UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

Controle Inteligente: Apoio técnico

Desenvolvedor:
Engº M.Sc. Jean Mário Moreira de Lima

Janeiro
2021

Sumário

1 Introdução

Nas disciplinas de Controle Inteligente (códigos), os alunos de graduação e de pós-graduação são expostos a conceitos de inteligência artifical (IA) e incentivados a aplicar técnicas desse ramo da ciência em sistemas de controle. Basicamente, implementam controladores baseados em lógica nebulosa (*fuzzy*) e otimizam tais controladores utilizando meta-heurísticas bioinspiradas para a planta base do laboratório de sistemas de controle CTEC-217: o sistemas de tanques acoplados Quanser. Além disso, há também a aplicação de redes neurais artificiais na identificação da planta citada.

Figura 1: *Ilustração tanques acoplados Quanser.*

Com o intuito de apoiar tecnicamente as atividades realizadas por alunos de graduação e pós-graduação, e contribuir na formação dos mesmos, esse material foi produzido e está dividido da seguinte maneira. A seção 1 contém instruções técnicas para o uso adequada da *toolbox* Fuzzy, introduzindo a ferramenta e descrevendo suas funcionalidades: entradas, saídas, funções de pertinência, base de regas e exportação do arquivo *.fis*. Como é possível utilizar o referido *toolbox* e configurá-lo via código, também há instruções para executar tal tarefa. Na seção 2 desse material, há instruções de como uma otimização baseada em meta-heurística pode ser aplicada para sintonizar um controlador fuzzy com seus diversos parâmetros. Informações sobre o conjunto de parâmetros de um controlador fuzzy, a combinação dos mesmos, das funções de pertinência, e pseudo-código de como gerenciar tais parâmetros em tempo de execução de uma meta-heurística. A seção 3 contém contempla o desenvolvimento das redes neurais PMC, em Python

e MATLAB, para identificação de sistemas dinâmicos: ferramentas, tratamento dos dados, algoritmo de treinamento backpropagation, biblioteca para treinamento (*scikit-learn* e *Keras*) e testes.

2 Controladores Nebulosos

Operadores humanos apresentam a habilidade de trabalhar com processos industriais de dinâmica não completamente conhecida. Através da experiência, tais operadores sabem que ação tomar diante de determinadas condições e eventos como, por exemplo, combinação de leitura de instrumentos de medição, sinais sonoros e etc. A lógica nebulosa (*fuzzy*) é capaz de expressar de maneira sistemática quantidades imprecisas, vagas e mal definidas. Por exemplo, controladores industriais nebulosos, baseados em lógica nebulosa, são capazes de empregar o conhecimento de operadores humanos adquirido experimentalmente. Assim, a ação de controle gerada por um controlador nebuloso pode ser tão eficiente quanto a dos operadores e sempre consistente.

Os controladores nebulosos apresentam como maior vantagem a utilização de regras heurísticas na obtenção de estratégias de controle utilizadas por operadores humanos, inserindo imprecisões na descrição dos processos controlados. As regras nebulosas podem ser utilizadas no controle direto, na supervisão e na assistência ao operador humano. O material didático CONTROLE INTELIGENTE desenvolvido por Araújo (2015) traz explicações mais detalhadas sobre a construção de controladores nebulosos voltados ao controle direto. Nesse material, o foco será o de projeto do controlador nebuloso na prática e a ferramenta utilizada para tal, o *fuzzy toolbox* do *MATLAB*.

2.1 Projeto de Controladores Nebulosos na Prática

Controladores nebulosos são os sistemas *fuzzy* de maior praticabilidade e efetivamente aplicados a plantas e processos industriais, cuja implementação pode ser feita tanto em software quando em hardware. O projeto de um sistema nebuloso dispõe de inúmeros graus de liberdade que implicam em grande flexibilidade para escolha dos parâmetros como, por exemplo, as diversas opções de métodos de *fuzzificação* e *defuzzificação*. Esse necessidade de flexibilidade fomenta o uso de um software para implementação do sistema nebuloso, com a implementação em hardware somente ao fim do projeto. Além disso, uma implementação somente em software é factível para maioria dos processos industriais, que são em geral térmicos, mecânicos ou químicos, com constante de tempo lenta. Devido as razões citadas, sistemas de software baseados em lógica nebulosa se tornaram predominantes.

Um controlador nebuloso pode ser implementado em qualquer linguagem de programação, de alto ou de baixo nível. Entretanto, o desenvolvimento requer conhecimento e tempo do projetista na linguagem escolhida. Além disso, toda e qualquer alteração no código deve ser recompilada. Esse ciclo de desenvolvimento pode afetar a sintonia de um controlador nebuloso uma vez que torna-se

difícil processar os efeitos de alterações de regras e funções de pertinência na resposta da planta. Os seguintes pontos são desejáveis no projeto de um controlador nebuloso:

- Redução ou eliminação dos requisitos de programação
- Capacidade de efetuar mudanças rápidas no algoritmo nebuloso em construção
- Disponibilidade de diversas opções e parâmetros que correspondem aos vários graus de liberdade, intrínsecos a um sistema nebuloso
- Projeção visual os efeitos das modificações do controlador nebuloso, de forma a auxiliar o projetista

Dentre os itens citados acima, destaca-se o último ponto listado. Um controlador nebuloso requer um longo intervalo de tempo para sintonia, com diversos ajustes de regras e funções de pertinência. Dessa forma, uma plataforma de desenvolvimento que ofereça ferramentas baseadas em gráficos, que possibilitem alterações rápidas e observações dos efeitos imediatos na resposta da planta, é de grande importância. Dispondo de tal ferramenta, o projetista pode implementar o controlador nebuloso de uma maneira mais intuitiva, robusta e com observação imediata dos efeitos, através da manipulação de padrões visuais. Uma vez que os requisitos de desempenho do controlador nebuloso são satisfeitos, o projetista pode gerar o programa de controle na linguagem que se queira. Como utilizado na disciplina, a próxima seção ilustra a utilização do *Fuzzy Logic Toolbox*, a interface de implementação de controladores fuzzy do *MATLAB*.

2.2 Usando o *Fuzzy Logic Toolbox* no Projeto de Controladores Nebulosos

O *Fuzzy Logic Toolbox* é uma biblioteca do MATLAB que contém uma interface gráfica (GUI) de implementação de sistemas nebulosos. Essa ferramenta especializada contempla todos os pontos desejados em um projeto de controlador nebuloso listados na seção anterior. Além disso, também permite a integração com o ambiente de simulação, *Simulink*. Há cinco interfaces gráficas para projetar um sistema nebuloso no *Fuzzy Logic Toolbox* e todas as cinco estão internamente conectadas, isto é, modificações realizadas em um processo, influencia as outas. As cinco ferramentas GUI são listadas abaixo, sendo a principal a *Fuzzy Inference System* (FIS).

- Fuzzy Inference System ou FIS Editor
- Membership Function Editor
- Rule Editor

- Rule Viewer
- Surface Viewer

A interface FIS pode ser inicializa pela linha de comando do MATLAB, digitando-se "fuzzy"e pressionado a teclar "Enter":

$$>> \text{fuzzy}$$

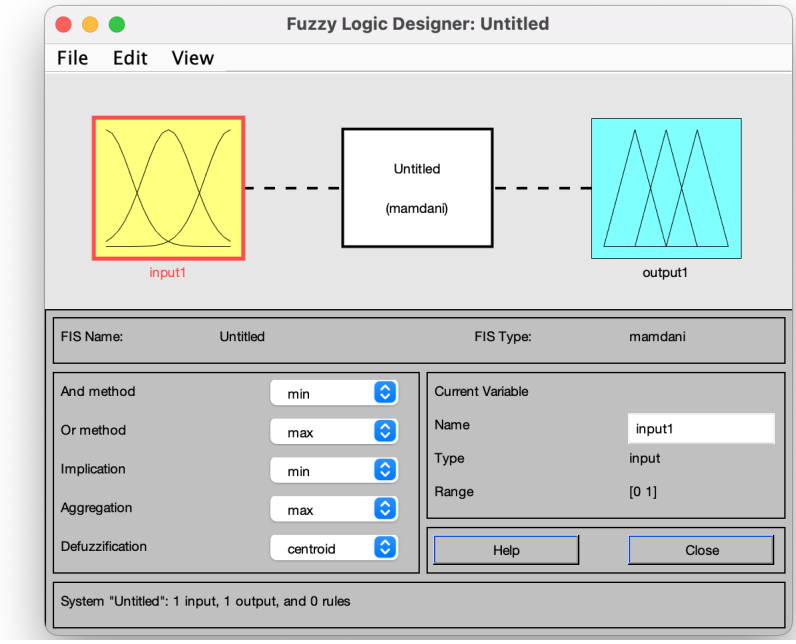


Figura 2: Tela de Edição FIS.

A Figura 2 ilustra a tela básica do FIS. Através da interface FIS, as outras interfaces gráficas podem ser acessadas. Da esquerda para a direita, a Figura 2 ilustra o editor de funções de pertinência da entrada (*Membership Function Editor*), o editor de regras de inferência (*Rule Editor*) e o editor de funções de pertinência de saída (*Membership Function Editor*). Na área abaixo dos GUIs algumas opções ilustram as funções de inferência, o método de defuzzificação e nome de cada variável de entrada ou saída.

2.2.1 Exemplo de Projeto de Controlador Nebuloso PI

Seja de a variação do erro e e du a variação da saída (sinal de controle) u. A expressão que rege a relação entre tais grandezas no domínio do tempo é:

$$du = K_p e + K_I de. \tag{1}$$

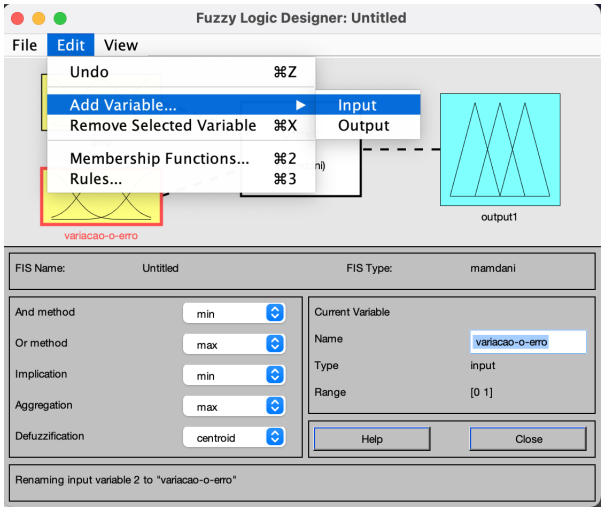
A regra nebulosa pode ser definida da seguinte maneira:

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

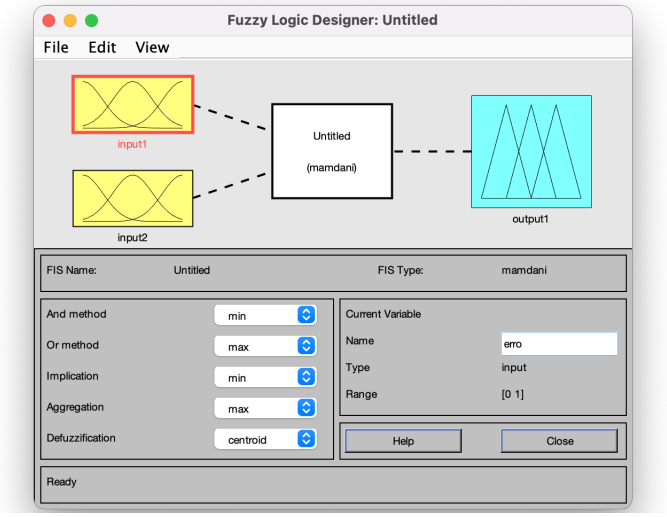
De acordo com a regra nebulosa, duas entradas compõem o controlador nebuloso: o erro e sua primeira derivada. Como saída do controlador nebuloso, tem-se a variação do sinal de controle. Antes de usar o sinal de saída para o controle do processo, é necessário que a expressão da variação de controle seja integrada.

A seguir, indica-se o passo a passo de como proceder para construir o controlador nebuloso PI usando o *Fuzzy Logic Toolbox*. Vale ressaltar que antes dessa etapa, é necessário definir as funções de pertinência de ambas as entradas e, em sequência, a base de regras que estabelece a relação entre as funções de pertinência dos antecedentes e da saída. Neste exemplo adota-se um número de cinco funções de pertinência para ambas as entradas e sete funções de pertinência para a saída. Nas entradas: negativo grande (NG), negativo pequeno (NP), zero (ZE), positivo pequeno (PP), positivo grande (PG). Na saída: negativo grande (NG), negativo médio (NM), negativo pequeno (NP), zero (ZE), positivo pequeno (PP), positivo médio (PM), positivo grande (PG).

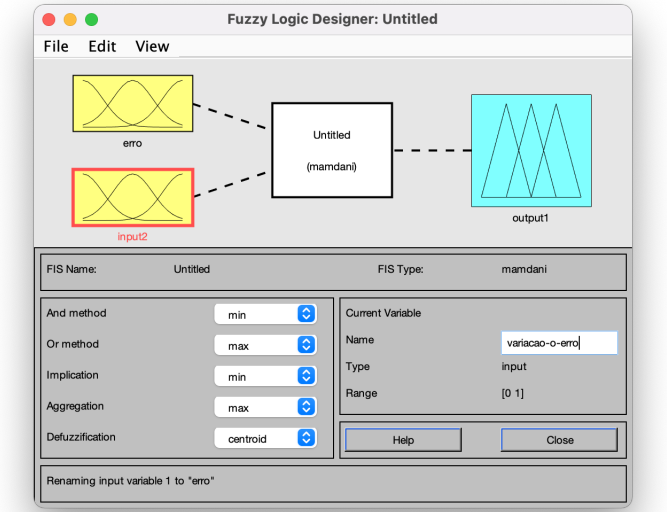
1. Duas entradas são necessárias. O menu indicado na imagem permite a adição de variáveis de entrada.



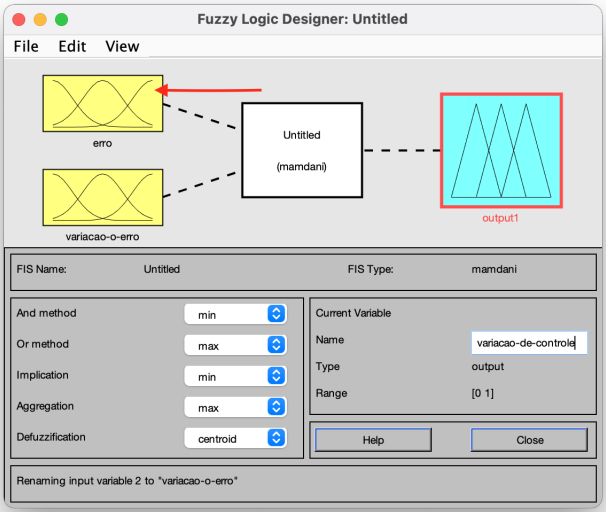
2. Selecione *input1* e defina o nome da variável como erro.



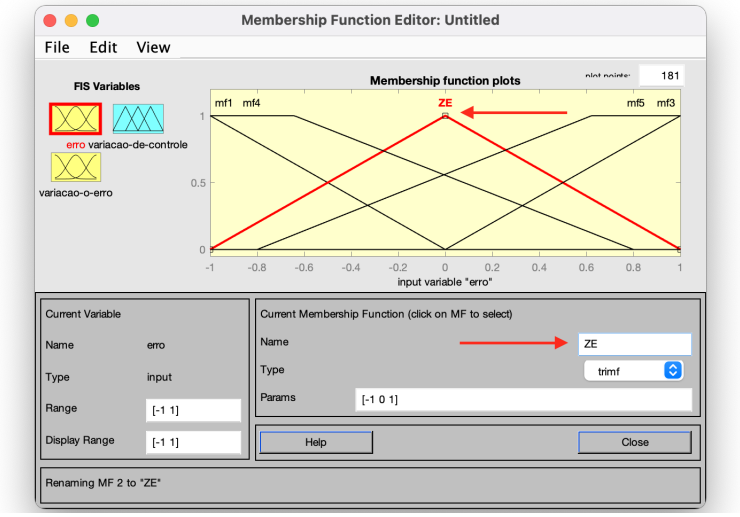
3. Selecione *input2* e defina o nome da variável como variacao-do-erro.



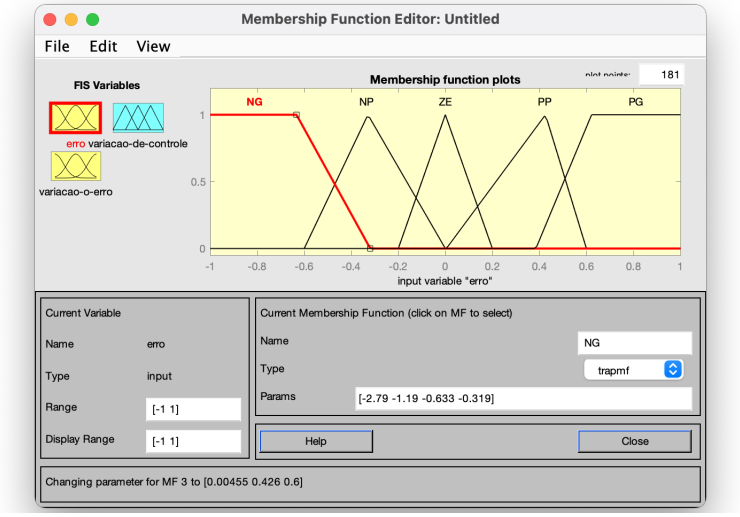
4. Selecione *output1* e defina o nome da variável como variacao-de-controle. Após isso, clique duas vezes em "erro"(seta vermelha).



5. A GUI *Membership Function Editor* irá abrir. Primeiramente remova todas as funções de pertinência pre-existentes: *Edit* $\longrightarrow$ *Remove All MFs*.
6. No campo *Range*, defina o domínio de valores da variável de entrada que, nesse caso, é o erro. No sistema de controle, é comum usar uma constante que multiplique a entrada para normalizá-la entre 0 e 1 ou -1 e 1. A escolha cabe ao usuário, desde que a faixa de valores possíveis para a entrada esteja bem representada no campo *Range*.
7. Adicione as funções de pertinência: *Edit* $\longrightarrow$ *Add MFs...*.
8. De acordo com a Base de Regra exemplo, tem-se 5 funções de pertinência para a entrada erro. Então, cria-se 3 partições do tipo de triangular e duas do tipo trapezoidais. Na caixa de diálogo que abriu, chamada *Membership Functions*, escolhe-se o tipo da partição e a sua quantidade. Primeiramente, escolhe-se *MF Type* igual a *trimf*, que corresponde a função triangular, e *Number of MFs* o número de partições do tipo definido que, nesse caso, é igual a 3.
9. Repete-se o processo para adicionar as duas funções de pertinência trapezoidais a entrada erro: *Edit* $\longrightarrow$ *Add MFs...*. O termo *MF Type* igual a *trapmf* e *Number of MFs* igual a 2. Obtem-se resultado similar a:

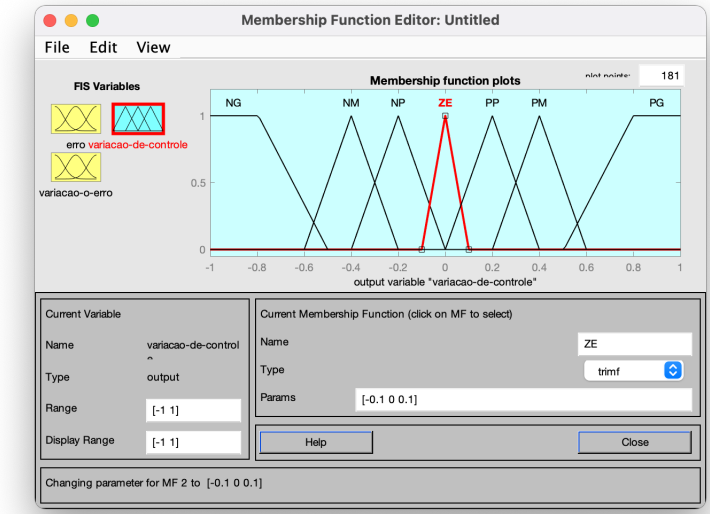


10. É possível selecionar cada função de pertinência e usar o mouse para ajustar individualmente o seu formato. Também é possível usar o campo *Params*, que é um vetor com valores que representam o formato da função de pertinência, para alterar o formato da respectiva função. Além disso, pode-se dar nome a cada função de pertinência de acordo com a base de regas. Após tais ajustes:

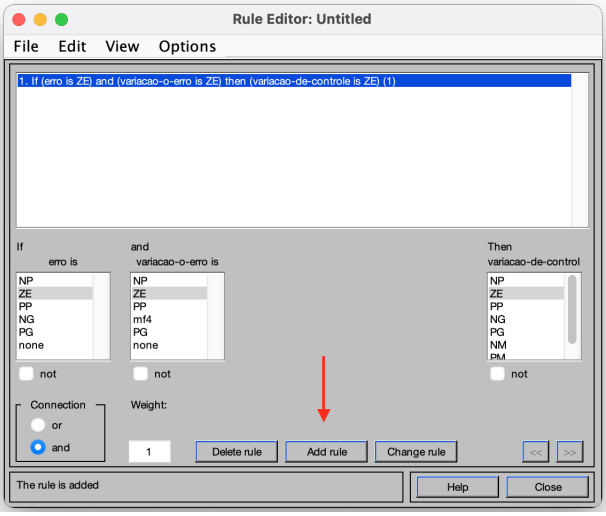


11. O mesmo processo de adição de funções de pertinência é repetido para a outra entrada, variação do erro. Clique duas vezes em "variação-do-erro"e siga desde o passo 5 até o 10. Atenção especial para o campo *Range*, que varia de acordo com a entrada que está sendo analisada, e com o número e o formato das funções de pertinência que pode variar.

12. O mesmo processo de adição de funções de pertinência pode ser aplicado a saída, variação do controle. Clique duas vezes em "variação-de-controle"e siga os passos do 5 ao 10. Adequar o campo *Range* ao domínio de valores da saída e implementar quantas funções de pertinência forem necessárias, de acordo com a base de regras. Nesse exemplo, também tem-se cinco funções de pertinência. Após a adição das funções e os devido ajustes, tem-se:



13. É necessário construir as regras de acordo com a base definida. Como nesse exemplo cada um das entradas tem 5 funções de pertinência, têm-se 25 regras a serem implementadas. Para tal, clique duas vezes no bloco "Untitled"ou no nome que você tenha atribuído ao sistema. O *Rule Editor* abrirá.
14. No *Rule Editor*, é necessário construir as 25 regras, para este exemplo. Selecione o valor de erro, a variação de erro e a respectiva inferência, variação de controle, a partir das entradas, de acordo com a base de regras previamente definida. Clique em *Add Rule* para adicionar a regra.



15. O método de "desnebulização"ou *defuzzificação*, pode ser escolhido no parâmetro *Defuzzification*. Por padrão, centroide é definido.
16. Para salvar, utilize: *File* $\longrightarrow$ *Export* $\longrightarrow$ *To Disk...*. O arquivo é salvo como ".fis".
17. Para usar a estrutura fuzzy ".fis"salva no Simulink, é preciso carregá-la para o *workspace* do *MATLAB*.

$$>> \text{fuz1} = \text{readfis('nome-arquivo.fis')};$$

3 Meta-heurísticas de Otimização Aplicadas a Controlador Nebuloso

Uma das dificuldades encontradas na implementação de controladores nebulosos é o grande número de parâmetros para serem sintonizados. Este número cresce de acordo com a quantidade de entradas, saídas, funções de pertinência e regras do controlador. Facilmente pode ultrapassar a casa de centenas de parâmetros, tornando a sintonia manual inviável. Nesse caso, técnicas de sintonia automáticas são desejáveis para facilitar o trabalho do projetista.

Recentemente, otimização baseada em meta-heurísticas vem sendo aplicada na busca de parâmetros para problemas difíceis ou complexos. Uma meta-heurística é um processo de geração iterativo que guia uma heurística subordinada combinando, inteligentemente, conceitos diferentes para explorar os espaços de pesquisa, a fim de encontrar, de forma eficiente, soluções próximas do ideal. Podemos entender que meta-heurísticas utilizam uma combinação de escolhas aleatórias e o histórico de resultados passados, encontrados pelo método, para realizar buscas na vizinhança dentro do espaço de pesquisa, isto é, no domínio de parâmetros que se quer encontrar.

Dentre as diversas estratégias de otimização baseadas em meta-heurísticas, o Algoritmo Genético (AG) é uma técnica não-determinística de busca e otimização que manipula um espaço de soluções potenciais utilizando mecanismos inspirados nas teorias de seleção natural de C. Darwin e na genética de G. Mendel. Os principais processos dos AGs são: representação do problema, função de aptidão, população, seleção, cruzamento, e mutação. Mais informações sobre o AG podem ser encontradas no material didático teórico da disciplina de Controle Inteligente. Aqui focaremos na manipulação dos parâmetros do controlador nebuloso.

3.1 Geração da População

A população é um conjunto de soluções candidatas (os indivíduos) que podem ser gerados aleatoriamente no início da execução do AG. O tamanho da população é definido pelo projetista. Um número grande de indivíduos implica em maiores chances de encontrar uma solução, entretanto, quanto maior esse número, maior será o tempo e custo computacional necessários.

No caso do controlador nebuloso, um indivíduo da população representa o conjunto de parâmetros que se quer otimizar. Normalmente, esse conjunto é representado pelos vértices das funções de pertinência de entrada e de saída, bem como as regras nebulosas. Assim, na geração da população, é preciso considerar a quantidade de entradas, saídas, funções de pertinência e de regras utilizadas no controlador nebuloso que se quer otimizar. Os limites de operação estabeleci-

dos para entradas e saídas também deve ser considerado na geração da população para que um indivíduo não extrapole a faixa previamente determinada. Como nas atividades de laboratório o controlador *fuzzy* é implementado no MATLAB via Simulink, a implementação do AG é feita via *script* no MATLAB. Para lidar com o sistema nebuloso pré-implementado, extraindo suas informações, é preciso importá-lo para o *workspace* do MATLAB.

```
1  % Leitura dos parametros do Controlador Fuzzy
2  Fuzzy = readfis('Fuzzy');
3  NumIn = getfis(Fuzzy,'NumInputs');
4  for i = 1:NumIn
5      NumMFsIn(i) = getfis(Fuzzy, 'input', i, 'NumMFs');
6      RangeIn(i,:) = getfis(Fuzzy, 'input', i, 'Range');
7  end
8  NumOut = getfis(Fuzzy,'NumOutputs');
9  for i = 1:NumOut
10     NumMFsOut(i) = getfis(Fuzzy, 'output', i, 'NumMFs');
11     RangeOut(i,:) = getfis(Fuzzy, 'output', i, 'Range');
12 end
13 limites = [RangeIn' RangeOut']';
14 NumRules = length(Fuzzy.rule)
```

O trecho de código acima ilustra como é possível extrair informações do controlador nebuloso .fis do tipo mamdani implementado na área de *script* MATLAB. Na linha 2, *readfis* exporta para o *workspace* o arquivo *Fuzzy.fis* que está na *Current Folder* do MATLAB como Fuzzy. O padrão para se obter outras informações de Fuzzy é utilizar a função *getfis*. O primeiro parâmetro passado à essa função é a variável exportada para o workspace que representa o sistema fuzzy. Outros parâmetros são passados a função *getfis* de acordo com o que se queira obter. O código acima ilustra como obter o número de entradas, armazenado em *NumIn*, bem como o número de funções de pertinência de entrada para cada uma das entradas e suas faixas de operação, representados por *NumFsIn* e *RangeIn*, respectivamente. Da mesma forma, aplica-se para a saída. Já para obtenção das quantidade de regas, utiliza-se a função *length*, nativa do MATLAB, com parâmetro a variável que representa o sistema nebuloso no *workspace*, no caso *Fuzzy*, seguido de *.rule*.

Considerando um controlador nebuloso que seja implementado com duas entradas de faixa de operação iguais, por exemplo de 0 a 30, cada uma com três funções de pertinência triangulares que, por sua vez, contam com três vértices cada, tem-se:

$PMFIn(:,:,1) =$
(2.9262 3.8096 8.3549 16.4064 18.9708 24.4417 27.1738 27.4013 28.7252)
(4.2566 4.7284 12.6528 14.5613 24.0084 27.4721 28.7150 28.9467 29.1178)

O termo $PMFIn$ representa os parâmetros das funções de pertinência de entrada. Cada vetor representa os parâmetros das funções de pertinência de cada

entrada adotada. Como no nosso exemplo são duas entradas, têm-se dois vetores. Cada valor, dentro dos vetores ilustrados acima, representa um vértice gerado aleatoriamente, mas dentro da faixa de operação e da quantidade correspondente de funções e vértices. Considerando uma saída com cinco funções de pertinência triangulares e faixa de operação de 0 a 4, ilustra-se o termo $PMFOut$, simbolizando os parâmetros das funções de pertinência da saida:

$PMFOut(:,:,1) =$
(0.1273 0.1428 0.6847 1.1077 1.5689 2.6219 2.6230 2.7149 2.8242
2.9725 3.0310 3.1688 3.3965 3.7360 3.8380)

A manipulação das regras dependem diretamente de dois fatores: a quantidade de regras implementadas no controlador nebuloso e do número de funções de pertinência de saída. De forma simples, para cada regra implementada, associa-se uma função de pertinência de saída. No escopo da abordagem estudada, ao objetivo é poder otimizar as regras de acordo com a função de pertinência de saída que é atribuída a cada regra. Mudar as escolhas de funções de pertinência de saída, de maneira adaptativa, para as regras, pode melhorar o modelo. Cada função de pertinência de saída é representada por um número inteiro associado a sua respectiva regra. Como, no exemplo, tem-se cinco funções de pertinência de saída, cada uma das nove regras de saída recebe um número de 1 a 5. Buscando a diversificação dos indivíduos, representa-se a possibilidade de 1 a 5 em números binários. Assim, precisamos de três números binários para representar cada valor de 1 a 5 e, como temos nove regras, tem-se uma sequencia de 27 dígitos binários representando os parâmetros de otimização das regras:

$Rules(:,:,1) =$
0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 1 0 1 0 0 1 0 1 1 0 1 1

Um indivíduo da população será, portanto, a combinação dos parâmetros das funções de pertinência de entrada e saída e os parâmetros das regras, isto é *PMFIn*, *PMFOut* e *Rules*, respectivamente. No nosso exemplo, PMFin é representado por dois vetores de dimensão igual a 9 cada, totalizando 18 posições, PMFOut é constituído por um vetor de tamanho 15 e, por fim, Rules com 27 valores. Nosso indivíduo terá um tamanho final igual a 60.

Com os indivíduos devidamente gerados, representando fielmente os parâmetros do controlador nebuloso, pode-se implementar as etapas do AG. A montagem dos parâmetros das funções de pertinência, de entrada e de saída, para cada indivíduo é crítica. Cada trecho deve seguir uma ordem específica de acordo com o formato das funções de pertinência utilizadas. A recomendação é que se utilize as funções triangulares, assim cada entrada e cada saída obedeceu a seguinte ordem:

$$a_1c_1a_2b_1c_2a_3b_2c_3a_4b_3[...]$$

Os termos a, b e c representam os vértice mais a esquerda, o centro e o vértice mais a direta do triângulo, respectivamente. Os termos 1, 2, 3 e 4, por sua vez, representam a índice da função de pertinência. Essa sequência é utilizada para as entradas e saídas do *fuzzy* mamdani, bem como para as entradas do *fuzzy* sugeno. Na saída do sugeno, cada função de pertinência têm 3 parâmetros: $[A\ B\ 0]$, e devem respeitar a seguinte condição $A > 0$, $B < A$.

3.2 Avaliação dos Indivíduos

Outro passo crítico da implementação do AG para geração adaptativa de parâmetros é a etapa de avaliação dos indivíduos. O procedimento padrão é gerar uma população de indivíduos e, para cada um deles, passar os parâmetro que o indivíduo carrega para o controlador nebuloso. A resposta do controlador é avaliada a partir de algum critério que indicará a taxa de sucesso do respectivo indivíduo que gerou os parâmetros utilizados. Assim, é possível avaliar a adaptação de cada indivíduo gerado, passo fundamental do AG. Dessa forma, os parâmetros que cada indivíduo guarda precisam ser passados,em tempo de execução, para o controlador nebuloso em operação no Simulink.

```
for i = 1:NumIn
    for j = 1:NumMFsIn(i)
        Fuzzy = setfis(Fuzzy,'input',i,'mf',j,'params',[
    parametros_individuo]);
    end
end
```

O código acima ilustra, de maneira genérica, como passar os parâmetros da função de pertinência j da entrada i para o sistema nebuloso importado como Fuzzy. O mesmo código, pode ser replicado para passar os parâmetros referentes as funções de pertinência de saída.

```
for i = 1:NumOut
    for j = 1:NumMFsOut(i)
        Fuzzy = setfis(Fuzzy,'output',i,'mf',j,'params',[
    parametros_individuo]);
    end
end
```

Já para passar os parâmetros referentes as regras, preenchemos o campo *consequent* da estrutura *rules* do sistema nebuloso Fuzzy. Como os parâmetros das regras foram computados de maneira binária, é preciso fazer uma conversão binária para decimal antes de atribuir o valor de cada regra.

```
for i = 1:NumRules
```

```
    Fuzzy.rule(1,i).consequent = bin2dec([parametros_individuo]);
end
```

Com os parâmetros passados ao Fuzzy, é recomendado verificar se o controladdo nebuloso no Simulink recebe o Fuzzy do *workspace*, atualizado com os parâmetros do indivíduo. A simulação pode ser inicia via Script e, ao final, o indivíduo pode ser avaliado. Uma forma de avaliar é através do Índice de Goodhart. Como tal procedimento pode ser levar longos intervalos de tempo, é recomendado salvar as populações de indivíduos, ou pelo menos a melhor população alcançada. Caso se queria avaliar outros indivíduos, a geração salva anteriormente pode ser utilizada para iniciar a nova etapa de busca.

4 Redes Neurais Artificiais

De acordo com Haykin (2007), uma rede neural é um processador maciçamente paralelamente distribuído de unidades reprocessamento simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso. Se assemelha ao cérebro por dois aspectos:

- Conhecimento é adquirido a partir do ambiente através de um processo de aprendizagem.
- Forças de conexões entre neurônios, pesos sinápticos, são utilizadas para armazenar o conhecimento.

As principais propriedades de uma RNA são: Não-linearidade, **mapeamento entrada-saída**, capacidade de generalização, adaptabilidade, resposta a evidências, informação contextual, tolerância a falhas e entre outras. Mais informações teóricas e técnicas sobre as redes neurais artificiais (RNA) podem ser encontradas na apostila teórica da disciplina de Controle Inteligente e nas mais diversas referências sobre o assunto. Aqui, nosso foco será o projeto de RNA.

4.1 Projeto de Redes Neurais Artificiais

No projeto de redes neurais foco da disciplina de Controle Inteligente, abordaremos a propriedade em destaque na seção anterior, o mapeamento entrada-saída. O mapeamento entrada-saída é feito através de um processo de *aprendizagem supervisionada*. Logo, envolve modificação dos pesos sinápticos de uma rede neural pela aplicação de um conjunto de amostras de treinamento **rotuladas**: $\{X, y\}$. Nesse tipo de amostra, cada exemplo consiste de um sinal de entrada único $\{X\}$ e de uma resposta-alvo desejada $\{y\}$ correspondente.

Apresenta-se à rede um exemplo escolhido ao acaso do conjunto $\{X_i, y_i\}$, e os pesos sinápticos (parâmetros livres) da rede são modificados para minimizar a diferença entre resposta desejada $\{y_i\}$ e a resposta da rede $\{\widehat{y_i}\}$, produzida pelo sinal de entrada. O treinamento da rede é repetido para muitos exemplos do conjunto até que a rede alcance o que é chamado de estado estável onde não haja modificações significativas nos pesos. Assim a rede aprende dos exemplos ao construir uma *mapeamento entrada-saída*, não necessitando de suposições ou conhecimento prévio do modelo.

4.1.1 Representação do Conhecimento em Redes Neurais Artificiais

Um conjunto de pares de entrada-saída, com cada par consistindo de um sinal de entrada e a resposta desejada correspondente, é referido como um conjunto de dados de treinamento ou amostra de treinamento.

Para ilustrar, considere o problema de reconhecimento de um dígito manuscrito. Neste problema, o sinal de entrada $\{X\}$ consiste de imagens representando um dos 10 dígitos como ilustrado na Figura 3. A resposta desejada é definida pela "identidade" do dígito particular $\{y\} = \{0, 1, 2, ..., 9\}$ cuja imagem é apresentada para a rede com o sinal de entrada.

Figura 3: *Ilustração do conjunto de imagens representando cada um dos 10 dígitos.*

Resumidamente, o projeto de uma Rede Neural pode prosseguir da seguinte maneira:

1. Coleta do conjunto de dados que será utilizado para treinamento e testes.
2. Preprocessamento e padronização dos dados.
3. Uma arquitetura apropriada é selecionada para rede neural. Ilustrando com o exemplo anterior, a camada de entrada consistindo de nós de fonte iguais em número aos pixels de uma imagem de entrada e, uma camada de saída consistindo de 10 neurônios (um para cada dígito). A arquitetura mais comum é uma rede completamente conectada que pode ter uma ou mais camadas intermediárias com todos os neurônios de uma camada l conectados a todos os neurônios da camada $l+1$. Redes Perceptron de Múltiplas Camadas (PMC) utilizam em suas camadas intermediárias funções de ativação *não-lineares* suave como a função sigmoide.
4. Um subconjunto dos dados de entrada (conjunto de treinamento) é então utilizado para treinar a rede por meio de um algoritmo apropriado. Esta fase do projeto é a aprendizagem.
5. O desempenho da rede é testado com dados que NÃO foram apresentados durante o treinamento. Uma imagem de entrada é apresentada para rede, mas desta vez não lhe é fornecida a identidade do dígito que corresponde a esta imagem em particular. Comparando-se o reconhecimento do digito for-

necido pela rede com a real identidade do dígito, o desempenho é estimado. Segunda fase chamada de generalização.

O projeto de uma rede neural é baseado diretamente nos dados do mundo real, permitindo-se que um conjunto de dados fale por si mesmo. RNA fornece um modelo implícito do ambiente no qual está inserida, realizando o processamento da informação de interesse. Em uma RNA com arquitetura específica, a representação o conhecimento do meio ambiente é definida pelos valores assumidos pelos parâmetros livres (pesos sinápticos e bias).

4.2 Coleta e Preprocessamento dos Dados

A coleta e o preprocessamento dos dados é parte fundamental do projeto de redes neurais e de aprendizado de máquina como um todo. É a partir dos dados e da qualidade do dados que são utilizados, que pode-se construir modelos de aprendizado de máquina robustos.

Para o sistema de tanques, recomenda-se a aplicação de sinais *PRS* (pseudo-random signals) na entrada na planta que abriguem toda a faixa de operação da bomba (-4V a +4V), utilizando o sistema de intertravamentos para evitar a bomba puxe ar ou que o tanque transborde.

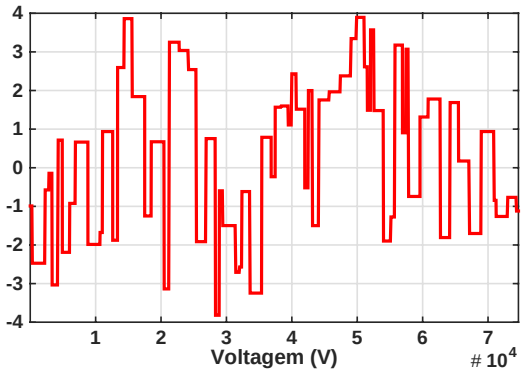


Figura 4: *Sinal PRS aplicada a entrada da planta para coleta de dados.*

A Figura 4 ilustra um exemplo de sinal *PRS*. Agora, os dados precisam ser ajustados para se adequarem a forma como o algoritmo de aprendizado pode processá-los. As três formas mais significativas de ajuste são: escala, padronização e normalização. Na escala, os dados são escaladas entre os valores 0 e 1 ou -1 e 1. Na padronização, o valor da média dos dados é removida de cada amostra

de dado e o resultado é escalado (dividido) pelo desvio padrão. Por fim, a normalização que, utilizando operações exponenciais/logarítmicas, adapta a distribuição dos dados para a distribuição normal.

Como no caso do sistema de tanques temos um sistema dinâmico, é preciso aplicar conceitos da engenharia de características (*feature engineering*) para obtermos modelos de RNA mais eficientes. Uma rede PMC é um rede essencialmente estática. Para que o modelo consiga aprender e generalizar a dinâmica intrínseca do sistema, podemos acrescentar algumas características aos conjunto de dados. Para sistemas dinâmicos especificamente, os valores atrasados de saída e de entrada do sistema podem ser usados como características (features).

Seja o conjunto $\{\mathbf{x}(t), y(t)\}_k$ com k amostras, o modelo tem uma única entrada $x(t)$ que é aplicada a uma memória de linha de atraso com q unidades, e uma saída única $y(t)$ que é realimentada para a entrada através de uma outra memória de linha de atraso, também com q unidades. As informações destas duas memórias de linha de atraso são utilizados para alimentar a camada de entrada da rede PMC. O valor presente da entrada do modelo é representado por $x(t)$, e o valor correspondente da saída do modelo é representado por $y(t+1)$; isto é, a entrada está atrasada com relação a saída por uma unidade de tempo. Assim o vetor de sinal aplicado à camada de entrada do PMC consiste de uma janela de dados constituídas das seguintes componentes:

- Os valores presente e passados da entrada, ou seja $x(t), x(t-1), ..., x(t-q+1)$ que representam as entradas.
- Os valores atrasados da saída, ou seja, $y(t), y(t-1), ..., y(t-q+1)$, sobre as quais é feita a regressão da saída do modelo $y(t+1)$.

Para ilustrar, utilizaremos como exemplo o conjunto de dados de um sistema dinâmico, o *benchmark* Box-Jenkins de uma fornalha a gás. O *benchmark* dispõe de uma variável $x(t)$ de entrada, que á taxa do fluxo de gás e a saída $y(t)$, que é a concentração do CO_2 no gás.

A Figura 5 lista os dados coletados sem nenhum preprocessamento (a) e após a padronização (b). Dois modelos PMC, A e B, de uma camada oculta e mesmo conjunto de hiperparâmetros são treinados. O modelo A utiliza o conjunto de dados (a) e o segundo modelo, igual ao primeiro em termos de arquitetura da rede, utiliza o conjunto de dados (b). A Figura 6 mostra o resultado dos testes dos modelos A e B. O modelo B, que utilizou os dados padronizados apresentou resultado significativamente melhor que A, justificam-se assim como o pre-processamento dos dados é parte importante na tarefa de aprendizado. Entretanto, mesmo B vencendo a disputa com A, seu desempenho parece não ser adequado a tarefa de identificar o sistema dinâmico da fornalha a gás Box-Jenkins. Adicionar atributos ao conjunto de dados, principalmente valores atrasados de entrada e de saída, pode melhorar a performance do modelo.

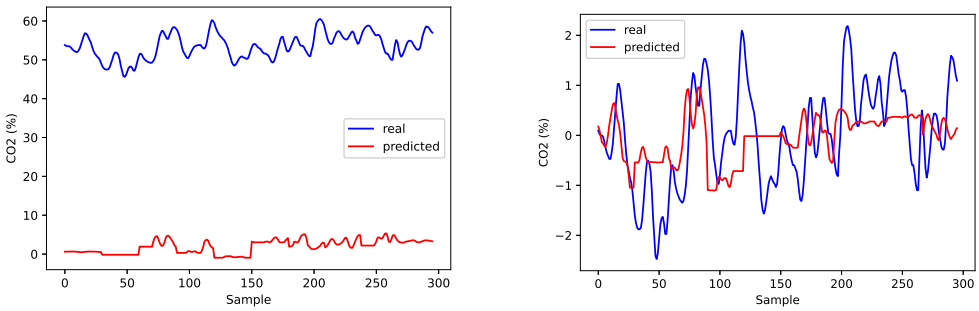
	Input Gas Rate	CO2 (%)
0	-0.109	53.8
1	0.000	53.6
2	0.178	53.5
3	0.339	53.5
4	0.373	53.4
...	...	...
145	-0.161	50.4
146	-0.553	50.2
147	-0.603	50.4
148	-0.424	51.2
149	-0.194	52.3

(a) Dados sem preprocessamento.

	Input Gas Rate	CO2 (%)
0	-0.048710	0.090993
1	0.053069	0.028429
2	0.219276	-0.002853
3	0.369610	-0.002853
4	0.401357	-0.034136
...	...	...
145	-0.097264	-0.972601
146	-0.463294	-1.035166
147	-0.509981	-0.972601
148	-0.342840	-0.722344
149	-0.128078	-0.378240

(b) Dados padronizados.

Figura 5: Ilustra-se parte do conjunto de dados da fornalha a gás Box-Jenkins. No item (a) têm-se o conjuntos de dados sem pre-processamento e o mesmo conjunto padronizado pela remoção da média e escalado a variância unitária, no item (b).



(a) A (b) B

Figura 6: Gráficos (a) e (b) ilustram os valores reais e estimados pelos modelos A e B treinados com dados sem pre-processamento e dados padronizados, respectivamente.

Adiciona-se características ao conjunto de dados: valores da entrada $x(t)$ atrasados até a ordem $q = 3$ e valores da saída $y(t)$ também atrasados até ordem $q = 3$. A Figura 7 ilustra o conjunto de dados gerados a partir desse processo de *feature*

Input Gas Rate	Input Gas Rate (-1)	Input Gas Rate (-2)	Input Gas Rate (-3)	CO2 (%) (-1)	CO2 (%) (-2)	CO2 (%) (-3)	CO2_(%)
-0.048710	-0.048710	-0.048710	-0.048710	0.090993	0.090993	0.090993	0.090993
0.053069	-0.048710	-0.048710	-0.048710	0.090993	0.090993	0.090993	0.028429
0.219276	0.053069	-0.048710	-0.048710	0.028429	0.090993	0.090993	-0.002853
0.369610	0.219276	0.053069	-0.048710	-0.002853	0.028429	0.090993	-0.002853
0.401357	0.369610	0.219276	0.053069	-0.002853	-0.002853	0.028429	-0.034136
...	...	...	...	...	...	...	...
-0.097264	0.425635	0.790731	0.956938	-0.941319	-0.878755	-0.816190	-0.972601
-0.463294	-0.097264	0.425635	0.790731	-0.972601	-0.941319	-0.878755	-1.035166
-0.509981	-0.463294	-0.097264	0.425635	-1.035166	-0.972601	-0.941319	-0.972601
-0.342840	-0.509981	-0.463294	-0.097264	-0.972601	-1.035166	-0.972601	-0.722344
-0.128078	-0.342840	-0.509981	-0.463294	-0.722344	-0.972601	-1.035166	-0.378240

Figura 7: Conjunto de dados do *benchmark* Box-Jenkins com adição de atributos: entradas e saídas atrasadas em 3 unidades.

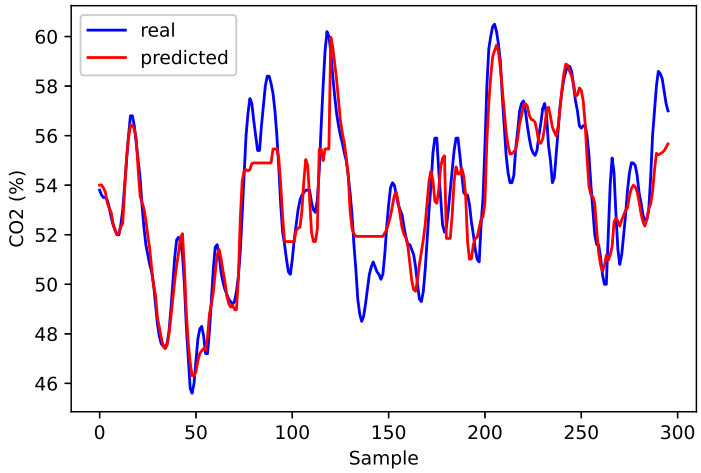


Figura 8: Valor real e valor estimado pelo modelo treinado com dados padronizados e adição de atributos.

engineering. Treina-se um modelo C, também PMC e similar aos modelos A e B treinados anteriormente, com mesma arquitetura, mas com o conjunto de dados após o processo de adição de características. O resultado do treinamento do novo modelo é exposto na Figura 8. O modelo C apresenta desempenho superior a B, se aproximando mais do valor real esperado. Reitera-se a importância do pre-

processamento dos dados com adição de features para identificação de sistemas dinâmicos não-lineares.

4.3 Treinamento e Validação das RNAs

4.3.1 Configuração da RNA

Depois da realização do pré-processamento, os dados estão prontos para serem apresentados à rede neural considerada para a etapa de treinamento. Esse processo pode ser trabalhoso e não segue uma receita fixa. De maneira geral, três etapas definem a configuração da rede:

- Definição do tipo de rede (paradigma neural) apropriado à aplicação que se deseja fazer.
- Definição da arquitetura da rede a ser treinada. Por exemplo: o número de camadas, número de neurônios (unidades) por camada, etc.
- Definição dos hiperparâmetros do algoritmo de aprendizagem. Por exemplo: taxa de aprendizagem, funções de ativação, etc.

As escolha dos parâmetros nas etapas de configuração da rede impactam no desempenho da rede resultante. Uma definição adequada de configurações faz a rede ter maiores chances de convergência no treinamento e maior capacidade de generalização. Novamente, não há uma receita a ser seguida nessa etapa. Normalmente, a definição dessas configurações e feita de forma empírica. Metaheurísticas também podem ser aplicadas a definição dessas configurações.

4.3.2 Treinamento da RNA

Após a etapa de configuração da rede neral, inicia-se o processo de treinamento. A etapa de aprendizado é gradual e iterativa, e é nessa etapa que os pesos das conexões são modificados várias vezes de acordo com uma regra de aprendizado (algoritmo) que define como os pesos são alterados. A cada iteração do algoritmo de adaptação dos pesos de uma rede é chamada de época e, em cada época, todo o conjunto de dados de treinamento é apresentado. Nessa fase é importante considerar alguns aspectos: inicialização da rede, o modo de treinamento e o tempo de treinamento. Uma boa escolha dos valores iniciais dos pesos da rede pode diminuir o tempo de treinamento. Normalmente, pesos inicias da rede são valores aleatórios com distribuição uniforme e um intervalo definido. Se a escolha dos pesos for inadequada, o processo de aprendizagem pode saturar já no início. O modo de treinamento mais empregado é o modo-padrão sequencial: a atualização dos pesos é realizada após a apresentação de cada exemplo do conjunto de treinamento. Esse método requer menor armazenamento de dados nos processamentos

correspondentes, além de ser mais resistente a problema de mínimos locais. Já o modo por lote (*batch*), no qual o ajuste dos pesos é realizado após a apresentação de todos os exemplos de treinamento que constituem a época, tem-se uma melhor estimativa do vetor gradiente no processo de aprendizagem supervisionada. Basicamente, a eficiência relativa dos dois modos depende do problema a ser tratado.

Durante o treinamento, uma rede pode se especializar nos exemplos do conjunto de treinamento, decorando essa base de aprendizado. Esse comportamento caracteriza um problema de aprendizado conhecido com superaprendizado (*overfitting*). Há diversas técnicas que tentam evitar ou reduzir os efeitos do *overfitting*. Aqui, recomendamos o validação cruzada (*cross validation*).

Há diversos fatores que podem influenciar no tempo de treinamento de uma RNA e, por isso, é necessário utilizar algum critério de parada. Normalmente, para o algoritmo de retropropagação do erro, é utilizado o número máximo de ciclos, no qual deve ser levado em consideração a taxa de erro médio por ciclo e a capacidade de generalização da rede. Em determinando instante do treinamento, a generalização pode começar e degenerar, levando a rede ao *overfitting*. No momento em que a rede apresentar uma boa capacidade de generalização e quando a taxa de erro for suficientemente pequena, o processo de treinamento deve ser interrompido. De maneira geral, o ideal é encontrar um ponto de parada com erro mínimo e generalização máxima.

4.3.3 Testes da RNA

Ao fim do processo de treinamento da RNA, o conhecimento adquirido está armazenado nas conexões sinápticas do modelo empregado, isto é, nas matrizes que representam os parâmetros livres, $\{\mathbf{W}, \mathbf{b}\}$ pesos e bias, da rede. Agora a capacidade de generalização da RNA precisa ser verificada e, para isso, iniciamos a fase de testes. Neste etapa, exemplos nunca vistos pela rede são apresentados à mesma. A RNA terá de responder corretamente a esses novos exemplos para confirmar que seu processo de treinamento foi satisfatório. Considerando que a resposta foi positiva, avança-se a próxima etapa que é integração da rede desenvolvida a aplicação. Entretanto, se o desempenho não for adequado, deve-se retornar a etapa anterior e refazer parte do processo. Alterações sugeridas são: alteração na topologia da rede, e diferentes valores de hiper-parâmetros.

4.4 Integração da RNA ao sistema de tanques acoplados

Com a rede devidamente treinada e validada, pode-se integrá-la ao ambiente da aplicação. No caso da disciplina de controle inteligente esse ambiente é o *Simulink*. O modelo treinado e validado pode ser representado como uma função *.m* do *MATLAB* que é facilmente incorporada um objeto do *Simulink* chamado de

bloco de função MATLAB (*MATLAB function block*) como ilustrado na Figura X a seguir.

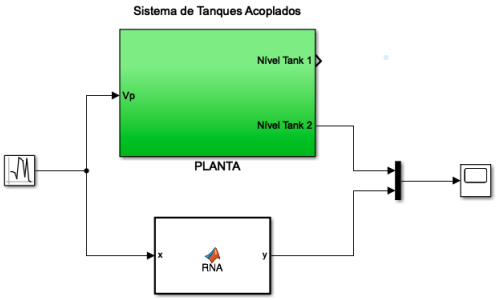


Figura 9: Esquemático exemplo da integração da RNA ao sistema de tanques acoplados

A Figura 9 ilustra o esquemático da simulação de tanques acoplados em operação junto com a rede neural. No exemplo ilustrado, é possível verificar graficamente os valores de nível do tanque 2 e a saída da rede que representação a estimação do nível do tanque 2.

4.4.1 NNtool

Caso a rede tenha sido criada utilizando o *nntool* - o *toolbox* de redes neurais do *MATLAB*, ao final de treinamento, exporte a rede para o *workspace*. Utilizando-se a função *gensim*, é possível gerar um bloco que represente a rede no *Simulink*. Para mais informações sobre esse procedimento consulte a documentação do *MATLAB* aqui.

A Figura 10 ilustra o esquemático da simulação de tanques acoplados em operação junto com a rede neural implementada no *nntool*. No exemplo ilustrado, é possível verificar graficamente os valores de nível do tanque 2 e a saída da rede que representa o nível do tanque 2.

4.5 Keras

Keras é uma API escrita em *Python* para aprendizado de máquina e, principalmente, aprendizado profundo, rodando no topo da plataforma *TensorFlow*. O Keras provê as funcionalidades e abstrações necessárias para o desenvolvimento e entrega de soluções baseadas em aprendizado de maneira eficiente.

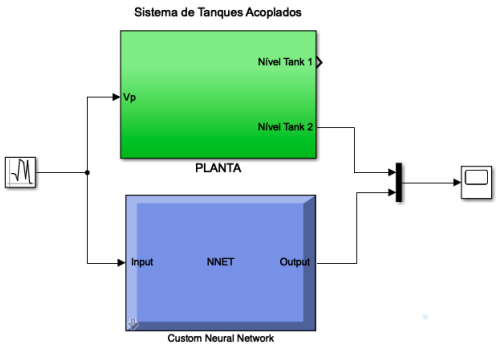


Figura 10: Esquemático exemplo da integração da RNA desenvolvida no nntool ao sistema de tanques acoplados

```
# first neural network with keras tutorial
from keras.models import Sequential
from keras.layers import Dense
# load the dataset
dataset = ...
# split into input (X) and output (y) variables
X = dataset[...]
y = dataset[...]
# define the keras model
model = Sequential()
model.add(Dense(20, input_dim=..., activation='sigmoid'))
model.add(Dense(1, activation='linear'))
# compile the keras model
model.compile(loss='mean_squared_error', optimizer='adam')
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
model.save('my_model.h5')
```

Listing 1: Exemplo Keras

O código 18 ilustra um exemplo básico para implementação de um modelo no Keras. No exemplo, uma rede neural com 20 neurônios na camada oculta e um uma unidade de saída é implementada. O treinamento é realizado em lotes de valor 10 e com 150 épocas. Mais informações sobre como implementar modelos de aprendizado de máquina no Keras podem se encontrados na documentação da própria API aqui. Caso o aluno opte pelo Keras para desenvolver a RNA, o modelo salvo pode ser facilmente importato no *MATLAB* para testes junto com a planta real. A função do MATLAB *importKerasNetwork* habilita ao usuário importar o modelo Keras como uma rede no MATLAB. Portanto, utilizando-se a função *gensim*, é possível gerar um bloco que represente a rede no *Simulink* como feito

para redes implementadas no *toolbox* de redes neurais do MATLAB, o *nntool*. Mais informações sobre *importKerasNetwork* podem ser encontradas aqui.