d="A%d":"trapmf",[%f %f %f %f]\n\n',cont1,cont1,dErro(nde-
1),dErro(nde),dErromax,dErromax);
% Escreve a saída Saida
cont1 = 1;
fprintf(fid,'[Output1]\nName="Saida"\nRange=[%f %f]\n',-Saidamax,Saidamax);
fprintf(fid, 'NumMFs=%d\nMF%d="A%d":"trapmf",[%f %f %f %f]\n',nsa,cont1,cont1,-
Saidamax,-Saidamax,Saida(1),Saida(2));
for k = 1:nsa-2
    cont1 = cont1+1;
    if k == (nsa-rem(nsa,2))/2
        fprintf(fid, 'MF%d="ZE":"trimf",[%f %f %f]\n',cont1,Saida(k),Saida(k+1),Saida(k+2));
```

```
   else
      fprintf(fid, 'MF%d="A%d":"trimf",[%f %f
%f]\n',cont1,cont1,Saida(k),Saida(k+1),Saida(k+2));
   end
end
cont1 = cont1+1;
fprintf(fid, 'MF%d="A%d":"trapmf",[%f %f %f %f]\n\n',cont1,cont1,Saida(nsa-
1),Saida(nsa),Saidamax,Saidamax);

% Escrever regras para N variáveis linguísticas:
switch ne
   case 7 % N = 7
      fprintf(fid,'[Rules]\n');
      fprintf(fid, '1 1, 1 (1) : 1\n');
      fprintf(fid, '1 2, 1 (1) : 1\n');
      fprintf(fid, '1 3, 1 (1) : 1\n');
      fprintf(fid, '1 4, 1 (1) : 1\n');
      fprintf(fid, '1 5, 2 (1) : 1\n');
      fprintf(fid, '1 6, 3 (1) : 1\n');
      fprintf(fid, '1 7, 4 (1) : 1\n');
      fprintf(fid, '2 1, 1 (1) : 1\n');
      fprintf(fid, '2 2, 1 (1) : 1\n');
      fprintf(fid, '2 3, 1 (1) : 1\n');
      fprintf(fid, '2 4, 2 (1) : 1\n');
      fprintf(fid, '2 5, 3 (1) : 1\n');
      fprintf(fid, '2 6, 4 (1) : 1\n');
      fprintf(fid, '2 7, 5 (1) : 1\n');
      fprintf(fid, '3 1, 1 (1) : 1\n');
      fprintf(fid, '3 2, 1 (1) : 1\n');
      fprintf(fid, '3 3, 2 (1) : 1\n');
      fprintf(fid, '3 4, 3 (1) : 1\n');
      fprintf(fid, '3 5, 4 (1) : 1\n');
      fprintf(fid, '3 6, 5 (1) : 1\n');
```

```
    fprintf(fid, '3 7, 6 (1) : 1\n');
    fprintf(fid, '4 1, 1 (1) : 1\n');
    fprintf(fid, '4 2, 2 (1) : 1\n');
    fprintf(fid, '4 3, 3 (1) : 1\n');
    fprintf(fid, '4 4, 4 (1) : 1\n');
    fprintf(fid, '4 5, 5 (1) : 1\n');
    fprintf(fid, '4 6, 6 (1) : 1\n');
    fprintf(fid, '4 7, 7 (1) : 1\n');
    fprintf(fid, '5 1, 2 (1) : 1\n');
    fprintf(fid, '5 2, 3 (1) : 1\n');
    fprintf(fid, '5 3, 4 (1) : 1\n');
    fprintf(fid, '5 4, 5 (1) : 1\n');
    fprintf(fid, '5 5, 6 (1) : 1\n');
    fprintf(fid, '5 6, 7 (1) : 1\n');
    fprintf(fid, '5 7, 7 (1) : 1\n');
    fprintf(fid, '6 1, 3 (1) : 1\n');
    fprintf(fid, '6 2, 4 (1) : 1\n');
    fprintf(fid, '6 3, 5 (1) : 1\n');
    fprintf(fid, '6 4, 6 (1) : 1\n');
    fprintf(fid, '6 5, 7 (1) : 1\n');
    fprintf(fid, '6 6, 7 (1) : 1\n');
    fprintf(fid, '6 7, 7 (1) : 1\n');
    fprintf(fid, '7 1, 4 (1) : 1\n');
    fprintf(fid, '7 2, 5 (1) : 1\n');
    fprintf(fid, '7 3, 6 (1) : 1\n');
    fprintf(fid, '7 4, 7 (1) : 1\n');
    fprintf(fid, '7 5, 7 (1) : 1\n');
    fprintf(fid, '7 6, 7 (1) : 1\n');
    fprintf(fid, '7 7, 7 (1) : 1\n');

case 5 % N = 5
    fprintf(fid,'[Rules]\n');
    fprintf(fid, '1 1, 1 (1) : 1\n');
```

```
    fprintf(fid, '1 2, 1 (1) : 1\n');
    fprintf(fid, '1 3, 1 (1) : 1\n');
    fprintf(fid, '1 4, 2 (1) : 1\n');
    fprintf(fid, '1 5, 3 (1) : 1\n');
    fprintf(fid, '2 1, 1 (1) : 1\n');
    fprintf(fid, '2 2, 1 (1) : 1\n');
    fprintf(fid, '2 3, 2 (1) : 1\n');
    fprintf(fid, '2 4, 3 (1) : 1\n');
    fprintf(fid, '2 5, 4 (1) : 1\n');
    fprintf(fid, '3 1, 1 (1) : 1\n');
    fprintf(fid, '3 2, 2 (1) : 1\n');
    fprintf(fid, '3 3, 3 (1) : 1\n');
    fprintf(fid, '3 4, 4 (1) : 1\n');
    fprintf(fid, '3 5, 5 (1) : 1\n');
    fprintf(fid, '4 1, 2 (1) : 1\n');
    fprintf(fid, '4 2, 3 (1) : 1\n');
    fprintf(fid, '4 3, 4 (1) : 1\n');
    fprintf(fid, '4 4, 5 (1) : 1\n');
    fprintf(fid, '4 5, 5 (1) : 1\n');
    fprintf(fid, '5 1, 3 (1) : 1\n');
    fprintf(fid, '5 2, 4 (1) : 1\n');
    fprintf(fid, '5 3, 5 (1) : 1\n');
    fprintf(fid, '5 4, 5 (1) : 1\n');
    fprintf(fid, '5 5, 5 (1) : 1\n');

case 3 % N = 3
    fprintf(fid,'[Rules]\n');
    fprintf(fid, '1 1, 1 (1) : 1\n');
    fprintf(fid, '1 2, 1 (1) : 1\n');
    fprintf(fid, '1 3, 2 (1) : 1\n');
    fprintf(fid, '2 1, 1 (1) : 1\n');
    fprintf(fid, '2 2, 2 (1) : 1\n');
    fprintf(fid, '2 3, 3 (1) : 1\n');
```

```
        fprintf(fid, '3 1, 2 (1) : 1\n');
        fprintf(fid, '3 2, 3 (1) : 1\n');
        fprintf(fid, '3 3, 3 (1) : 1\n');

    otherwise % N diferente de 7, 5 e 3
        fprintf(fid,'[Rules]\n');
        x = [ones(1,(ne-rem(ne,2))/2) 1:ne ne*ones(1,(ne-rem(ne,2))/2)];
        for k = 1:1:ne
            for l = 1:1:ne
                fprintf(fid, '%d %d, %d (1) : 1\n',k,l,x(k+l-1));
            end
        end
end

fclose(fid);
end
```

A.11 – calcula_parametros.m

```
function [mp_m,tr_m,tp_m,td_m,ts_m,erro_m] = calcula_parametros(t,y,ref)
% ------------------- Parâmetros Importantes ----------------------
degrauini = evalin('base','degrauini');
degraufim = evalin('base','degraufim');
mag_d = (degraufim-degrauini)/2;

% Para que o cálculo pelo stepinfo seja feito para degraus que saem de
% valores altos para baixos, as respostas devem sair de um valor baixo para
% um valor alto. Portanto, inverte-se o degrau:
if mag_d < 0
    ref = (sign(mag_d))*(ref - degrauini);
    y = (sign(mag_d))*(y - degrauini);
    degraufim = abs(degraufim - degrauini);
    degrauini = 0;
```

```
end

% ------------------------- Cálculos -----------------------------
% Cálculo do sobressinal, tempo de subida, tempo de pico, tempo de
% assentamento.
% Como o programa está configurado no simulink para gerar uma perturbação
% no meio da simulação, a análise é feita antes dela ocorrer.
% Obs: caso a perturbação seja tirada, mudar "perturbacao" para 0.
perturbacao = 1; % 1 - Sim, 0 - Nao
if perturbacao == 1
   % ------ Com Perturbação ------
   onde_max = not(t < t(end)/2);
   amostra_max = find(onde_max,1);
   fiminfo = stepinfo(y(1:amostra_max-1),t(1:amostra_max-1),degraufim);
else
   % ------ Sem Perturbação ------
   fiminfo = stepinfo(y,t,degraufim);
end

% Cálculo do tempo de atraso e do erro estacionário:
assignin('base','tb',t);
assignin('base','yb',y);
onde_d = y >= abs(mag_d)+abs(degrauini);
amostra_d = find(onde_d,1);
td_m = (t(amostra_d)+t(amostra_d-1))/2;

if perturbacao == 1
   y_aux = y(1:amostra_max-1);
   media = mean(y_aux(round(4*length(y_aux)/5)));
   erro_m = degraufim - media;
else
   media = mean(y(round(4*legnth(y)/5)));
   erro_m = degraufim - media;
```

```
end
% ---------------------- Preparando a saída ----------------------
mp_m = fiminfo.Overshoot;
tr_m = fiminfo.RiseTime;
tp_m = fiminfo.PeakTime;
ts_m = fiminfo.SettlingTime;
end
```

A.12 – execute_generation_AG.m

```
function [best handles] =
execute_generation_AG(GEx,Ne,Nde,Nsa,population,maximum_generations,tx_cross,tx_mut
,elite,tx_exclui,gevaria,limites,handles)
% Lê o valor do sinal stop exportado para o workspace no callback do botão stop:
assignin('base', 'stop_flag', false);
current_generation = 0;         % Contador de gerações
best_fitness_antigo = 0;
ultima_alteracao = 1;

% Para não escrever "handles.axes..." toda hora, faz-se ax apontar pra essa informação:
ax1 = handles.axes1;
ax2 = handles.axes2;
axe = handles.mferro;
axde = handles.mfderro;
axsa = handles.mfsaida;

population1 = population;       % Inicializando population1 para contas serem mais rápidas

% Loop de Gerações:
while (current_generation < maximum_generations && ~evalin('base','stop_flag'))
   current_generation = current_generation +1;

   % ------- Operadores ----------------------------------------
```

```
    % Faz Operações de Crossover e Mutação:
    if (current_generation > 1)
        population1 =
new_population(GEx,Ne,Nde,Nsa,population,fitness,tx_cross,tx_mut,elite,tx_exclui,handles.v
ariaveis.erromin, handles.variaveis.erromax,handles.variaveis.derromin,
handles.variaveis.derromax, handles.variaveis.saidamin, handles.variaveis.saidamax, gevaria,
limites);
    end

    % Encontra a fitness da população:
     [population, fitness] = gera_fitness(GEx,Ne,Nde,Nsa,population1,gevaria,limites,handles);

    % Ordena a População e o vetor de Fitness em Ordem Decrescente:
     [fitness,population] = quicksort(fitness,population);

    % ------------------------------------------------------------
    % Atualiza as informações do melhor indivíduo:
    best = population(1,:);
    best_fitness = fitness(1);
    handles.variaveis.fitness = best_fitness;
    % Caso limites == 1 (limite varia), esses valores serão ignorados no escreve ponto fis
central
    if limites == 1
        Erro = 1/Ne:1/Ne:1;
        dErro = 1/Nde:1/Nde:1;
        Saida = 1/Nsa:1/Nsa:1;
        % Gerando os Vetores com Limite Variaveis ====================
        Erro  = best(1+gevaria)*Erro;
        dErro = best(2+gevaria)*dErro;
        Saida = best(3+gevaria)*Saida;
        Erromax  = best(1+gevaria);
        dErromax = best(2+gevaria);
        Saidamax = best(3+gevaria);
```

```
    else
        % Gerando os vetores para limites fixos ======================
        Erro = best(1+gevaria:Ne+gevaria);
        dErro = best(Ne+1+gevaria:Ne+Nde+gevaria);
        Saida = best((Ne+Nde)+1+gevaria:Ne+Nde+Nsa+gevaria);
        Erromax  = handles.variaveis.erromax;
        dErromax = handles.variaveis.derromax;
        Saidamax = handles.variaveis.saidamax;
    end

    assignin('base', 'erro', Erro);
    assignin('base', 'derro', dErro);
    assignin('base', 'saida', Saida);
    % Atualiza o .fis para ser executado no simulink (agfuzzyrules.fis):
    escreve_ponto_fis_central(Ne,Nde,Nsa,Erromax,dErromax,Saidamax,Erro,dErro,Saida);
    % Traz do Workspace os valores de Kp, Kd e Ki e calcula os valores de GE, GCE, GIE e
GU:
    Kp = evalin('base','Kp');
    Kd = evalin('base','Kd');
    Ki = evalin('base','Ki');

    if gevaria == 1
        GE = best(1);
    else
        GE = GEx;
    end
    set(handles.ge,'String',num2str(GE));
    set(handles.fitness,'String',handles.variaveis.fitness);

    if limites == 1
        set(handles.erro,'String',num2str([-Erromax Erromax]));
        set(handles.derro,'String',num2str([-dErromax dErromax]));
        set(handles.saida,'String',num2str([-Saidamax Saidamax]));
```

```
else
   set(handles.erro,'String',num2str(Erro));
   set(handles.derro,'String',num2str(dErro));
   set(handles.saida,'String',num2str(Saida));
end

GU = Kp/GE;
GCE = Kd/GU;
GIE = Ki/GU;
G05 = 1/GE;
assignin('base','GE1',GE);
assignin('base','GCE1',GCE);
assignin('base','GIE1',GIE);
assignin('base','GU',GU);
assignin('base','G05',G05);
assignin('base','erromin',-Erromax);
assignin('base','erromax',Erromax);
assignin('base','derromin',-dErromax);
assignin('base','derromax',dErromax);
assignin('base','saidamin',-Saidamax);
assignin('base','saidamax',Saidamax);

fisistema = readfis('agfuzzyrules.fis');
assignin('base','fisistema',fisistema);
sim('pidl');
t = tempo;

% Calcula os parâmetros para o LOG ----------------------------
 [mp_m,tr_m,tp_m,td_m,ts_m,erro_m] = calcula_parametros(t,y,ref);
fit_m = best_fitness;
assignin('base','mp_m',mp_m);
assignin('base','fitness',fit_m);
assignin('base','tr_m',tr_m);
```

```
    assignin('base','tp_m',tp_m);
    assignin('base','td_m',td_m);
    assignin('base','ts_m',ts_m);
    assignin('base','erro_m',erro_m);
    if limites == 1
        handles.variaveis.log(current_generation,:) = [mp_m tr_m fit_m GE GCE GIE GU tp_m
td_m ts_m erro_m -Erromax Erromax -dErromax dErromax -Saidamax Saidamax];
    else
        handles.variaveis.log(current_generation,:) = [mp_m tr_m fit_m GE GCE GIE GU tp_m
td_m ts_m erro_m Erro dErro Saida];
    end
    assignin('base','current_generation',current_generation);
    % ------------------------------------------------------------

    if (current_generation == maximum_generations)
        % Traçando o sistema:
        plot(ax1,t,y,'r',t,ref,'--g');
        drawnow
        assignin('base', 'yb', y);
        assignin('base', 'ref', ref);
        assignin('base', 'tb', t);
        assignin('base', 'uoutf', uoutf);
        assignin('base', 'uout', uout);
        % Traçando a fitness:
        plot(ax2,[ultima_alteracao current_generation],[best_fitness_antigo best_fitness], 'b');
        drawnow
        plot(ax2,current_generation, best_fitness, 'r.');
        drawnow
        figure,plot(t,y,'r',t,ref,'--g');
        h = legend('Controlador Fuzzy','Controlador Clássico',2,'Location','NorthEast');
        set(h,'Interpreter','none')
        title('Resposta ao Degrau');
        xlabel('Tempo (s)');
```

```
    ylabel('Amplitude');
    grid
    % Atualiza campo de fitness:
    set(handles.fitness,'String',num2str(handles.variaveis.fitness));
    set(handles.fitness,'BackgroundColor',[0.89 0.94 0.9]);
    % Fim da execução do programa:
    set(handles.wrkng,'Visible','off');
else
    % Traçando o sistema:
    plot(ax1,t,y,'b',t,ref,'--g');
    drawnow
    % Traçando a fitness:
    plot(ax2,[ultima_alteracao current_generation],[best_fitness_antigo best_fitness], 'b');
    drawnow
    limite_max = 1.2*best_fitness;
    axis(ax2,[0 maximum_generations (limite_max/3) limite_max]);
end

% Traçando as variáveis linguísticas:
fisistema = readfis('agfuzzyrules');
[xmf ymf] = plotmf(fisistema,'input',1);
plot(axe,xmf,ymf);
grid(axe,'on');
xlabel(axe, 'Erro');
axis(axe,[-Erromax Erromax 0 1]);
drawnow
[xmf ymf] = plotmf(fisistema,'input',2);
plot(axde,xmf,ymf);
grid(axde,'on');
xlabel(axde, 'dErro');
axis(axde,[-dErromax dErromax 0 1]);
drawnow
[xmf ymf] = plotmf(fisistema,'output',1);
```

```
    plot(axsa,xmf,ymf);
    grid(axsa,'on');
    xlabel(axsa, 'Saída');
    axis(axsa,[-Saidamax Saidamax 0 1]);
    drawnow
    ultima_alteracao = current_generation;
    best_fitness_antigo = best_fitness;
end

if evalin('base','stop_flag')
    % Traçando o sistema:
    plot(ax1,t,y,'r',t,ref,'--g');
    drawnow
    assignin('base', 'yb', y);
    assignin('base', 'ref', ref);
    assignin('base', 'tb', t);
    assignin('base', 'uoutf', uoutf);
    assignin('base', 'uout', uout);
    % Traçando a fitness:
    plot(ax2,[ultima_alteracao current_generation],[best_fitness_antigo best_fitness], 'b');
    drawnow
    limite_max = 1.2*best_fitness;
    axis(ax2,[0 maximum_generations (limite_max/3) limite_max]);
    plot(ax2,current_generation, best_fitness, 'r.');
    drawnow
    % Traçando as variáveis linguísticas:
    fisistema = readfis('agfuzzyrules');
    [xmf ymf] = plotmf(fisistema,'input',1);
    plot(axe,xmf,ymf);
    grid(axe,'on');
    xlabel(axe, 'Erro');
    axis(axe,[-Erromax Erromax 0 1]);
    drawnow
```

```
    [xmf ymf] = plotmf(fisistema,'input',2);
    plot(axde,xmf,ymf);
    grid(axde,'on');
    xlabel(axde, 'dErro');
    axis(axde,[-dErromax dErromax 0 1]);
    drawnow
    [xmf ymf] = plotmf(fisistema,'output',1);
    plot(axsa,xmf,ymf);
    grid(axsa,'on');
    xlabel(axsa, 'Saída');
    axis(axsa,[-Saidamax Saidamax 0 1]);
    drawnow
    figure,plot(t,y,'r',t,ref,'--g');
    h = legend('Controlador Fuzzy','Controlador Clássico',2,'Location','NorthEast');
    set(h,'Interpreter','none')
    title('Resposta ao Degrau');
    xlabel('Tempo (s)');
    ylabel('Amplitude');
    grid
    % Atualiza campo de fitness:
    set(handles.fitness,'String',num2str(handles.variaveis.fitness));
    set(handles.fitness,'BackgroundColor',[0.89 0.94 0.9]);
    % Fim da execução do programa:
    set(handles.wrkng,'Visible','off');
end
```

A.13 – new_population.m

```
function [n_population] =
new_population(GE,Ne,Nde,Nsa,population,fitness,tx_cross,tx_mut,elite,tx_exclui,erromin,er
romax,derromin,derromax,saidamin,saidamax,gevaria,limites)

max = size(population,1);              % max = tamanho máximo da população
```

```
excluido = round(max*tx_exclui);  % Determina quantidade que será excluída da população
n_population = population;            % Copia para a nova população os elementos elite
k = elite;                                % Define Elitismo

population((end-excluido+1:end),:) = [];    % Exclusão
fitness((end-excluido+1:end)) = [];           % Exclusão

% ------- Roleta ----------------------------------------------------
sft = sum(fitness);               % Implementa a roleta
psel = fitness/sft;               % Calcula a propabilidade de cada indivíduo
cpsel = cumsum(psel);         % Estabelece a "área" de cada indivíduo proporcionalmente a sua
aptidão
N=max;                            % Escrevendo de Forma mais didatica
II = zeros(1,N);
for ii=1:N                        % Roda a roleta n vezes selecionando um indivíduo aleatoriamente
   r = rand(1);
   R = cpsel > r;
   I = find(R,1);              % Retorna a posição do primeiro indivíduo cuja área é maior que r
   II(ii) = I;
end
population = population(II,:);      % População formada com os indivíduos escolhidos na
roleta
% -------------------------------------------------------------------

% ------- Operações -------------------------------------------------
population =
operador_AG(GE,Ne,Nde,Nsa,population,tx_cross,tx_mut,0,erromax,0,derromax,0,saidamax,
gevaria,limites);       % Realiza crossover e mutação

% Caso esteja usando elitismo, a nova população ficaria com o indivíduo de
% elite, mais N(tamanho da população) indivíduos gerados pela roleta,
% portanto, é necessário retirar esse indivíduo a mais
n_population(k+1:end,:) = population(k+1:end,:);
```

```
% ------------------------------------------------------------------
end
```

A.14 – operador_AG.m

```
function [n_population] =
operador_AG(GEmax,Ne,Nde,Nsa,population,tx_cross,tx_mut,erromin,erromax,derromin,der
romax,saidamin,saidamax,gevaria,limites)
% Função responsável por fazer as operações do Algoritmo Genético:
% limites == 1 significa que os limites variam:
size_of_population = size(population,1);
if limites == 1
   % Crossover:
   m = population(1:2:end-1,:);
   f = population(2:2:end,:);

   xss = rand(size(m,1),1);
   e = round(2 + (size(m,2)-2)*rand(1,size(m,1)));
   nm = m;
   nf = f;
   for k = 1:size(m,1)
      if xss(k) <= tx_cross
         nm(k,:) = [m(k,1:(e(k)-1)) f(k,e(k):end)];
         nf(k,:) = [f(k,1:(e(k)-1)) m(k,e(k):end)];
      end
   end

   n_population = zeros(size(population));
   n_population(1:2:end-1) = nm;
   n_population(2:2:end) = nf;

   % Mutação:
   taxa_mutacao = rand(size(population));
```

```
    mut = taxa_mutacao <= tx_mut;
    if gevaria == 1
        mut_popu = repmat([GEmax erromax derromax
saidamax],size_of_population,1).*rand(size_of_population,size(m,2));
    else
        mut_popu = repmat([erromax derromax
saidamax],size_of_population,1).*rand(size_of_population,size(m,2));
    end
    n_population(logical(mut)) = mut_popu(logical(mut));

else

    % Crossover:
    m = population(1:2:end-1,:);
    f = population(2:2:end,:);

    xss = rand(size(m,1),1);
    e = round(2 + (size(m,2)-2)*rand(1,size(m,1)));
    nm = m;
    nf = f;
    for k = 1:size(m,1)
        if xss(k) <= tx_cross
            nm(k,:) = [m(k,1:(e(k)-1)) f(k,e(k):end)];
            nf(k,:) = [f(k,1:(e(k)-1)) m(k,e(k):end)];
        end
    end

    n_population = zeros(size(population));
    n_population(1:2:end-1) = nm;
    n_population(2:2:end) = nf;

    % Mutação:
    taxa_mutacao = rand(size(population));
```

```
    mut = taxa_mutacao <= tx_mut;
    if gevaria == 1
        mut_popu = [GEmax*ones(size_of_population,1) erromax*ones(size_of_population,Ne)
derromax*ones(size_of_population,Nde)
saidamax*ones(size_of_population,Nsa)].*rand(size(population));
    else
        mut_popu = [erromax*ones(size_of_population,Ne)
derromax*ones(size_of_population,Nde)
saidamax*ones(size_of_population,Nsa)].*rand(size(population));
    end
    n_population(logical(mut)) = mut_popu(logical(mut));

    % Acertando a ordem das variáveis Fuzzy:
    n_population(:,1+gevaria:end) = [sort(n_population(:,1+gevaria:Ne+gevaria),2)
sort(n_population(:,Ne+1+gevaria:Ne+Nde+gevaria),2)
sort(n_population(:,Ne+Nde+1+gevaria:Ne+Nde+Nsa+gevaria),2)];
end
end
```