Algoritmos Genéticos para Otimização de Controladores Fuzzy

GABRIEL PINHEIRO CAVALCANTI GUERRA SEABRA[1], JACIMARA VITÓRIA NUNES PACHECO[2], LEANDRO DE SOUZA RODRIGUES[3], AND RAFAEL DE MEDEIROS MARIZ CAPUANO[4]

[1]Centro de Tecnologia, UFRN, Avenida Senador Salgado Filho, 3000, Natal, RN, Brasil
[2]Centro de Tecnologia, UFRN, Avenida Senador Salgado Filho, 3000, Natal, RN, Brasil
[3]Centro de Tecnologia, UFRN, Avenida Senador Salgado Filho, 3000, Natal, RN, Brasil
[4]Centro de Tecnologia, UFRN, Avenida Senador Salgado Filho, 3000, Natal, RN, Brasil

Compiled April 28, 2021

Na maioria das vezes é necessário introduzir no sistema, um controlador que modifica a sua dinâmica, para atender a especificações requeridas. Em um sistema controlado é necessário se ter um equipamento físico, o mais utilizado é o eletrônico, o qual precisa conter uma parte lógica. Tal equipamento, pode ser formado por diferentes combinações de 3 diferentes ações de controle: Ação Proporcional, Ação Integral (ou Integrativa) e Ação Derivativa. Este relatório tem como objetivo simular esse tipo de sistema, utilizando como ferramenta, o programa *Matlab*/*Simulink*, no qual foi possível simular o controle do nível de tanques, utilizando o controlador do tipo *Fuzzy* PI,em que seus parâmetros de entrada são gerados por um algoritmo genético.

http://dx.doi.org/10.1364/ao.XX.XXXXXX

1. INTRODUÇÃO

Os controladores *fuzzy* são uma alternativa aos controladores convencionais. Com o uso de regras e funções de pertinência, características de controladores **fuzzy**, é possível a inclusão de não linearidades nas leis de controle. Controladores *fuzzy* permitem a expressão de ambiguidade do pensamento humano através de regras que transportam o conhecimento especializado de um humano para uma máquina[4].

A sintonia destes controladores é realizada de forma manual através da experiência dos operadores de determinado processo, com o auxílio de gráficos de resposta do sistema, o que por sua vez pode acarretar uma certa demora até que este esteja parametrizado de maneira conveniente. Neste sentido, é válido afirmar que recursos computacionais dotados de alto poder de processamento podem simplificar e agilizar esta tarefa, de modo que entre estes encontram-se os Algoritmos Genéticos[6].

O Algoritmo Genético consiste em uma técnica de otimização inspirada em conceitos existentes no campo da biologia, mais especificamente, na teoria da evolução, proposta por Charles Darwin, onde as melhores soluções para um determinado problema possuem maior capacidade para sobreviverem e estas consequentemente tornam-se o ponto de partida para a busca de soluções ainda melhores [6].

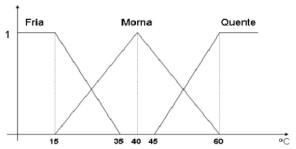
Nesse sentido, este relatório tem como objetivo implementar uma sintonia dos parâmetros de um controlador *Fuzzy* Proporcional Integrativo(PI), este controlador está inserido numa planta de dois tanques. Para isso, foi feito em *Matlab* códigos que implementam a sintonia do controlador *Fuzzy* por algoritmo genético, sendo testado na planta no ambiente **Simulink**, e esse resultado é comparado com sintonizadores manuais

2. ANÁLISES TEÓRICAS RECORRENTES DO PROJETO

A. Sistema Fuzzy

Em um sistema de controle *fuzzy*, a entrada do controlador está em um formato numérico. É necessário traduzi-la para a linguagem de conjuntos *fuzzy*. A etapa de tradução para a linguagem dos conjuntos *fuzzy* é chamada de fuzzificação. Essa etapa mapeia as relações das entradas do controlador com as funções de pertinência predefinidas no projeto do controlador. Nessa etapa é avaliado o grau de pertinência da entrada numérica fornecida[4]. A Figura 1 apresenta um exemplo com a temperatura da água como entrada, percebe-se que de 0 a 35° é considerado frio, de 15° a 60º morna é de 45° em diante é quente, ou seja, a linguagem humana descreve o sistema.

Fig. 1. Exemplo de Fuzzificação da temperatura da água



Fonte: Research gate

O controlador *fuzzy* tem uma base de regras que representa o conhecimento do controlador para a estratégia de controle. É de grande importância que a base de regras mapeie todas as combinações de entradas possíveis garantindo que sempre alguma regra será disparada para qualquer situação. Também é importante não conter contradições entre as regras[4].

Os controladores **fuzzy** descritos na literatura são classificados em função das características gerais do seu método de tomada de decisão. Embora diferentes métodos sejam apresentados na literatura, de acordo com Sugeno (1985) eles podem ser reunidos em dois grandes grupos. O primeiro reúne os que são baseados nas funções de implicação *fuzzy* e em operadores de composição para a definição da saída *fuzzy* do controlador. Já os do segundo grupo dispensam a definição de funções de implicação e operadores para a inferência. Os controladores do tipo Mamdani, referidos nesta seção como controlador de Mamdani, são baseados no primeiro grupo, e os controladores do tipo desenvolvido por Takagi e Sugeno, aqui referidos como controlador de Sugeno, fazem parte do segundo grupo[5]. A tabela abaixo mostra as vantagens dos dois métodos e no anexo A e B mostra uma ilustração do funcionamento dos dois métodos.

Table 1. Comparativo de Vantagens do Mamdani e Sugeno

Mamdani	Sugeno
Intuitivo	Computacionalmente eficiente
Adequado para entrada humana	Funciona bem com técnicas lineares, como controle PID
Base de regra mais interpretável	Funciona bem com técnicas adaptativas e de otimização
Têm ampla aceitação	Garantir a continuidade da superfície de saída
	Adequado para análise matemática

Fonte: Matlab

B. Controlador Fuzzy PI

Segundo Simões e Shaw, seja de a derivação do erro, **du** a variação da saída **e** e o erro, No domínio do tempo tem:

$du = K_p e + K_i de$ [1]

A definição da regra *Fuzzy* é:

SE erro $=E_i$ **E** variação do erro dE_i **ENTÃO**, variação do controle dU_i.

Esse controlador *Fuzzy* tem duas entradas: erro e sua primeira derivada. Nota que a expressão da variação do controle deve ser integrada para ser usada para controlar um processo.

C. Avaliação de desempenho

A finalidade do controlador é assegurar que o comportamento desejado do sistema a ser controlado se ajusta tanto quanto for possível aos requisitos previamente estabelecidos, ou seja, o controlador guia o sistema para seguir um comportamento determinado previamente [4]. O índice de Goodhart considera o esforço de controle, a variância do sinal de controle, e os desvios em relação ao setpoint(referência). Abaixo pode ver as expressões do esforço do controle médio [2], variância do sinal de controle [3] e média do erro quadrático [4]:

$\epsilon_1 = (1/N)\sum_{K=1}^{N} u(k)$ [2]

$\epsilon_2 = (1/N)\sum_{K=1}^{N} (u(k) - \epsilon_1)^2$ [3]

$\epsilon_3 = (1/N)\sum_{K=1}^{N} (r(k) - y(k))$ [4]

$\epsilon = a_1\epsilon_1 + a_2\epsilon_2 + a_3\epsilon_3$ [5]

Os fatores de ponderação atribuídos a cada parcela são a1, a2 e a3. São valores reais entre 0 e 1

3. OTIMIZAÇÃO NUMÉRICA

A. Algoritmo Genético

Algoritmos genéticos são um tipo de algoritmo de otimização, o que significa que são usados para encontrar a(s) solução(ões) ótima(s) para um determinado problema computacional que maximiza ou minimiza uma função particular. Os algoritmos genéticos representam um ramo do campo de estudo denominado computação evolutiva, na medida em que imitam os processos biológicos de reprodução e seleção natural para resolver as soluções "mais adequadas" . Como na evolução, muitos de um dos processos do algoritmo genético são aleatórios, no entanto, esta técnica de otimização permite definir o nível de randomização e o nível de controle . Esses algoritmos são muito mais poderosos e eficientes do que a pesquisa aleatória e algoritmos de pesquisa exaustiva e ainda exigem nenhuma informação extra sobre o problema fornecido. Este recurso permite que eles encontrem soluções a problemas que outros métodos de otimização não podem resolver devido à falta de continuidade, derivadas, linearidade ou outros recursos[3].

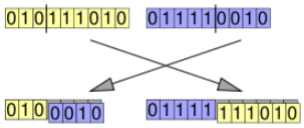
Uma vez que os algoritmos genéticos são projetados para simular um processo biológico, muitos dos a terminologia é emprestada da biologia. No entanto, as entidades a que esta terminologia se refere em algoritmos genéticos são muito mais simples do que suas contrapartes biológicas . O básico componentes comuns a quase todos os algoritmos genéticos são[3]:

- uma função de "fitness"(*emportugus* : *aptido*) para otimização
- uma população de cromossomos
- seleção de quais cromossomos irão se reproduzir
- crossover para produzir a próxima geração de cromossomos
- mutação aleatória de cromossomos na nova geração

A função de aptidão é a função que o algoritmo está tentando otimizar . A palavra "Aptidão" é tirada da teoria da evolução. É usado aqui porque os testes de função de aptidão e quantificam o quão 'adequado' cada solução potencial é. O termo cromossomo refere-se a um valor ou valores numéricos que representam uma solução candidata ao problema que o algoritmo genético está tentando resolver [8]. Cada solução candidata é codificada como uma matriz de valores de parâmetro, um processo que também é encontrado em outra otimização algoritmos [3].

Operador de Recombinação, também chamado de cruzamento ou *crossover*, é um operador que simula a troca genética entre os indivíduos escolhidos. Através da aplicação do operador de recombinação surgem outros indivíduos com características dos pais. O objetivo desta aplicação é a troca de informação entre diferentes soluções candidatas [Goldberg 1989]. A figura 2 mostra como é essa operação, em amarelo é o Pai1 e em azul o Pai2, é feito um corte no cromossomo dois pais e recombinados.

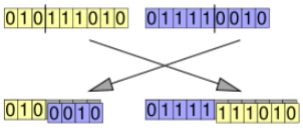
Fig. 2. Operador de recombinação



Fonte:Wikipédia

O operador de mutação altera aleatoriamente genes do indivíduo. Com o passar das gerações, a população tende a convergir para uma certa região do espaço de busca. A mutação permite explorar outras possibilidades no espaço de busca. Na maioria das vezes a mutação produz um indivíduo ruim, mas isso é necessário para explorar novas regiões do espaço de busca, o que permite fugir de um máximo ou mínimo local[4]. Abaixo, na figura 3, mostra uma mutação pontual, onde uma posição do cromossomo é trocada.

Fig. 3. operador de mutação



Fonte:Arxiv

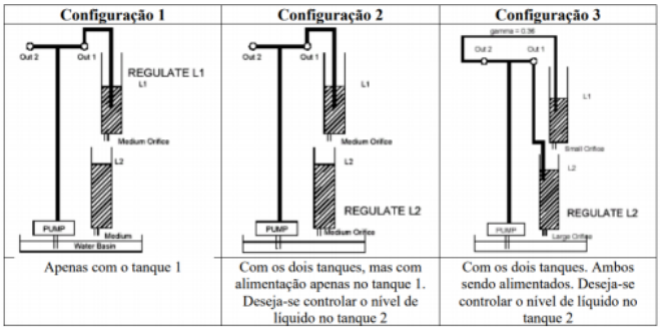
B. METODOLOGIA E PROJETO

controlador *Fuzzy* está inserido numa planta que contém dois tanques em cascata como mostra o tópico C, a planta foi implementada no *Simulink*. O detalhes do controlador *Fuzzy* é mostrado no tópico D, a implementação se deu na ferramenta *Fuzzy* do *matlab*, o qual gera um arquivo .fis. Depois que é obtido um modelo inicial do controlador *fuzzy* é possível otimizar com sintonizador por algoritmo genético. O sintonizador é descrito no tópico 3.3, esse foi obtido no **script** do *Matlab*

C. Planta do controlador

A configuração do Sistema de Dois Tanques adotada para este trabalho resulta na alimentação do primeiro tanque por meio da bomba e o segundo tanque alimentado somente pelo fluxo de saída do primeiro tanque como pode ser visualizado na Figura 4. L1 e L2 representam os níveis de água do primeiro e do segundo tanque, respectivamente. A variável a ser controlada é o nível de água do primeiro e segundo tanque, ou seja, L1 e L2.

Fig. 4. Configuração física do sistema de dois tanques



Fonte: Disponibilizada pelo professor

Onde a vazão do tanque 1[6] e tanque 2 [7] pode ser modelado por:

$F_1 = a_1\sqrt{2gL_1}$ [6]

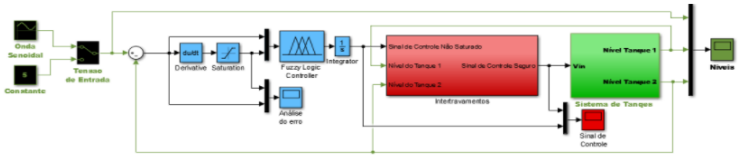
$F_2 = a_2\sqrt{2gL_2}$ [7]

em que a_x= área do orifício de saída x e L_x = nível do tanque x.

A taxa de variação do nível do tanque 1 é obtida dividindo-se a taxa de variação volumétrica pela área da base do tanque 1 . Sendo que a variação volumétrica no tanque 1 é dada pela diferença entre as vazões de entrada e de saída. A taxa de variação do nível do tanque 2 , assim como no caso do tanque 1, é obtida dividindo-se a taxa de variação volumétrica pela área da base do tanque.

A planta do tanque de nível junto com o controle PI pode ser vista na figura abaixo(figura 5). Detalhes do intertravamento e sistema de tanques pode ser visto no anexo C e D, O bloco de **fuzzy** será obtido com auxílio do sintonizador que será detalhado nos próximos tópicos.

Fig. 5. Planta do sistema dois tanques no ambiente Simulink



Fonte: Disponibilizada pelo professor

D. Sistema fuzzy projetado

O programa *Matlab* dispõe de uma função, em que é possível projetar o sistema *Fuzzy* desejado.

O sistema *fuzzy* projetado neste trabalho, apresenta duas variáveis de entrada, que são o erro e a variação do erro, ilustradona Figura 6 como, erro e varErro. A partir dessas entradas obtém-se uma saída vinculada à um sinal de controle pré-estabelecido.

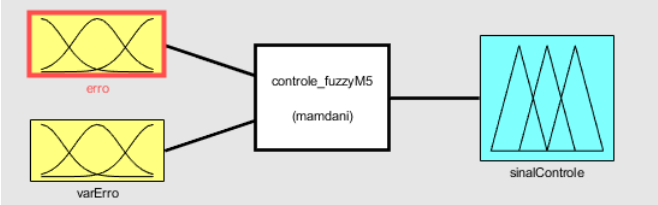


Fig. 6. Modelo do controlador Fuzzy no Simulink

E. Modelo Sugeno

Para as funções de pertinência do modelo Sugeno foram construídas duas variáveis linguísticas, sendo elas: variação do erro (*var_erro*) e o próprio erro (*erro*).

A variável *var_erro* informa para o controlador se o sistema está se aproximando ou se distanciando do referencial a ser seguido. Desta forma, uma variação positiva significa que o erro está se distanciando para um valor maior que o erro zero.

Nas Figuras 10 e 8 podemos verificar os pontos de início e fim de cada uma das funções de pertinência. Para esta variável analisada o grupo decidiu usar apenas três MF's já que os valores máximos e mínimos que essa variável pode assumir é 0.3 e -0.3 respectivamente, de forma que essa quantidade de regras já descreveriam o sistema suficientemente bem. Essas MF's são: variação negativa (VN), variação zero (VZ) e variação positiva (VP).

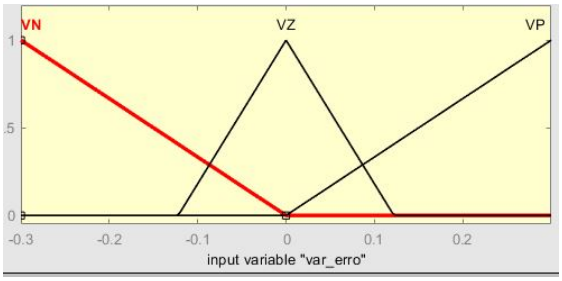


Fig. 7. Funções de pertinência modelo Sugeno, para a entrada da variação do erro

Nome	Ponto inicial	Ponto máximo	Ponto final
Variação negativa	-0.3	-0.3	0
Variação zero	-0.122	0	0.122
Variação positiva	0	0.3	0.3

Fig. 8. Exemplo de intervalo de valores para as funções de pertinência da variação do erro

Já a variável *erro* indica apenas a diferença entre o referencial e o valor lido no nível do tanque. Um erro positivo indica que o tanque está com um nível inferior a referência.

Para esta variável foram usadas cinco funções de pertinência, visto que o universo de valores que ela pode assumir é 100 vezes maior que *var_erro*, podendo assumir valores entre -30 e 30. Para descrever então foram usadas as funções de pertinência: negativo grande (NG), negativo pequeno (NP), erro zero (EZ), positivo pequeno (PP) e positivo grande (PG).

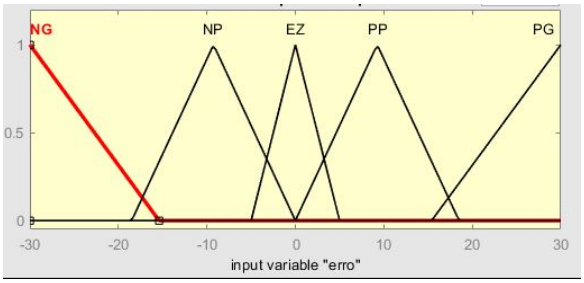


Fig. 9. Funções de pertinência modelo Sugeno, para a entrada do erro

Nome	Ponto inicial	Ponto máximo	Ponto final
Negativo grande	-30	-30	-15.4
Negativo pequeno	-18.5	--9.25	0
Erro zero	-5	0	5
Positivo pequeno	0	9.25	18.5
Positivo grande	15.4	30	30

Fig. 10. Exemplo de intervalo e valores, para as funções de pertinência do erro

Desta forma, é formada uma base com 15 regras no total, como mostradas a seguir.

1. **SE** (*erro* é *NG*) *and* (*var_erro* é *VN*) **ENTÃO** (*out_fuzzy* é *DUM*)
2. **SE** (*erro* é *NG*) *and* (*var_erro* é *VZ*) **ENTÃO** (*out_fuzzy* é *DUG*)
3. **SE** (*erro* é *NG*) *and* (*var_erro* é *VP*) **ENTÃO** (*out_fuzzy* é *DUG*)
4. **SE** (*erro* é *EZ*) *and* (*var_erro* é *VZ*) **ENTÃO** (*out_fuzzy* é *DUP*)
5. **SE** (*erro* é *EZ*) *and* (*var_erro* é *VP*) **ENTÃO** (*out_fuzzy* é *DUM*)
6. **SE** (*erro* é *EZ*) *and* (*var_erro* é *VN*) **ENTÃO** (*out_fuzzy* é *DUM*)
7. **SE** (*erro* é *PG*) *and* (*var_erro* é *VZ*) **ENTÃO** (*out_fuzzy* é *DUG*)
8. **SE** (*erro* é *PG*) *and* (*var_erro* é *VP*) **ENTÃO** (*out_fuzzy* é *DUM*)
9. **SE** (*erro* é *PG*) *and* (*var_erro* é *VN*) **ENTÃO** (*out_fuzzy* é *DUG*)
10. **SE** (*erro* é *NP*) *and* (*var_erro* é *VN*) **ENTÃO** (*out_fuzzy* é *DUM*)
11. **SE** (*erro* é *NP*) *and* (*var_erro* é *VP*) **ENTÃO** (*out_fuzzy* é *DUP*)
12. **SE** (*erro* é *NP*) *and* (*var_erro* é *VZ*) **ENTÃO** (*out_fuzzy* é *DUP*)
13. **SE** (*erro* é *PP*) *and* (*var_erro* é *VN*) **ENTÃO** (*out_fuzzy* é *DUP*)
14. **SE** (*erro* é *PP*) *and* (*var_erro* é *VP*) **ENTÃO** (*out_fuzzy* é *DUM*)
15. **SE** (*erro* é *PP*) *and* (*var_erro* é *VZ*) **ENTÃO** (*out_fuzzy* é *DUP*)

F. Modelo Mamdani

Para a simulação do modelo Mamdani, diferentemente do Sugeno optou-se por utilizar 3 funções de pertinência para a entrada do erro e uma segunda com 3 para a entrada da variação do erro. Foi testado inicialmente 6 funções de pertinência para o erro, porém apesar da simplificação obtiveram-se resultados melhores. A base então foi configurada com 9 regras no total.

Em relação as entradas Erro e variação do erro, as funções de pertinência foram EN (erro negativo), EZ (Erro em zero), EP (Erro positivo), VN(variação do erro negativa), EZ (variação do erro em zero) e EP (variação do erro positiva). Na saída foram utilizadas as seguintes funções: DUNP(saída negativa pequena), DUNG(saída negativa grande), DUZ (saída em zero), DUPP (saída positiva pequena) e DUPG (saída positiva grande).

Nos exemplos de simulações as funções de pertinência de entrada são semelhantes, dessa forma, o que mais manipulou-se para obtenção da sitônia, foi o intervalo estabelecido para a variável de saída.

Observando que com o aumento da parte negativa do intervalo da variável de saída, o nível descia e vice-versa, pôde-se obter melhorias na sitônia do sistema, encontrando por meio de testes, valores para o intervalo da variável, os quais aproximavam-se do referencial.

Abaixo podemos ver cada uma das MF's nas Figuras 11 e 13 assim como seus limites nas Figuras 12 e 14.

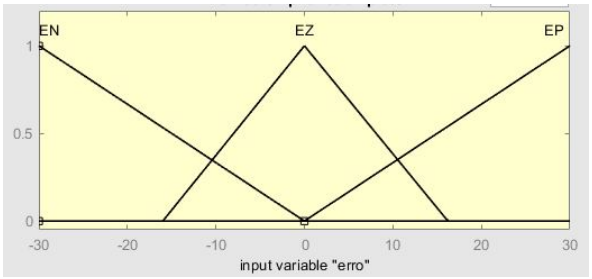


Fig. 11. Exemplo de funções de pertinência modelo Mamdani, para a entrada do erro

Nome	Ponto inicial	Ponto máximo	Ponto final
Erro negativo	-30	-30	0
Erro zero	-16	0	16
Erro positivo	0	30	30

Fig. 12. Exemplo de intervalos de valores para as funções de pertinência no modelo Mamdani

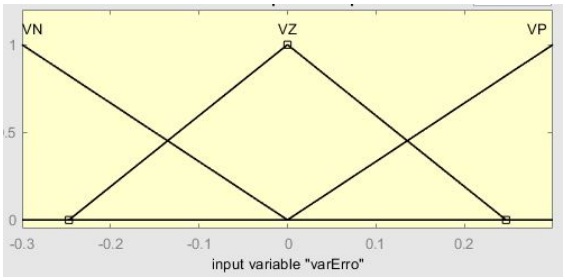


Fig. 13. Exemplo de funções de pertinência no modelo Mamdani, para a entrada da variação do erro

Nome	Ponto inicial	Ponto máximo	Ponto final
Variação negativa	-0.3	-0.3	0
Variação zero	-0.247	0	0.347
Variação positiva	0	0.3	0.3

Fig. 14. Exemplo de intervalos de valores, para as funções de pertinência da variação do erro no modelo Mamdani

Na lista abaixo pode-se ver as 9 regras presentes na base do controlador Mamdani.

1. **SE** (*erro* é *EN*) *and* (*var_erro* é *VN*) **ENTÃO** (*out_fuzzy* é *DUNG*)
2. **SE** (*erro* é *EN*) *and* (*var_erro* é *VZ*) **ENTÃO** (*out_fuzzy* é *DUNP*)
3. **SE** (*erro* é *EN*) *and* (*var_erro* é *VP*) **ENTÃO** (*out_fuzzy* é *DUPG*)
4. **SE** (*erro* é *EZ*) *and* (*var_erro* é *VN*) **ENTÃO** (*out_fuzzy* é *DUZ*)
5. **SE** (*erro* é *EZ*) *and* (*var_erro* é *VZ*) **ENTÃO** (*out_fuzzy* é *DUZ*)
6. **SE** (*erro* é *EZ*) *and* (*var_erro* é *VP*) **ENTÃO** (*out_fuzzy* é *DUPG*)
7. **SE** (*erro* é *EP*) *and* (*var_erro* é *VN*) **ENTÃO** (*out_fuzzy* é *DUPG*)
8. **SE** (*erro* é *EP*) *and* (*var_erro* é *VZ*) **ENTÃO** (*out_fuzzy* é *DUPP*)
9. **SE** (*erro* é *EP*) *and* (*var_erro* é *VP*) **ENTÃO** (*out_fuzzy* é *DUPG*)

O sinal de controle referente ao Mandani desta vez se da na forma de valores fuzzyficados, em forma de funções de pertencimento. Desta forma temos os seguintes possíveis valores.

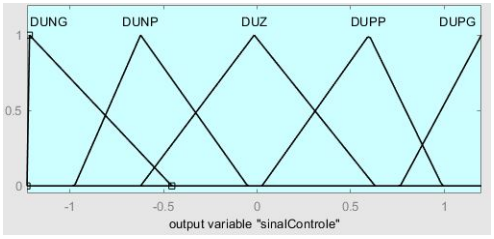


Fig. 15. Sinal de controle fuzzyficado do Mandani

Nome	Ponto inicial	Ponto máximo	Ponto final
DUNG	-1.23	-1.217	-0.4553
DUNP	-0.9762	-0.6225	-0.05035
DUZ	-0.6225	0	0.6225
DUPP	0.02676	0.599	0.9911
DUPG	0.7662	1.2	1.2

Fig. 16. Exemplo de valores limites dos sinais de controle fuzzyficados do Mandani

4. CÓDIGOS DO ALGORITMO GENÉTICO

Neste tópico é explicado as principais funções do algoritmo genético, todas as funções se encontram no anexo (?? pode ser um link do github tambem).

Segundo a documentação do Matlab, os passos do algortimo genético são:

1. O algoritmo começa criando uma população inicial aleatória.
2. O algoritmo então cria uma sequência de novas populações. Em cada etapa, o algoritmo usa os indivíduos da geração atual para criar a próxima população. Para criar a nova população, o algoritmo executa as seguintes etapas:

a) Pontua cada membro da população atual calculando seu valor de aptidão

b) Dimensiona as pontuações de aptidão para convertê-las em uma faixa de valores mais utilizável.

c) Seleciona membros, chamados de pais

d) Alguns dos indivíduos da população atual que apresentam maior aptidão física são escolhidos como elite. Esses indivíduos da elite são passados automaticamente para a próxima população.

e) Produz filhos dos pais. Os filhos são produzidos por meio de mudanças aleatórias em um único pai - mutação - ou pela combinação das entradas de vetor de um par de pais - crossover .

f) Substitui a população atual pelos filhos para formar a próxima geração.

3. O algoritmo para quando um dos critérios de parada é atendido

A. Gerando cromossomos dos indivíduos

Fig. 17. função que gera um indivíduo

```
function novo_cromo=criar_cromossomo()
    %Criar erro de -30 a 30
    for i=1:15
        tempo1(i)=(60*rand)-30;
    end
    tempo1=sort(tempo1);
    %Criar var_erro de -0.3 a 0.3
    for i=1:9
        tempo2(i)=(0.6*rand)-0.3;
        tempo2=sort(tempo2);
    end
    tempo2=sort(tempo2);
    %Criar regras com valores 1, 2 ou 3
    for i=1:15
        tempo3(i)=ceil(3*rand);
    end
    finalesco=[0.01 3 0.01 3.5 0.1 5];
    novo_cromo=[tempo1 tempo2 tempo3 finalesco];
end
```

Fonte:Autoria própria

A função “novo_cromo”, gera um cromossomo, ou seja, um conjunto de valores aleatórios chamados de “genes”,de forma análoga a seu homônimo biológico, o gene representa uma característica do indivíduo,a ser avaliada,como o cromossomo possui as variáveis “erro”,”var_erro” e as regras do sistema difuso,para gerar, foi necessário colocar os limites entre do erro entre -30 e 30 no primeiro laço,colocar a variação do erro entre -0.3 e 0.3,além de criar as regras com valores no último laço.

Como os cromossomos são aleatórios, apenas com os limites estabelecidos, foi usada a função “rand”que gera números aleatórios para um vetor de tamanho pré estabelecido.

B. Classificação da população

Fig. 18. função que classifica o indivíduo

```
function [pop_rankeado, elites, notas]=ranked(populacao,tamanho_pop, qntd_elite)
    for i=1:45
        %Para cada individuo gera um fis, simula, da a nota
        AGfiss(populacao(i,:));
        sim('Simulacao_Exemplo_04_Fuzzy_v2');
        notas(i)=goodhart(Sinal_Controle_U(:,2),15,Niveis(:,4));
    end
    temp=[populacao notas']; %concatena a população c a nota de cada individuo
    %Rankeia de maneira crescente (menor nota em cima pois é melhor)
    temp_sorted=sortrows(temp,tamanho_individuo+1,{'ascend'});
    elites=temp_sorted(1:qntd_elite,1:tamanho_individuo); %Separa os da elite
    pop_rankeado=temp_sorted; %vetor população+nota rankeado
    notas=pop_rankeado(:,tamanho_individuo+1); %vetor coluna só de notas
end
```

Fonte:Autoria própria

A lógica para classificar utiliza o já citado método de goodhart para "rankear" as populações, basicamente, a função "AGfiss" envia os dados dos indivíduos em formato ".fis" para o modelo feito através do *simulink*, este que recebe e os utiliza como entradas do sistema.Os resultados então são rankeados através da função "goodhart",que usa os valores dos sinal de controle e dos níveis como parâmetros. O código faz o "rankeamento" de forma crescente e os apresenta na forma de colunas.

C. Geração de novo indivíduo - operador de recombinação e mutação

Fig. 19. função que classifica o indivíduo

```
function filho=gerar_novo_individuo(pai1, pai2, pai3, tamanho_individuo)
    filho=[pai1(1,1:13) pai2(1,14:27) pai3(1,28:39) 0.01 3 0.01 3.5 0.1 5];
    chance_wolwerine=rand;
    if chance_wolwerine<=0.05
        pos_mut=ceil(39*rand); %arredonda pra cima pra nunca ser zero, só pos
        if (pos_mut>=1)&&(pos_mut<=15)
            filho(pos_mut)=(60*rand)-30;
        elseif (pos_mut>=16)&&(pos_mut<=24)
            filho(pos_mut)=(0.6*rand)-0.3;
        else
            filho(pos_mut)=ceil(3*rand);
        end
    end
    tempo1=sort(filho(1:15));
    tempo2=sort(filho(16:24));
    filho=[tempo1 tempo2 filho(25:45)];
end
```

Fonte:Autoria própria

Nesta função aplicada ao Sugeno temos o objetivo de gerar os indivíduos "filhos" da geração anterior. Para o Sugeno, com uma grande quantidade de cromossomos que ultrapassavam os limites do tanque e a demora da simulação, mantivemos em blocos os pontos referentes as variáveis de erro e variação do erro, com isso o filho teria uma variável totalmente igual a do pai. Com isso, reduzimos consideravelmente as probabilidades de novas amostras, mas conseguimos reduzir um pouco a quantidade de gerações necessárias.

Já para o Mandani utilizamos um cruzamento um pouco maior, de tal forma que os genes de um pai eram agrupados em blocos de apenas três valores, como cada variável era representada por 9 ou 15 valores, foi possível obter uma diversidade bem maior, custando mais gerações para se achar um ótimo entre a população original.

Apesar dessas diferenças, mantivemos as regras fixas em todos os cromossomos gerados neste trabalho, tendo em vista que grande parte das novas regras geradas tornariam o cromossomo muito ruim e sem sentido algum quando usando variáveis linguisticas, visto que poderia ser gerada uma regra como "se erro grande e variando muito positivamente saída grande".

5. RESULTADOS

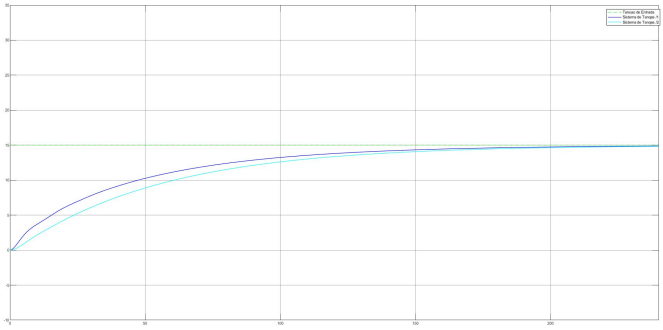
A. Sugeno

Após gerar a população inicial de maneira randômica verificamos que existiam três grandes classes de indivíduos: os que tinham um *overshoot* muito grande e faziam o tanque vazar; os que tinham uma resposta muito lenta e demoravam mais do que o tempo de simulação para chegar ao *setpoint*; e por fim os que logo convergiam para o *setpoint* de maneira suave e estabilizava.

Os cromossomos "divergentes" eram logo eliminados das simulações, pois recebiam forçadamente uma nota mais elevada que os demais. Com uma população inicial de 60 indivíduos após 14 gerações numa simulação de 5h30 a nota do melhor individuo já convergia para um valor fixo, parando então o processo.

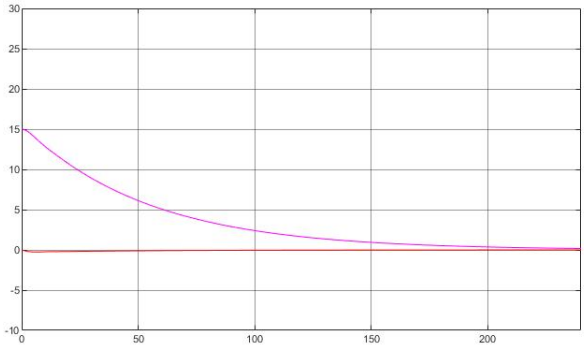
Na Figura 20 podemos ver como o nível 2 se aproxima do objetivo de 15cm de maneira suave, sem *overshoot* e erro ao final (comprovando na Figura 21).

Fig. 20. Melhor comportamento no Sugeno



Fonte:Autoria própria

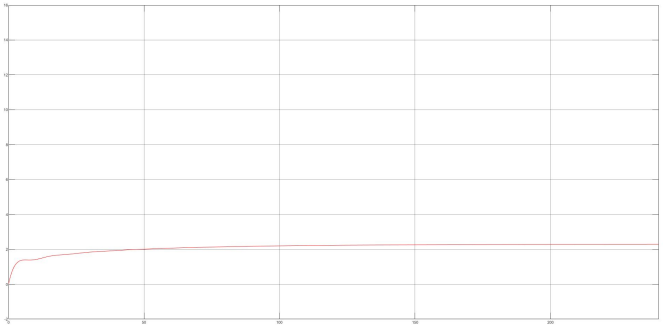
Fig. 21. Erro do melhor cromossomo do Sugeno



Fonte:Autoria própria

Podemos ainda verificar a boa aplicabilidade desse individuo num sistema real percebendo que o sinal de controle não tem grande variação, se mantendo estável, o que é positivo para durabilidade e precisão de sistemas mecânicos como a bomba a qual estamos simulando.

Fig. 22. Sinal de controle do melhor cromossomo do Sugeno



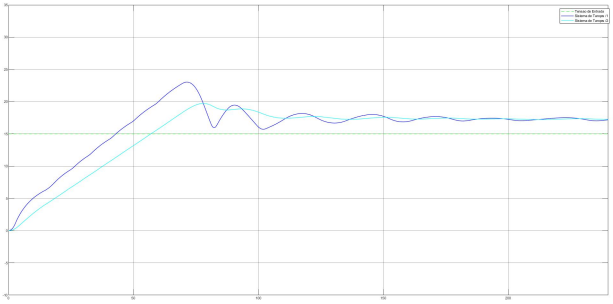
Fonte:Autoria própria

B. Mandani

Nas simulações referentes ao Mandani já percebemos uma instabilidade maior entre nossos cromossomos, poucos indivíduos conseguiam zerar o erro e nunca tinham um comportamento tão suave como seus pares do Sugeno, apresentando sempre um *overshoot* considerável. Com isso, após 30 gerações ao longo de 13h a simulação foi parada.

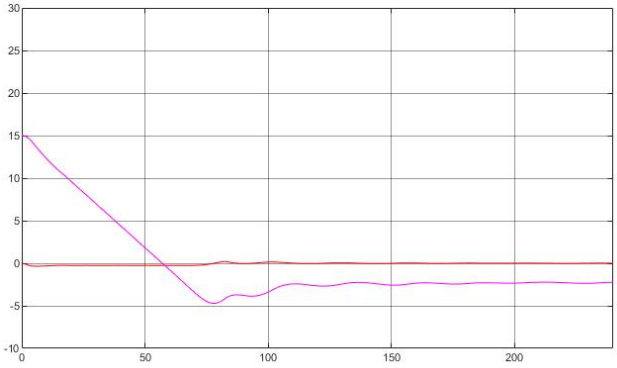
Na Figura 23 podemos ver o comportamento simulado do controlador *fuzzy* Mandani obtido ao final do processo. Apesar de não conseguir zerar o erro, o qual final ficou a cerca de 16% do objetivo (Figura **??**), foi possível obter um sinal de controle relativamente estável (Figura 25), sem grandes variações, o que é saudável para a bomba.

Fig. 23. Melhor comportamento no Mandani



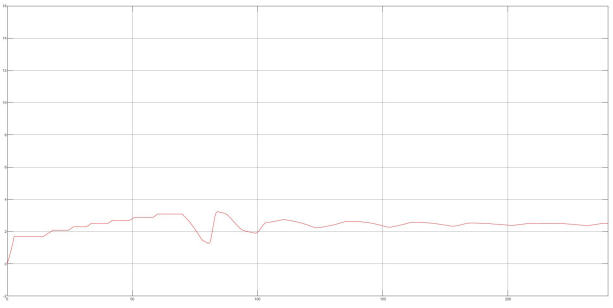
Fonte:Autoria própria

Fig. 24. Erro do melhor cromossomo do Mandani



Fonte:Autoria própria

Fig. 25. Erro do melhor cromossomo do Mandani



Fonte:Autoria própria

6. CONCLUSÃO

Os problemas de engenharia envolvem sistemas de difícil modelagem e sistemas altamente não lineares, o qual o controlador Fuzzy se mostra uma boa opção para suas soluções. A grande dificuldade desses controladores é em sintonizar seus parâmetros, o tornar otimizados. O algoritmo genético pode ser usado como um método otimizador pois não precisa de uma função diferenciável e continua que descreva o sistema, o qual é o caso do controlador PI Fuzzy.

Neste trabalho conseguimos atingir o objetivo de maneira satisfatória, entretanto temos a ciência que algumas melhorias ainda poderiam ser feitas no projeto. No Sugeno, os resultados mostraram que foi encontrado o melhor individuo, mesmo que um tempo de estabilização um pouco longo; no Mandani o erro não foi zero, o qual é o maior objetivo do controlador PI. Simulações demoradas trouxeram problemas para fazer os ajustes finos do próprio código como uma escolha melhor do método de avaliação, o qual poderíamos combinar o Goodhart a outros que medissem outras informações. Além disso poderíamos testar com mais precisão os dois métodos de cruzamento usados e identificar o melhor. Paralelamente, as simulações poderiam também ter sido otimizadas modificando um pouco a planta do simulink, travando a simulação logo que identificasse a divergência do controlador, visto que nesses casos as simulações individuais dos cromossomos demoravam mais do que o convencional.

7. REFERÊNCIAS

[1] SHAW, Ian. SIMÕES, Marcelo. Controle e modelagem Fuzzy, 1999. Editora Fapesp 2º ed.

[2] BODENHOFER, Ulrich. Tuning of Fuzzy Systems Using Genetic Algorithms, 1996. Institut für Mathematik.

[3] CARR, Jenna. An Introduction to Genetic Algorithm, 2014.

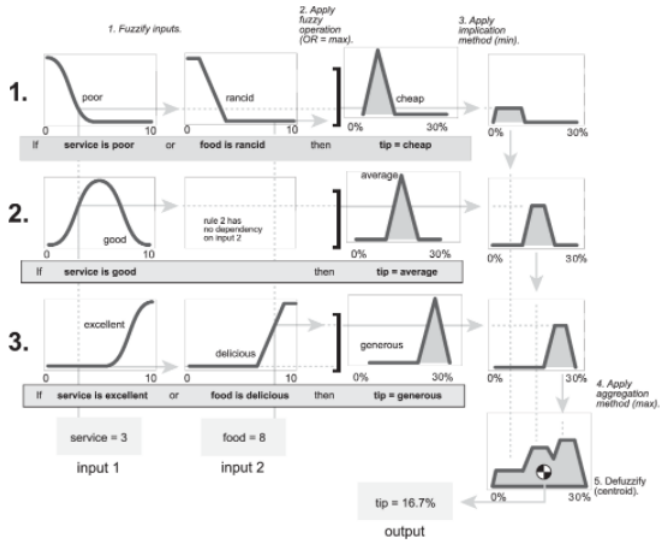
[4] SANTOS, Fernanda. Algoritmo Genético para Otimização da Sintonia de um Controlador Fuzzy, 2012. Universidade Federal do Rio Grande do Norte.

[5] ANDRADE, Michelle; JACQUES, Maria. ESTUDO COMPARATIVO DE CONTROLADORES DE MAMDANI E SUGENO PARA CONTROLE DO TRÁFEGO EM INTERSEÇÕES ISOLADAS. Universidade de Brasília.

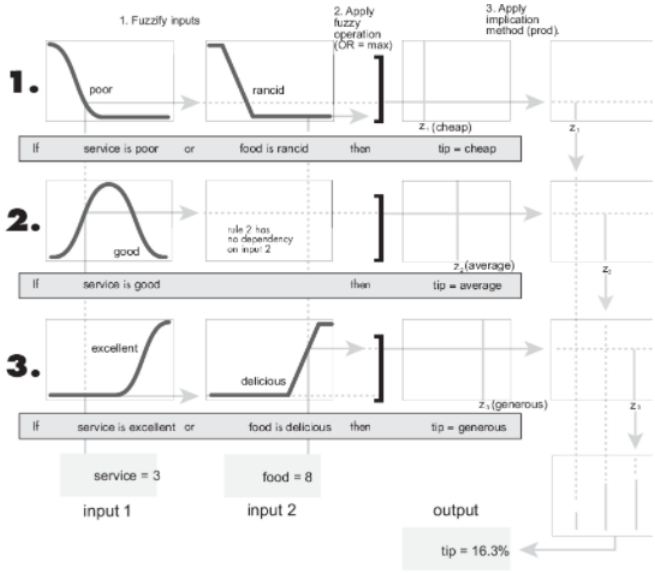
[6] MADEIRA, Daniel. Utilização de algoritmos genéticos para sintonia de controladores PID, 2016.Disponível em: <https://www.embarcados.com.br/algoritmos-geneticos-e-controladores-pid/>

[7] MENEGHETTI UGULINO DE ARAÚJO,F. Controle Inteligente. Natal, Rio Grande do Norte, 2015. (Apostila)

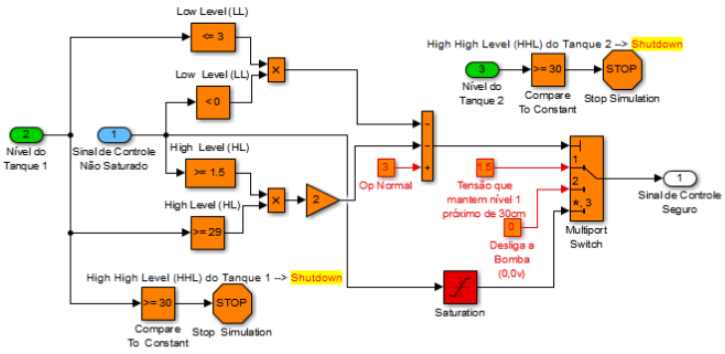
ANEXO A



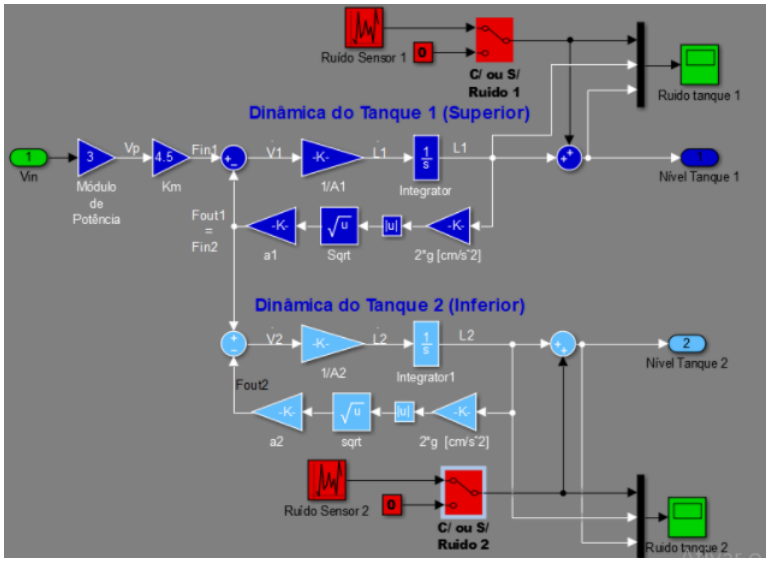
ANEXO B



ANEXO C



ANEXO D



ANEXO E
link dos códigos: https://github.com/leandrolrs18/AlgoritmoGeneticoControleFuzzy