Universidade Federal
do Rio de Janeiro
Escola Politécnica

IMPLEMENTAÇÃO DE CONTROLE FUZZY EM SOFTWARE INDUSTRIAL

Publio M. Lima

Projeto de Graduação apresentado ao Curso de Engenharia de Controle e Automação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientadores: Mauricio Bezerra de Souza Júnior
Argimiro Resende Secchi

Rio de Janeiro
Abril de 2016

IMPLEMENTAÇÃO DE CONTROLE FUZZY EM SOFTWARE INDUSTRIAL

Publio M. Lima

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE AUTOMAÇÃO.

Examinado por:

Prof. Mauricio Bezerra de Souza Júnior, D.Sc.

Prof. Argimiro Resende Secchi, D.Sc.

Mario Cesar Mello Massa de Campos, D.Sc

Prof. Marcos Vicente de Brito Moreira, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2016

A meus pais, que sempre me deram suporte, tanto financeiro quanto emocional, para que eu pudesse chegar até aqui

Agradecimentos

Gostaria de agradecer a minha familia, meus pais pelo suporte financeiro e juntamente a minha irmã e meus irmãos pelo suporte emocional.

Aos meus orientadores e professores, que estiveram comigo durante a graduação e me ajudaram a chegar a esse ponto.

A PETROBRAS pelos uso dos softwares que foram utilizados nesse projeto e também pelo suporte da TOOL onde começou esse projeto.

Também à CoppeTeX pelo suporte do LaTeX utilizado nesse projeto.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Automação.

IMPLEMENTAÇÃO DE CONTROLE FUZZY EM SOFTWARE INDUSTRIAL

Publio M. Lima

Abril/2016

Orientadores: Mauricio Bezerra de Souza Júnior
Argimiro Resende Secchi

Curso: Engenharia de Controle e Automação

A lógica fuzzy é uma ferramenta muito poderosa na matemática. É uma evolução da lógica booleana, em que os valores lógicos das variáveis podem ser qualquer valor real entre 0 e 1. O conceito fuzzy traz a incerteza no valor verdade. A ideia por trás da lógica fuzzy, é imitar o comportamento humano na tomada de decisões. Com valores verdade intermediários consegue-se atribuir uma linguagem mais próxima da natural para as máquinas, por isso a lógica fuzzy tem sido muito utilizada na área de inteligência artificial.

A lógica fuzzy já é implementada em vários projetos e tem se mostrado uma excelente ferramenta.

Neste projeto são apresentadas a implementação da lógica fuzzy e a implementação de um controlador fuzzy PI em um software industrial, o MPA (Módulo de Procedimentos Automatizados) (TECGRAF/Puc-Rio/PETROBRAS). Fazem-se descrições sobre o processo de simulação de sistemas no simulador EMSO (Environment for Modeling, Simulation and Optimization) e comunicação interna entre servidores de controle (MPA-Server) e simulação (EMSO-OPC) através de um servidor intermediário (Top Server), além do controle feito pelo fuzzy PI em um sistema de primeira ordem e em um reator químico misturado com cinética de Van der Vusse.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer.

INDUSTRIAL SOFTWARE IMPLEMENTATION OF FUZZY CONTROL

Publio M. Lima

April/2016

Advisors: Mauricio Bezerra de Souza Júnior
Argimiro Resende Secchi

Course: Automation and Control Engineering

Fuzzy is a powerful tool on mathematics, it is an evolution on boolean logic, in which the logic values can assume any real value between 0 and 1. The concept of fuzzy brings the uncertainty in the truth value. The idea behind fuzzy logic is to mimic the human behavior in decision making. With intermediaries values of truth, it can be attributed a language closer to the natural for machines. Therefore it is very used in the artificial intelligence area. The fuzzy logic is already implemented on many projects and it is a great tool for works on the control and modelling area.

In this work, a fuzzy logic implementation on an industrial MPA (Módulo de Procedimentos Automatizados) (TECHGRAPH/Puc-Rio/PETROBRAS) software is presented. In this software a PI fuzzy controller is also implemented. The simulation process, on EMSO (Environment for Modeling, Simulation and Optimization) software, communication inside a simulation server (EMSO-OPC), a control server (MPA-Server) and an intermediary server (Top Server) are described. The simulations of a first order system and a chemical reactor is also presented, in which fuzzy PI control is applied.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

e_t	Erro no tempo t, p. 6
sp_t	SetPoint no tempo t, p. 6
y_t	Variável Controlada no tempo t, p. 6

Lista de Abreviaturas

EMSO	Environment for Modeling, Simulation and Optimization, p. 13
MPA	Módulo de Procedimentos Automatizados, p. 2

Capítulo 1

Introdução

1.1 Motivação

A lógica fuzzy, [1], é um avanço da lógica booleana clássica em que o valor verdade pode ser qualquer valor real de 0 a 1. Como descrito em [2], isso permite que estados não precisos sejam tratados por dispositivos de controle. Essa lógica tem como característica gerar valores de saída sem a necessidade de entradas precisas. Ela pode, então, ser aplicada para o controle de processos, pois aproxima do pensamento humano o modo de tomada de decisões feito por máquinas.

1.1.1 O Controle Fuzzy

Controladores mais complexos estão cada vez mais fáceis de serem implementados com os avanços tecnológicos. Como o controlador fuzzy tem uma característica natural de ser não linear, pode se ter um grande ganho com a sua implementação, já que possui um excelente potencial para controlar sistemas que o controlador tradicional PID não conseguiria. Ele também possui potencial para otimizar sistemas de forma melhorada, quando comparado ao controle linear.

Controladores fuzzy vêm crescendo muito, tendo várias aplicações de sucesso, como [3–8]. Isso, também, é um dos motivos pelo qual é interessante usar o controle fuzzy em outros tipos de sistema, e assim, ver se possuí um ganho também neste.

O controle fuzzy tem uma associação forte com a lógica matemática e, depois de estabelecido um sistema de controle baseado em fuzzy que funcione bem, pode ser usada uma linguagem quase natural, associada a uma lógica fuzzy, para controle de processos. Com essa linguagem mais próxima do natural, será mais fácil criar sistemas de controle.

1.1.2 MPA

O MPA (Módulo de Procedimentos Automatizados) (TECGRAF/Puc-Rio/PETROBRAS) é um software de automação utilizado pela PETROBRAS. Como descrito em [9], é uma ferramenta versátil, que pode ser adaptada para diversas aplicações, apenas modificando o algoritmo a ser executado.

Utiliza-se o software MPA, pois esse já é usado para controles complexos, normalmente supervisórios, na indústria (PETROBRAS). Já existem trabalhos criados no MPA, como [9], entre eles, alguns funcionando em plataformas e atuando em tempo real, vários não publicados por serem propriedade intelectual da PETROBRAS, tem-se um exemplo do uso do MPA no livro [10]. Estes tem conseguido grandes ganhos, em relação a trabalhos anteriores, com isso consegue-se perceber o quão útil é o MPA como ferramenta e sua aplicação real na indústria. Portanto, utiliza-se esta ferramenta como software para o desenvolvimento desse trabalho.

Além do fato de ter aplicação direta na indústria, o software MPA ainda permite uma grande variação de aplicações, já que ele é baseado num fluxograma, onde é possível desenvolver seu próprio algoritmo. Assim, ele pode funcionar tanto como uma simples calculadora quanto um dos controles mais avançados, como controle adaptativo ou o próprio controle fuzzy. Essa versatilidade é muito útil quando se está trabalhando com controles complexos, principalmente com controles baseados em lógica, pois assim se pode criar, dentro do próprio fluxograma, as exceções para aprimorar essa lógica.

1.2 Objetivo

O objetivo deste trabalho é implementar um controlador fuzzy no software de automação industrial, MPA. A ação de controle baseada em lógica fuzzy procura emular o comportamento de um controlador PI, porém com características não lineares do controle fuzzy. Ele foi aplicado a um sistema de primeira ordem e a um reator contínuo de tanque agitado (CSTR), com características não lineares e de fase não mínima. Os resultados foram comparados ao controle desses sistemas feito por um controlador PI, implementado no SIMULINK.

Além disso, implementou-se um método de sintonia para uma melhor comparação entre os controles, utilizando, no MPA, um algoritmo de otimização de Hooke & Jeeves, para a minimização da integral do erro quadrático.

Capítulo 2

Revisão Bibliográfica

2.1 Lógica Fuzzy

A lógica fuzzy é uma técnica que faz com que máquinas simulem o raciocínio humano na solução de problemas, para isso, ela procura descrever o comportamento humano na tomada de ações, como mostrado em [2].

A lógica fuzzy, diferentemente da lógica booleana, introduz a idéia de um valor lógico difuso (variáveis linguísticas), que se caracteriza por ter valor entre 0 e 1, assim ela consegue atribuir valores de incerteza entre o verdadeiro e o falso booleano, esse valor também pode ser chamado de grau de pertinência (verdade). As implementações da lógica difusa permitem que estados não precisos possam ser tratados por dispositivos de controle e permitem avaliar conceitos não quantificáveis.

Um sistema baseado em lógica fuzzy é separado normalmente em 3 partes, fuzzificação, inferências (que utiliza um banco de regras) e defuzzificação. A parte de fuzzificação se caracteriza pelo mapeamento de entradas numéricas em variáveis linguísticas, enquanto que uma lógica booleana usaria apenas valores de 0 ou 1. Como mostrado na figura 2.1, a lógica fuzzy pode atribuir vários valores intermediários de pertinência. A parte de inferências utiliza o banco de regras e associa variáveis linguísticas de entrada às variáveis linguísticas de saída. Por fim, a parte de defuzzificação retorna valores numéricos das variáveis linguísticas de saída.

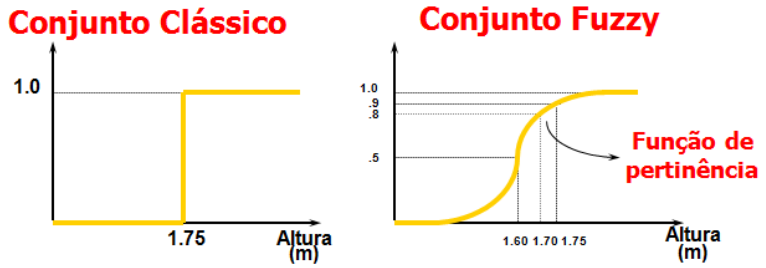

Figura 2.1: Clássico x Fuzzy.

A ideia de um valor lógico difuso é mostrado na Figura 2.1 de [2]. Na fuzzificação, normalmente, usam-se funções de pertinências em forma triangulares, trapezoidais ou gaussianas. Essas associam o valor real a um valor linguístico, como, por exemplo, pode-se criar uma função como na Figura 2.2, onde uma pessoa com 2 metros tem pertinência de 1 em ser alta, e uma pessoa com 1,5 m de altura é considerada 0,5 baixa e 0,5 média.

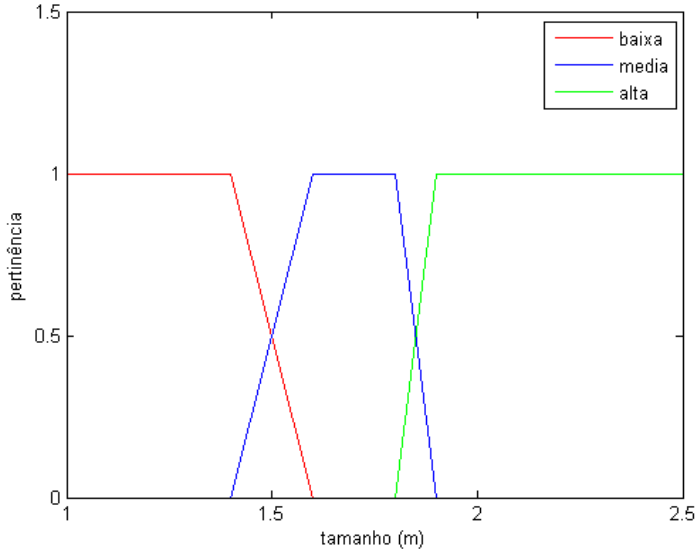

Figura 2.2: Fuzzy: Função de Pertinência das Variáveis de Entrada.

Associam-se também funções de pertinência a variáveis de saída. Pode-se usar, por exemplo, a variável risco de bater a cabeça ao entrar no carro como variável de saída representada pela Figura 2.3.

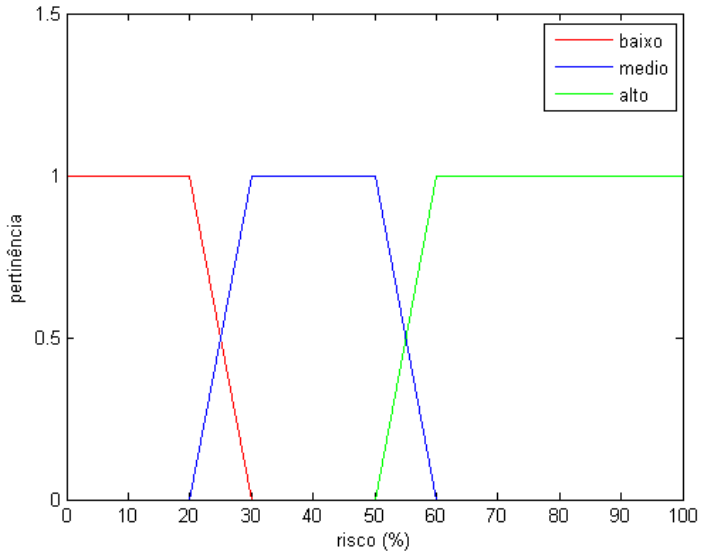

Figura 2.3: Fuzzy: Função de Pertinência da Variável de Saída.

Com essas pertinências, podem-se criar regras de inferência do tipo “se então”, como por exemplo, se uma pessoa é alta, o risco dela bater a cabeça também é alto ou se uma pessoa é baixa ou média, o risco é baixo. Com isso, por exemplo, se uma pessoa tem 1,5 m, então ela será, pela fuzzificação, 0,5 baixa e 0,5 média. Aplicando a regra de “se uma pessoa for baixa ou média o risco é baixo” tem-se o risco baixo como o máximo entre baixa e média, max(0,5;0,5) = 0,5.

Tendo 0,5 de pertinência em risco baixo, pode se passar para a parte de defuzificação, onde se gera inicialmente, com os valores de risco obtidos com a aplicação da regra, o gráfico mostrado na Figura 2.4. Com esse resultado, pode-se usar a técnica de defuzzificação escolhida. Existem várias formas de realizar esse processo, sendo o mais comum, e que será utilizado nesse trabalho, o método em que é feito uma média ponderada pela pertinência de todos os pontos do gráfico de saída, gerando um valor final para o risco, que é conhecido como o método da centróide. Neste exemplo, tem-se o risco 12,55 %.

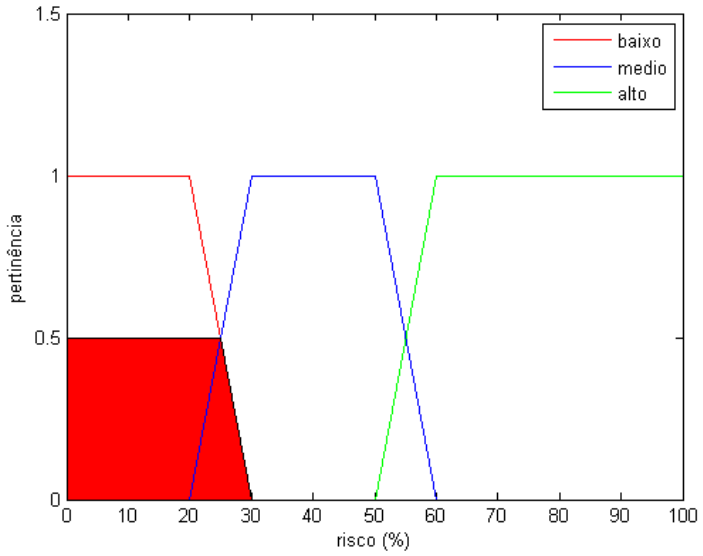

Figura 2.4: Fuzzy: Função de Pertinência de Saída com Aplicação da Regra.

Além de regras diretas, pode se criar regras com mais de uma entrada, para os casos em que existem 2 tipos de lógica, a lógica “ou”, onde o valor da saída é calculado como o máximo entre as duas entradas, e lógica “e”, em que o valor da saída é o mínimo entre as duas entradas. Essas regras equivalem à lógica proposicional em que os valores são sempre 0 ou 1.

2.2 Fuzzy PI

Na literatura, encontram-se alguns trabalhos que utilizam um equivalente do controlador PI, implementado com lógica fuzzy, como [11, 12]. Para esse trabalho, implementa-se o modelo de PI fuzzy apresentado em [11], onde o sistema é montado conforme a Figura 2.5 retirada de [11], em que G_e é um fator de normalização para o erro e G_r é um fator de normalização para a variação desse erro, G_u é um fator utilizado para voltar o valor de saída da defuzzificação para a dimensão da variável de controle.

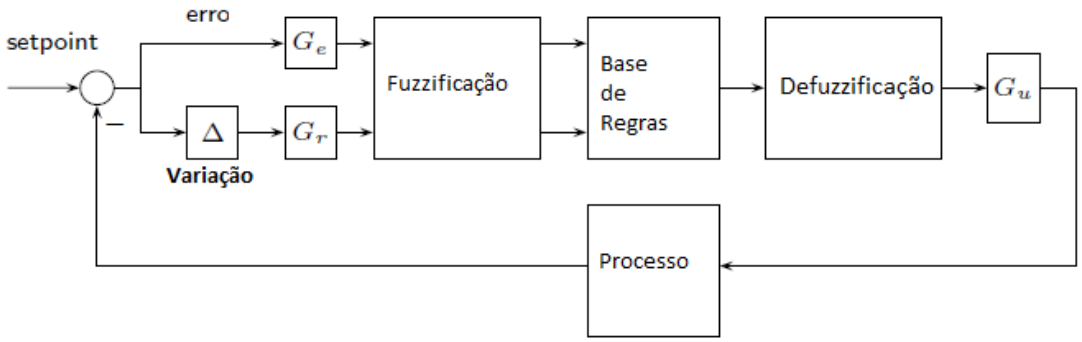

Figura 2.5: Fuzzy PI: Modelo de Controle.

A partir desse modelo, montam-se as variáveis de erro e variação deste erro (rate), definidas como $e_t := (y_t - sp_t)$, onde y_t é o valor da variável controlada no tempo t e sp_t é o valor do setpoint neste tempo, e $r_t := (e_t - e_{t-1})$, que são normalizadas para que estejam no intervalo entre -1 e 1, e depois desta normalização dentro da etapa de fuzzificação são multiplicados por fatores de controle Kp e Ki. Faz-se, então, a fuzzificação através da associação com as variáveis linguísticas mostradas na Figura 2.6 de [11], a qual mostra também a função de pertinência que foi usada para a saída (deriv).

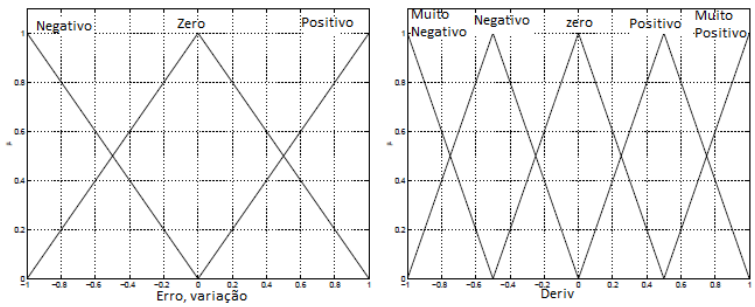

Figura 2.6: Fuzzy PI: Variáveis linguisticas Após Normalização [11]

Após criadas as variáveis e associado variáveis linguísticas às entradas, faz-se uso de 9 regras de inferência para simular o PI:

Tabela 2.1: Regras para Fuzzy PI

Regras	Se		Então
	Erro	Variação	Deriv
1	Negativo	Negativo	Muito Negativo
2	Negativo	Zero	Negativo
3	Zero	Negativo	Negativo
4	Negativo	Positivo	Zero
5	Zero	Zero	Zero
6	Positivo	Negativo	Zero
7	Zero	Positivo	Positivo
8	Positivo	Zero	Positivo
9	Positivo	Positivo	Muito Positivo

Após aplicada cada uma dessas regras, consegue-se fazer a defuzzificação através da centróide e gerar o controle incremental, que será acrescentado à entrada atual, $u(k) = u(k-1) + deriv$, gerando assim, a variável de controle para o sistema. O deriv é gerado de acordo com o gráfico das variáveis de entrada pela saída mostrado na figura 2.7.

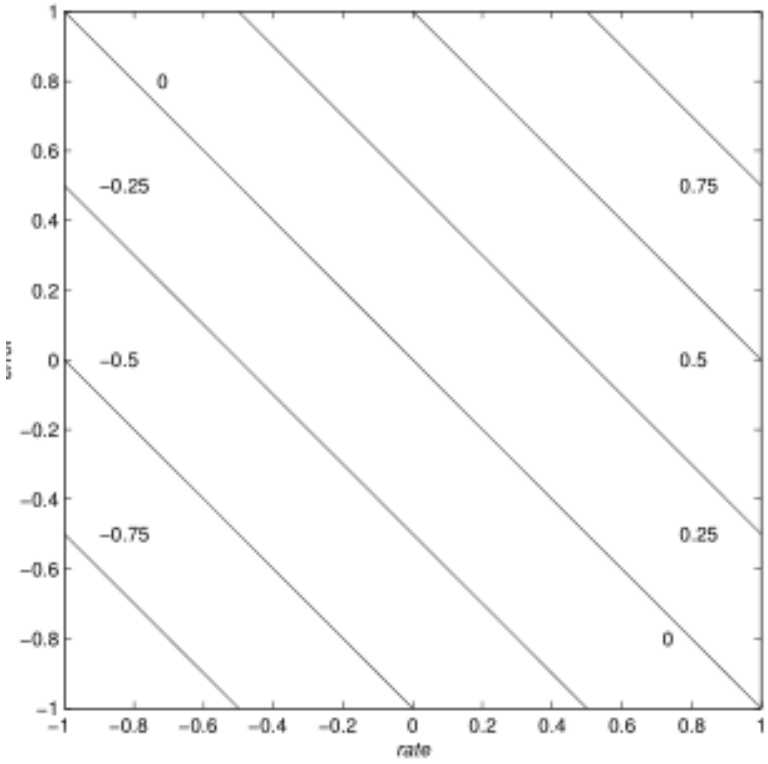

Figura 2.7: Erro e Variação do Erro (Rate) x Saída (Deriv)

Além disso, ainda pode-se usar 2 parâmetros de controle durante a normalização, onde o erro e a variação (rate) normalizados, que ficam entre -1 e 1, são multiplicados por esses parâmetros. O erro é multiplicado por um fator ki e a rate por um fator kp. O erro vai ser incrementado constantemente, fazendo assim o fator integral do erro, que vai ser multiplicado pelo fator ki, enquanto a variação ao ser integrada vai gerar o próprio erro, que vai ser multiplicado por kp, seguindo a lógica do controlador tradicional PI.

2.3 Fuzzy na indústria

Há um crescente uso do controle fuzzy na indústria. Desde 1993, o controle fuzzy já é desejado em algumas áreas da indústria como colocado em [3]. Ainda na década de 90, precisamente em 1994, encontram-se na literatura mais trabalhos utilizando a lógica fuzzy como, por exemplo, [7], no qual mostra uma implementação da lógica fuzzy criada para a indústria.

Atualmente existem vários trabalhos feitos com o uso de lógica fuzzy para controle, entre eles alguns foram apresentados no XVIII ENMC (Décimo Oitavo Encontro Nacional de Modelagem Computacional) em 2015, como [4], onde os autores fazem o uso da lógica fuzzy para propor um algoritmo capaz de diagnosticar a qualidade da energia elétrica de forma mais subjetiva, já que existem muitas definições

para esse fator; [5], no qual é usada a lógica fuzzy para avaliar a viabilidade de se fazer biodiesel a partir de pinhão manso, que é um produto capaz de suprir parte da necessidade por petróleo; [6], onde os autores conseguiram utilizar a lógica fuzzy para modelar e controlar o complexo sistema biológico de Junção do Joelho Conjunto Perna-pé cuja modelagem matemática foi proposta em 2000 de forma empírica; entre outros trabalhos. Em todos esses trabalhos é mostrado que o controle fuzzy tem um potencial para gerar grandes ganhos na indústria e um grande potencial para a área de controle e modelagem.

Em 2011, PRECUP e HELLENDOORN [8] fazem um resumo sobre a aplicação do fuzzy na indústria mostrando tipos diferentes de implementação, entre esses, controladores fuzzy de Mandani e a diferença entre esses e os de Takagi-Sugeno. Além de mostrar os avanços em controles adaptativos e preditivos utilizando lógica fuzzy, esse artigo ainda referencia muitos outros trabalhos criados com fuzzy, além de citar vários pontos de melhoria que precisam ser feitos em sistemas fuzzy.

2.4 Sintonia de Controlador

A sintonia de um controlador fuzzy é muito complicada, como descrito em [13, 14], pois ele tem muitos parâmetros. Normalmente variáveis linguísticas são representadas por curvas, que podem ser na forma triangular, trapezoidal, gaussiana, entre outras. Além dessa diversidade, cada curva normalmente é representada por vários parâmetros,por exemplo, uma forma de triangular tem 3 parâmetros: ponto inicial, final e de pico. Já existem trabalhos que criam um método para sintonia desse tipo de sistema, como é o caso de [13], que coloca um algoritmo estocástico para sintonia desses parâmetros da lógica fuzzy.

Uma técnica similar é utilizada no trabalho de ANDRADE (2014) [14], que mostra como sintonizar os parâmetros fuzzy através de um algoritmo genético de otimização para fazer um controle fuzzy MISO (múltiplas entradas e saída única) para o nível de um tanque com características não lineares.

No trabalho de PADILHA (2001) [15] encontra-se um outro modo de sintonia de controladores fuzzy, onde o autor utiliza redes neurais para o aprendizado do controlador fuzzy.

Quanto à sintonia de controles em geral, já existem muitos métodos conhecidos, por exemplo, o método de Ziegler & Nichols, descrito em [16], é muito aplicado na área química para sintonia de PIDs e para comparação com outros métodos, como em [17]. Outro método comum de sintonia é a alocação de pólos, [16], que requer o conhecimento do comportamento exato do sistema, sua função de transferência e que, se mal sintonizado, pode levar a instabilidade do sistema. Há ainda métodos baseados em otimização, como é o caso do método mostrado em [18], para sintonia de

controle preditivo. A sintonia por otimização busca minimizar uma função objetivo. Uma das funções objetivo mais comuns é a integral do erro ao quadrado no tempo, muitas vezes utilizando, também, um fator de minimização da variável de controle. A maioria dos métodos de sintonia por otimização tem pelo menos um fator com a integral do erro ao quadrado, pois é um ponto muito desejado na maior parte dos controles como em [19] e [20], que utilizam algoritmos genéticos de otimização para sintonia de controladores PID.

Dentre todos os métodos para sintonia de controle, nesse projeto faz-se uma adaptação de uma sintonia baseada em otimização, utilizando o método de busca de Hooke & Jeeves.

2.5 MPA

O MPA é um ambiente que consegue ler informações do servidor, executar um algoritmo e atuar nesse servidor, como demonstrado em [9]. O MPA é uma ferramenta adaptável para sistemas de automação e controle, permitindo o desenvolvimento de um algoritmo para que o sistema atue de forma desejável.

Esse software pode interagir nos mesmos pontos que um operador poderia, conseguindo assim, controlar o sistema como se estivesse em uma sala de controle. Ele pode atuar diretamente no supervisório ou em um outro servidor qualquer que se associe a ele.

É, portanto, uma ferramenta bem versátil. Nele é possível criar um algoritmo bem complexo, podendo ser usado para aplicação direta do controle, ativando atuadores, ou até para otimização, como implementação de um algoritmo de Hooke & Jeeves ou outros métodos de otimização.

Como descrito em [9], esse módulo é um software que consegue implementar um algoritmo criado em um fluxograma e assim manipular variáveis externas (através do MPA server). Ele é separado em 4 etapas: i. a pré-configuração, onde um programa criado em linguagem Lua é carregado no MPA e informa a ele que tipo de equipamentos podem ser utilizados e quais funções esses equipamentos possuem. Em uma linguagem de programação isso seria como a definição de classes, onde se colocam os equipamentos com seus parâmetros e métodos, e é assim, que são definidos em Lua (como classes); ii. A etapa da planta, onde são geradas instâncias dos equipamentos (classes), criados na pré-configuração. Nessa área, consegue-se colocar as variáveis externas (Pontos), e colocá-las como parâmetros para os equipamentos, se assim estiver configurado na pré-configuração; iii. A parte de fluxo, onde se monta um algoritmo que irá usar essas instâncias criadas na planta para fazer a lógica prevista e montar o controle ou algoritmo desejado; iv. Área de execução, onde o MPA faz o link com o servidor e pode controlar aspectos do servidor diretamente pelo MPA,

como iniciar ou parar um fluxo ou pedir para que o fluxo mande mensagens sobre o que está processando.

Essas 4 abas na Figura de interface do MPA podem ser vistas na Figura 2.8. A aba info (demonstrada nessa figura) possui as informações sobre a pré-configuração que foram programadas em lua e as outras 2 abas seguintes são as abas de configuração da planta e da geração do algoritmo no fluxograma. A última aba é a execução, como explicado anteriormente, nela faz-se a comunicação com o servidor.

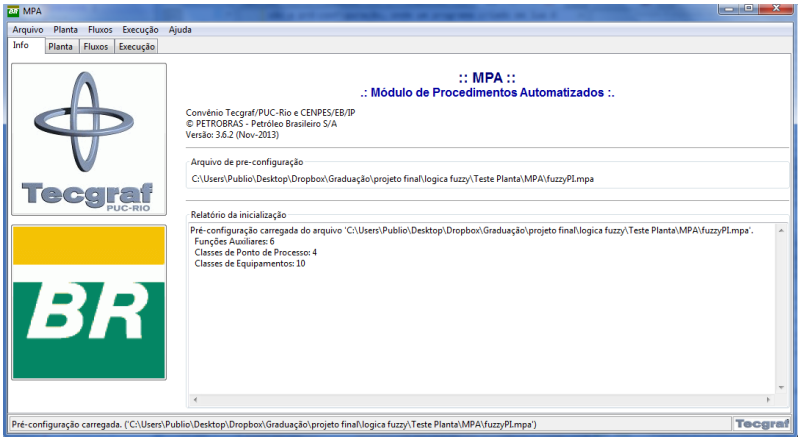

Figura 2.8: Tela inicial do MPA (Pré-Configuração).

A aba da planta possui interface como mostrada na Figura 2.9, onde se pode ver alguns parâmetros do equipamento FUZZY PI. Esse equipamento tem parâmetros além dos demonstrados nessa figura. Como dito anteriormente, essa aba tem como característica a criação de instâncias dos equipamentos com os quais será feita toda a lógica gerada na aba de fluxo.

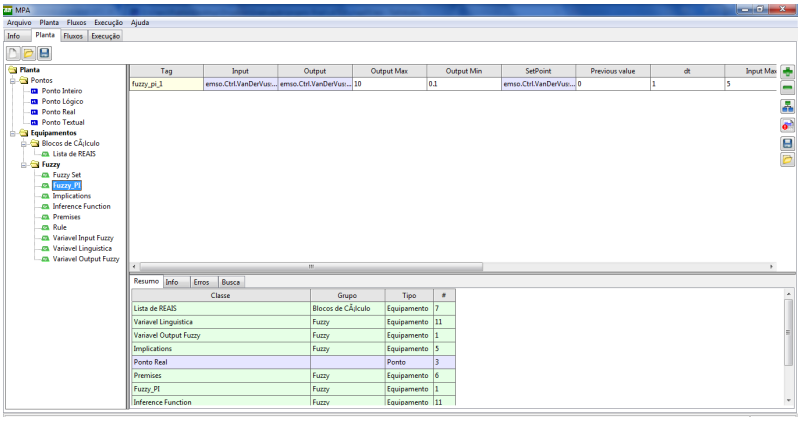

Figura 2.9: Exemplo de configuração de uma planta no MPA.

Para concluir a apresentação do funcionamento do MPA, é preciso descrever os fluxogramas. Eles são separados em 2 tipos, fluxos função e fluxos aplicação. Os fluxos aplicação são a lógica principal a ser executada e funcionam como um arquivo

de 'main' em uma linguagem de programação, outros fluxos podem iniciar ou parar esse fluxo, mas não podem criar novas instâncias. Por outro lado, fluxos função são como funções de uma linguagem de programação, esses podem ter parâmetros de entrada e saída e outros fluxos podem chamar várias instâncias desse fluxo, dentro da lógica principal.

Além do funcionamento do MPA em si, precisa-se da comunicação do MPA com os outros sistemas. Para isso, utiliza-se a aba de execução, que faz a comunicação através de um servidor da lógica interna com outros servidores externos ao MPA. Essa aba tem interface como mostrada na Figura 2.10.

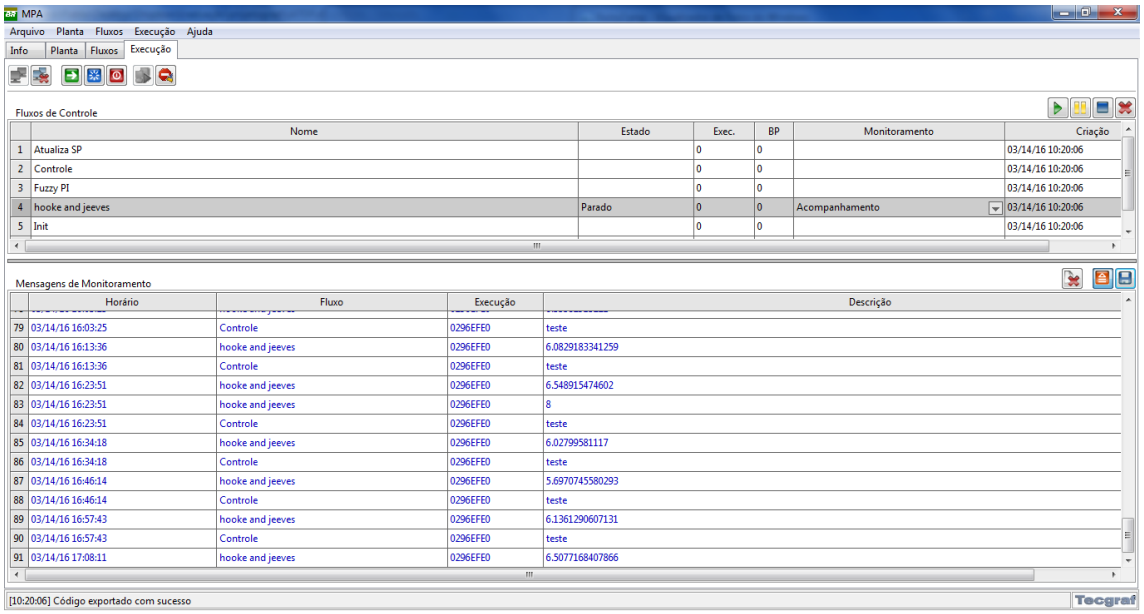

Figura 2.10: MPA interface de execução.

Nessa aba, pode-se conectar ou cancelar uma conexão do MPA a um servidor, enviar ou retirar informações de planta e fluxo desse servidor ou, até mesmo, encerrar o servidor que está conectado ao MPA. Abaixo dessas opções, a aba de execução diferencia os fluxogramas de aplicação, que podem ser inicializados, e fluxogramas de função, que devem ser chamados por outros fluxos durante a execução da lógica. Além dos fluxos, a aba de execução também mostra mensagens que foram configuradas para serem enviadas pelos fluxos em uma lista.

Capítulo 3

Método Proposto

A proposta desse trabalho é gerar um modelo de controle baseado em técnicas fuzzy para o MPA. Desse modo, este trabalho é separado em 4 etapas necessárias para o funcionamento desse controle:

- Simulação (no EMSO e no MatLab)
- Comunicação (no EMSO-OPC, no Top Server e no MPA)
- Controle (no MPA)
- Otimização (no MPA usando Hooke & Jeeves)

Utilizou-se o MPA para fazer o controle da planta, que foi simulada no EMSO. Assim, toda a parte de controle e sintonia (otimização) foi feita no MPA, se comunicando com o sistema através do MPAserver, enquanto que a parte de simulação e geração de dados foi feita no EMSO, que se comunica com o sistema através do EMSO-OPC, e toda a parte de variáveis e gerenciamento da comunicação foi feita com o Top Server,que se comunica diretamente com o servidor do EMSO-OPC e com o MPAserver. Além disso, foi feita a mesma simulação, controlada pelo método tradicional (PI), no MatLab. Nesse caso, tanto a simulação quanto o controle foram feitos no MatLab e assim não são necessários servidores para a comunicação entre os sistemas. Essa simulação foi realizada no MatLab para fazer a comparação entre resultados.

Como cada uma dessas partes foi feita para dois casos, um caso simples, com um sistema de primeira ordem, e um caso mais complexo, onde foi simulado um reator de Van der Vusse, esse trabalho foi dividido em aplicação 1 (primeira ordem), onde é feita uma introdução a comunicação e cada software utilizado e aplicação 2 (Van der Vusse), onde é mostrado como o controle se comporta em sistemas mais complexos.

Antes de examinar esses tipos de controle, foi feito um teste no MPA para conferir se a lógica fuzzy, ainda sem o controle PI, estava funcionando corretamente. Como

essa verificação é vital, pois mostra o funcionamento do fuzzy por si só, foi dedicada a ela uma sessão, além dos dois casos, onde é explicado sua implementação.

Como os códigos são extensos, com exceção dos de primeira ordem, que, por serem menores, são apresentados durante o texto. Todos os outros códigos utilizados estão expostos nos apêndices da seguinte forma:

- Apêndice A - Código de Fluxograma EMSO (Reator de Van der Vusse)
- Apêndice B - Código de S-Function para o MatLab (Reator de Van der Vusse)
- Apêndice C - Códigos de implementação do fuzzy no MPA

3.1 Teste Fuzzy

Esse é um teste para mostrar o funcionamento do fuzzy no MPA. Não é feito controle nenhum nessa etapa. Para testar o fuzzy, foi criado um exemplo simples no MPA, que gerencia dentro do próprio software e seu servidor todas as variáveis para a simulação. Os resultados do teste serão apresentados no próximo capítulo. Ele foi feito com 3 variáveis, dinheiro, pessoal e risco, onde as duas primeiras são as entradas e o calculo do risco é feita através da lógica fuzzy.

Essas variáveis linguísticas foram modeladas pelas funções de inferência, que podem ser vistas na Figura 3.1

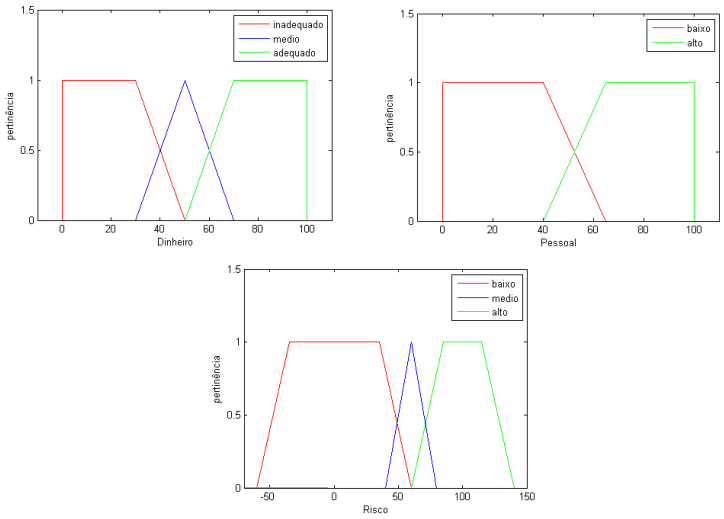

Figura 3.1: Funções de inferência para exemplo de lógica fuzzy.

Com essas variáveis, pôde-se aplicar uma lógica simples que testou os 3 casos de regras que podem ocorrer em um sistema fuzzy tradicional. Essas regras são:

- Regra ou - se o pessoal for baixo ou o dinheiro for adequado, então o risco é baixo.

- Regra e - se o pessoal for alto e o dinheiro for médio, então o risco é médio.
- Regra direta - se o dinheiro for inadequado, então o risco é alto.

Com essas lógicas foi criado um fluxo no MPA para utilizar esses dados na planta e fazer a lógica fuzzy. Esse fluxo pode ser visto na Figura 3.2, onde o MPA irá limpar todas os arquivos de gerenciamento e depois irá preenchê-los com os dados da aplicação das regras e, por fim, calcular o valor de saída através desses dados e a lógica fuzzy de cálculo de centróide.

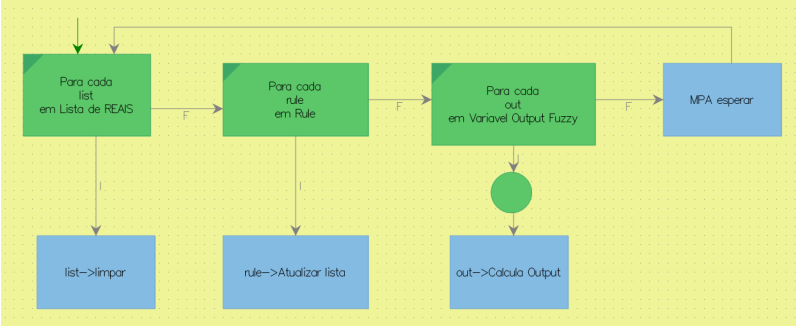

Figura 3.2: Fluxo no MPA para implementar lógica fuzzy.

Para completar o caso teste do fuzzy no MPA, ainda foi feita uma pequena interface, IHM (Interface Homem-Maquina), para tornar mais fácil o manuseio de variáveis e visualização dos resultados. essa simples IHM pode ser vista na Figura 3.3, onde pode-se alterar o valor das variáveis de entrada e quando o MPA estiver rodando é atualizado o valor da variável de saída.

Figura 3.3: IHM para teste fuzzy.

3.2 Aplicação 1: Sistema de primeira Ordem

3.2.1 Simulação

A simulação foi feita no EMSO. Este é um software que se baseia em equações, algébricas ou diferenciais, e possui uma interface gráfica onde pode-se ver gráficos

da simulação atual ou simulações anteriores. Ele é organizado em arquivos chamados de "Flowsheets"e dentro desses são definidos os equipamentos, conexões entre eles, variáveis, equações, especificações, condições iniciais, e opções da simulação.

No caso de um sistema simples de primeira ordem, esse programa ficaria como na Figura 3.4. Este caso possui uma simples equação diferencial e o sistema todo pode ser representado por uma função de transferência, no domínio de Laplace, dada por:

$$FT(s) = \frac{1}{(\tau s + 1)} \tag{3.1}$$

No caso da simulação desse projeto, para o caso de primeira ordem, o τ foi escolhido como 1 segundo.

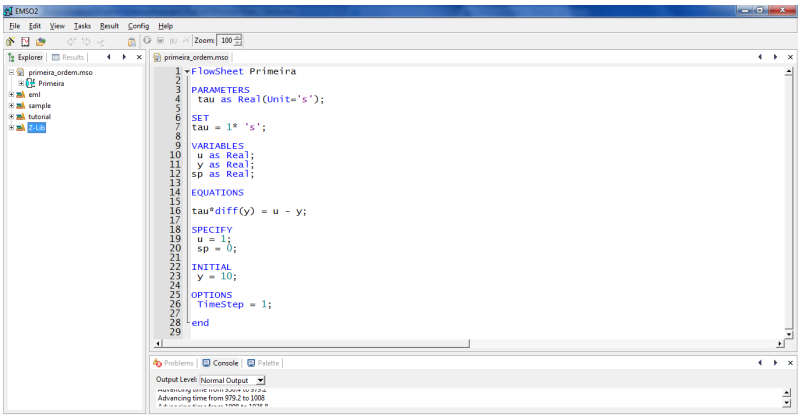

Figura 3.4: Flowsheet de primeira ordem no EMSO.

Além de utilizar o EMSO na simulação, onde esse flowsheet será usado posteriormente para a comunicação com o EMSO-OPC, também foi feita uma simulação de comparação no MatLab, utilizando o simulink. Essa mesma planta de primeira ordem no MatLab é mostrada na Figura 3.5

No MatLab para simulações mais complexas, utiliza-se um S-Function, mas como esse caso é simples, será melhor discutido o S-Function na sessão de Van der Vusse. No caso de primeira ordem, não é necessário o uso desse recurso.

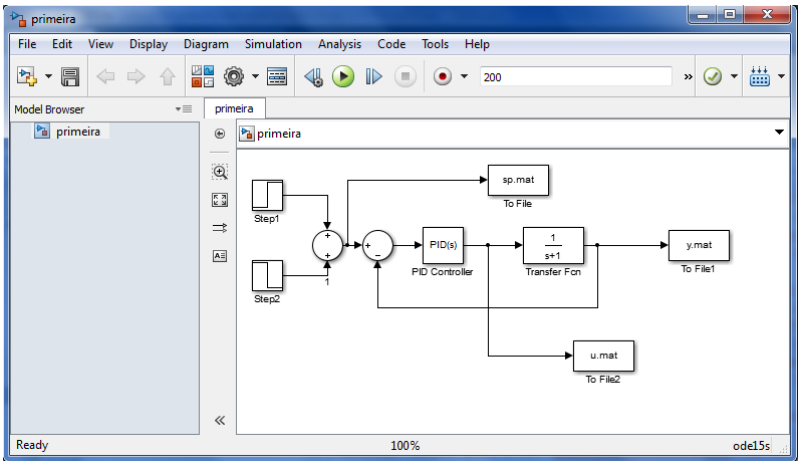

Figura 3.5: Simulação de primeira ordem simulink

3.2.2 Comunicação

Nessa sessão, discute-se, com mais detalhes, como foi feita a comunicação entre a simulação do processo e o controle, pois num sistema real essa comunicação não será imediata como é no MatLab, onde tanto o controle como as variáveis de processo estão no mesmo ambiente e assim, evitando atrasos de comunicação que podem instabilizar o sistema.

Para essa comunicação utilizam-se 3 servidores que trocam informações entre si e assim, conseguindo alterar o sistema (simulado no EMSO) através do controle (Construído no MPA).Esses 3 servidores são:

- O EMSO-OPC - Um servidor com comunicação tipo OPC que usa um flowsheet do EMSO para simular a planta.
- O Topserver - Um servidor gerenciador de variáveis que vai controlar e modificar as variáveis utilizadas tanto no controle como na simulação.
- O MPAserver - Um servidor próprio do MPA que fará o controle e se comunicará com o Topserver para controlar o sistema na simulação.

Esses servidores se comunicam como mostrado na Figura 3.6. O Top server é o gerenciador de variáveis principal, e se comunica com o EMSO-OPC, pegando as variáveis medidas deste e retornando para ele as manipuladas para a simulação. O MPA Server envia as variáveis medidas e então pega as manipuladas para o funcionamento do controle. Desse modo, o Top Server é um intermediário que comunica o controle com a simulação.

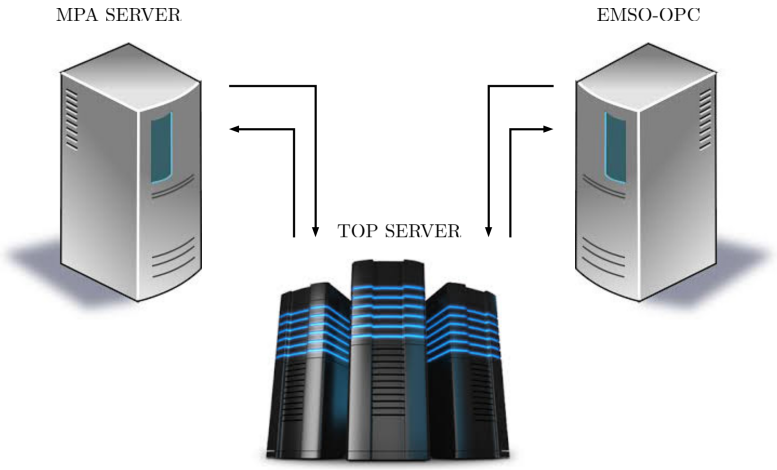

Figura 3.6: Comunicação entre Servidores.

EMSO-OPC

Mais especificamente, dizendo como cada um desses servidores funciona, é detalhado o servidor de simulação (EMSO-OPC). No EMSO-OPC o programa irá criar links entre variáveis de simulação e externas (Top Server). Ele possui a interface, configurada para o caso de primeira ordem, como na Figura 3.7. Esse servidor tem um botão onde se coloca o flowsheet, que vai trabalhar fazendo a simulação; um outro botão à direita, faz a comunicação com outro servidor (Top Server), e tem os comandos do próprio servidor OPC, onde criam-se novos links de comunicação, e comando para iniciar a simulação (no botão de "play").

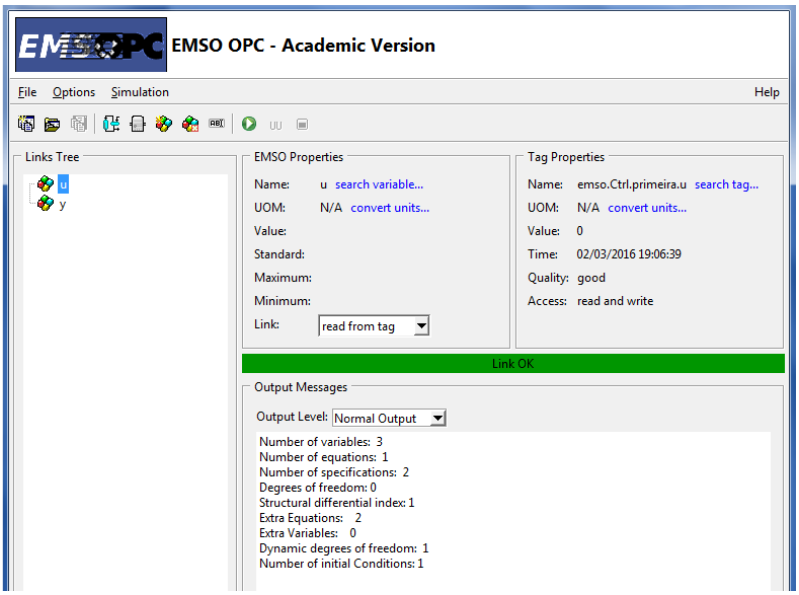

Figura 3.7: EMSO-OPC para sistema de primeira ordem.

Já na parte central da tela, ainda na Figura 3.7 é mostrado como são criados os links entre variáveis, definindo o tipo de variável e se lerá ou escreverá no ser-

vidor. Nesse caso, as variáveis controladas (medidas) são escritas no servidor e as manipuladas são lidas pelo EMSO-OPC. Em seguida, liga-se a essa variável OPC uma variável do flowsheet, escolhida na esquerda, e uma do servidor, conectado na direita, gerando assim um link entre a simulação e o gerenciador.

Top Server

O Top Server, como dito anteriormente, é um servidor gerenciador de variáveis. Ele é o primeiro a ser ligado, pois todos os outros se comunicam através dele. Ele também se comunica por protocolo OPC e possui um cliente, o "OPC Quick Client", onde pode-se ver as variáveis em tempo real, mostrado no canto esquerdo da Figura 3.8. O resto da interface gráfica dele, também mostrado nessa figura, possui um criador de variáveis, onde é escolhido o tipo dessa variável, seu nome, o tipo de acesso (read, write ou ambos) e aloca-se memória para essa variável. Se forem alocadas memórias muito próximas, pode haver problemas, por exemplo, como um float ocupa 4 bites de memória, após uma variável desse tipo, a próxima tem que ser no minimo quatro bites depois, caso isso não seja respeitado, os números interagirão entre si de forma não prevista gerando números quase aleatórios.

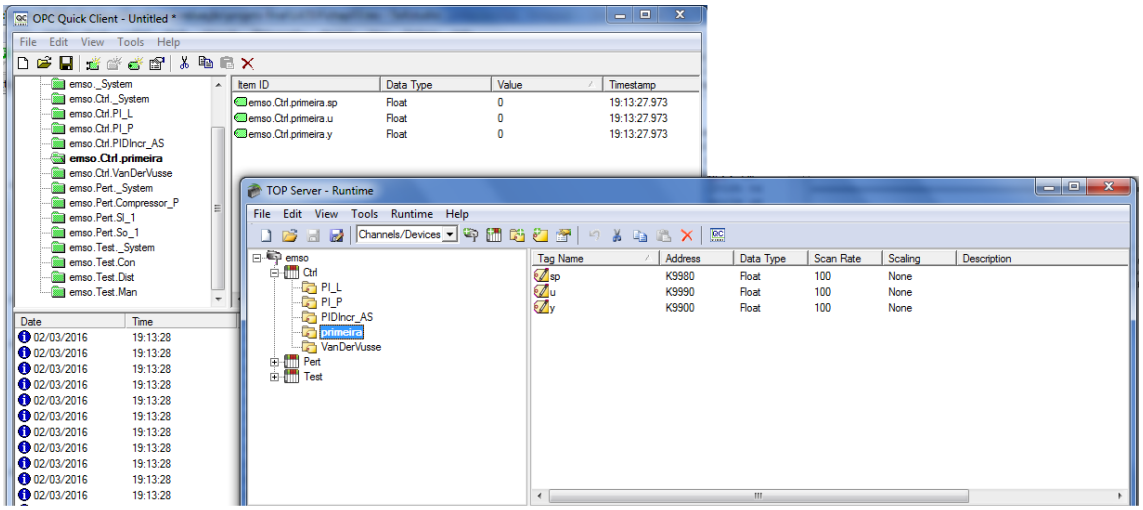

Figura 3.8: Top Server configurado para sistema de primeira ordem.

Diferentemente dos outros servidores, no Top Server se consegue gerar as variáveis sem que elas estejam definidas no processo ou no MPA. Essa vantagem faz do Top Server um bom servidor de gerenciamento, pois as variáveis dele não têm que fazer parte do sistema. Além disso, outra principal vantagem do Top Server em relação aos outros servidores é que ele pode ser conectado a mais de um servidor ao mesmo tempo. O EMSO-OPC e o MPA Server não podem se ligar a mais de um

servidor, por isso, eles são ligados ao Top Server e, assim, é gerada uma comunicação completa.

MPA Server

O MPA Server é o mais simples de todos os servidores utilizados. Ele tem uma interface em prompt comando (cmd), mostrada na Figura 3.9, e recebe os dados da planta e dos fluxos configurados no MPA para fazer o controle da planta, isso é melhor discutido na parte de controle.

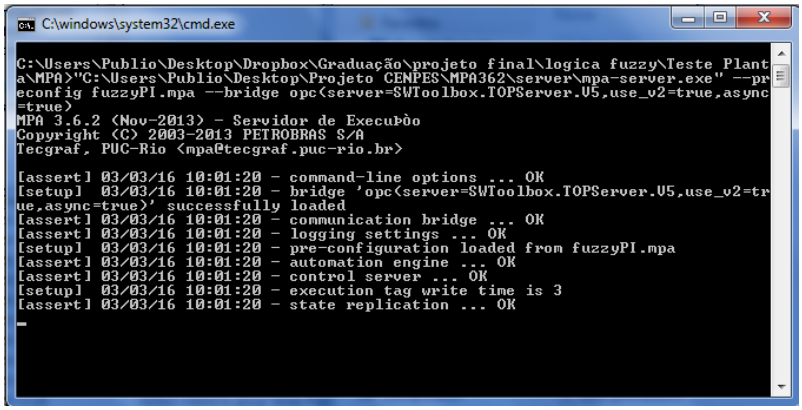

Figura 3.9: MPA Server.

Ele recebe uma pré-configuração, programada em lua, que deve ser a mesma configurada no MPA antes de enviar os dados para ele. Caso contrário, ele pode não reconhecer alguma função ou equipamento ou fazer uma função diferente da que foi prevista ao programar o MPA. A pré-configuração será melhor detalhada no item 3.23.

Com a programação em lua, esse servidor consegue se comunicar através de uma ponte (bridge) com o servidor do Top Server e, com essa comunicação definida, é possível rodar os fluxogramas programados, alterando as variáveis no servidor. Há dois tipos de variáveis no MPA Server: Pontos, que são externos ao MPA e são as variáveis do Top Server, e variáveis internas, que não podem ser alteradas pelo Top Server já que pertencem somente ao MPA Server, sendo que essas variáveis internas não são do servidor e não podem ser enviadas para a simulação ou outro servidor. O fluxograma pode alterar tanto pontos quanto variáveis internas desde que configurado corretamente.

3.2.3 Controle

O controle foi feito no software MPA. Esse software funciona como explicado na sessão 2.5.

Para esse projeto, criou-se dois algoritmos, um para execução do controlador fuzzy e um para fazer a sintonia dos parâmetros desse controlador pela minimização da integral do erro quadrático pelo método de Hooke & Jeeves. No MPA, esses algoritmos foram testados para o sistema de primeira ordem e para o reator de Van der Vusse, pois foi configurado um PI fuzzy de uso geral.

Nesse trabalho, o controle é aplicado na forma de realimentação. O esquema desse controle pode ser visto na Figura 3.10, onde, na parte de sistema, tem-se a simulação no EMSO do processo a ser controlado, na parte de controle, tem-se o controlador fuzzy no MPA e na comunicação, tem-se os servidores que interagem entre si.

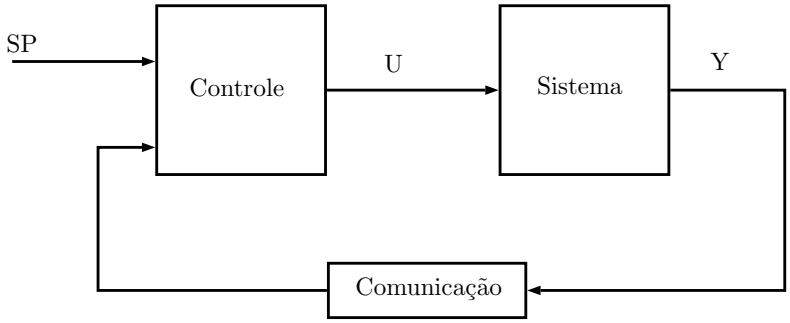

Figura 3.10: Arquitetura de Controle.

A pré-configuração foi programada em lua no aplicativo Notepad++, que é um conhecido editor de texto, muito utilizado por programadores. O código lua, que não será descrito integralmente aqui, pois é muito grande, por isso estará junto com os outros códigos mostrados nos apêndices, define os 9 equipamentos para fazer o fuzzy funcionar, além de utilizar um equipamento já pronto (lista de REAIS) e os pontos que são padrões do MPA. Esses 9 equipamentos, divididos em 5 grupos, são:

- Variáveis Linguísticas e funções de inferência - onde definem-se todas as funções linguísticas, tanto de saída como de entrada, para criar os parâmetros necessários para fuzzificar e desfuzzificar o sistema.

- Premissas, implicações e Regras - onde criam-se as regras fuzzy, utilizando variáveis linguísticas como premissas e uma função fuzzy e, ou ou direto (utilizando apenas uma variável), e outras variáveis linguísticas como implicação para fazer a lógica fuzzy dentro do MPA.

- Variáveis de entrada e de saída - onde definem-se todos os parâmetros das variáveis utilizadas na lógica fuzzy (limite inferior, superior, entre outros).

- Fuzzy Set - Basicamente é um gerenciador da lógica fuzzy.

- Fuzzy PI - onde colocam-se todos os parâmetros de controle PI, é feita a conexão com as variáveis do sistema real e são salvos os resultados do controle. A lógica fuzzy é a mesma para qualquer sistema, já que foi implementado um PI fuzzy; o que muda é esse equipamento, que definirá as entradas, saídas e parâmetros do sistema.

Cada um desses equipamentos possui parâmetros e métodos próprios que fazem o controle fuzzy funcionar quando todas as instâncias desses equipamentos estão bem configuradas. Para maior detalhes sobre esses métodos é necessário ver os próprios códigos que estão no apêndice C.

Como dito no Capítulo 2, a aba planta, que pode ser vista na Figura 2.9, tem mais parâmetros do que os mostrados. Entre esses parâmetros, o que será principalmente alterado, para sintonia, são os valores de Kp e Ki que são as referências para controle PI implementado, os outros devem estar bem sintonizados ou isso pode gerar problemas na sintonia, entre eles, os parâmetros de saída máximo e mínimo definem a normalização da saída e gera uma limitação para o máximo de ação de controle, o dt, que vai definir a ação do rate. Este deve estar igual ao passo do fluxo de controle. Os parâmetros de entrada máximo e minimo são usados para a normalização das entradas.

Já para a sintonia, precisa-se de mais um equipamento, que é o local onde se salvam os resultados obtidos pela sintonia. A maior diferença entre o controle e a sintonia está no fluxograma, já que a idéia da sintonia é rodar o programa de simulação várias vezes, tentando minimizar a integral do erro quadrático.

Nos fluxogramas pode-se construir a lógica a ser executada. A seguir é descrito cada fluxograma construído para o funcionamento do controle PI e depois o fluxograma para sintonia. Existem 4 fluxogramas para a parte de controle e um a mais para a parte de sintonia. Eles são:

- Init - É o fluxograma a ser executado, um fluxograma do tipo aplicação, que vai limpar os dados de controles anteriores e iniciar o fluxo "Controle".

- Controle - Também é um fluxograma do tipo aplicação, esse executará toda a lógica para fazer o controle em todo o tempo previsto.

- Fuzzy PI - Esse é um fluxograma do tipo função que é chamado durante um ciclo do Controle, ele executa as regras e atualiza variáveis do fuzzy.

- Atualização de SP - Outro fluxograma do tipo função, também chamado durante o ciclo do controle, para atualização do SP de maneira pré-programada para não ter que ser feito manualmente no servidor e assim poder comparar melhor os diferentes testes com a mesma planta.

- Hooke and Jeeves - Esse é o fluxograma exclusivo da sintonia. Usa o algortimo de otimização Hooke & Jeeves para sintonizar o controle do fuzzy PI.

O fluxograma Init é bem simples, como pode ser visto na Figura 3.11. Ele simplesmente zera todas as listas e inicia o controle. Ele é usado para a sintonia apenas no início para zerar as listas, não é utilizado sempre ao executar um controle pois o programa precisa de algumas listas para manter os dados de sintonia, então o fluxograma de controle é executado diretamente.

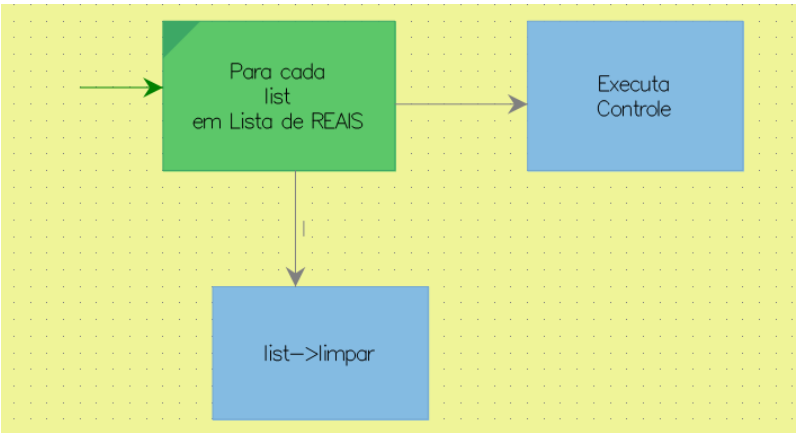

Figura 3.11: MPA Fluxo Init.

O fluxograma de controle, mostrado na Figura 3.12, foi construído para executar cada parte do controle. Então para cada equipamento de PI, ele lê os dados de entrada e set point, calcula os valores de erro e rate (normalizados para que estejam entre -1 e 1), encaminha para o fuzzy PI, que faz a aplicação da lógica fuzzy e retorna o valor do deriv, como mostrado anteriormente (no item 2.2). Com esse valor, o fluxograma atua sobre o sistema, modifica o valor da variável manipulada no servidor, e salva os valores das variáveis de interesse, para montar os gráficos de comparação posteriormente. Acabando esse ciclo, ele atualiza o valor de sp, utilizando o fluxograma de Atualização de SP, e retorna, para executar novamente o controle do inicio, até que acabe o tempo determinado de controle.

Além dessas funções, ainda nesse fluxograma, é acrescentada uma função a mais, quando utilizado para sintonia que está na Figura 3.12, dentro do quadrado. Essa parte do algoritmo garante que, antes de começar um novo teste de controle, o simulador volte para o mesmo ponto inicial. Isso é importante pois, dependendo da sintonia, ele pode não convergir novamente para o ponto inicial no final do controle anterior e então, a comparação teria um erro, já que a variável medida iria começar com um valor diferente.

Nota-se ainda na Figura 3.12 que, a cada instante é atualizada a variável erro no fluxograma, esse erro não precisaria ser atualizado para o controle, porém como

é necessário para a sintonia. Ele é utilizado para comparar as diversas sintonias. Como dito anteriormente, é utilizada a integral do erro quadrático e como este atualiza a cada instante, somando a variável do erro, essa variável terá, após todo o fluxograma de controle ser executado, a forma:

$$Erro = \sum_{i=0}^{t_{final}} e^2 \tag{3.2}$$

Para valores pequenos de passos tem-se o equivalente há:

$$Erro = \int_{0}^{t_{final}} e^2 dt \tag{3.3}$$

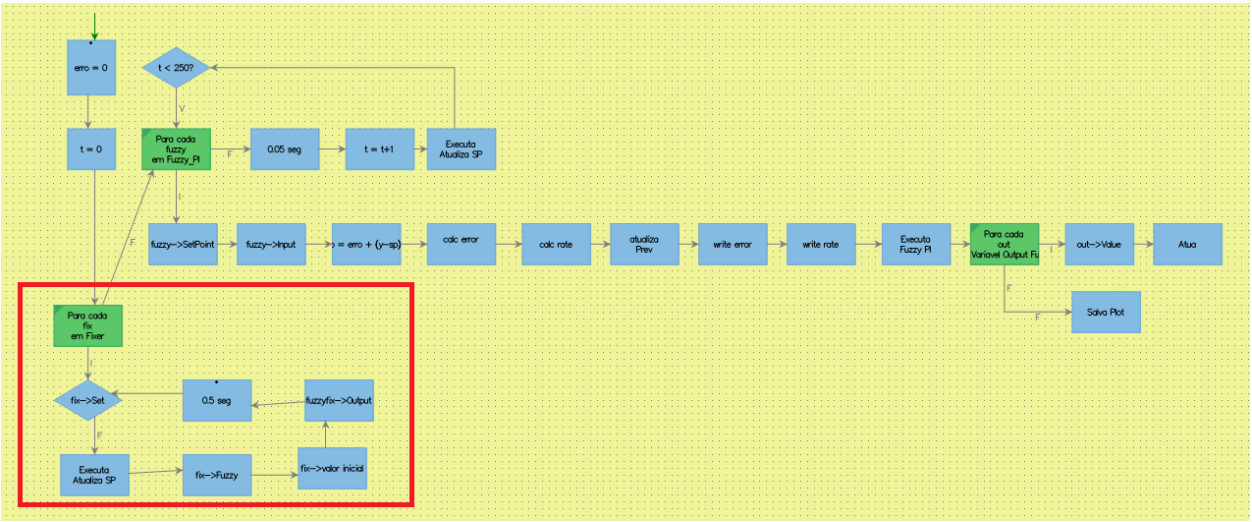

Figura 3.12: MPA fluxograma de Controle.

O fluxograma de Fuzzy PI é relativamente simples, porém ele está por trás de toda a lógica fuzzy do sistema, sendo então muito importante. Como mostrado na Figura 3.13, ele começa zerando as listas relativas a variáveis do fuzzy, para poder fazer novos cálculos com os mesmos arquivos. Na sequência, ele utiliza as regras fuzzy para determinar o valor das variáveis linguísticas de saída - neste trabalho as regras utilizadas foram baseadas no fuzzy PI de [11], então, as regras utilizadas são as mesmas das mostradas no capítulo 2 - e, finalmente, calcula a saída, através do cálculo de centróide, que será lançada novamente para o fluxograma de Controle.

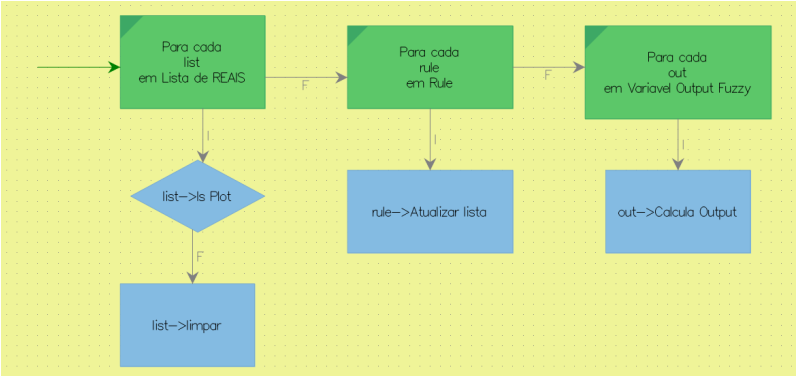

Figura 3.13: MPA Fluxograma de Fuzzy PI.

O último fluxograma não é importante para sistemas reais, pois terá um operador ou programa externo para o gerenciamento de set points. Porém, como o servidor não define os setpoints, é preciso fazer algumas mudanças para ver o comportamento do sistema e conseguir, assim, avaliar o controle fuzzy feito pelo MPA. Nessa idéia, o último fluxograma, que é um fluxograma do tipo função, recebe um valor de tempo e retorna um valor equivalente ao setpoint calculado naquele tempo de acordo com a lógica desejada, no caso em estudo, usa-se uma lógica simples de degraus, não necessariamente unitários, para fazer variações no setpoint. Esse fluxograma está mostrado na Figura 3.14.

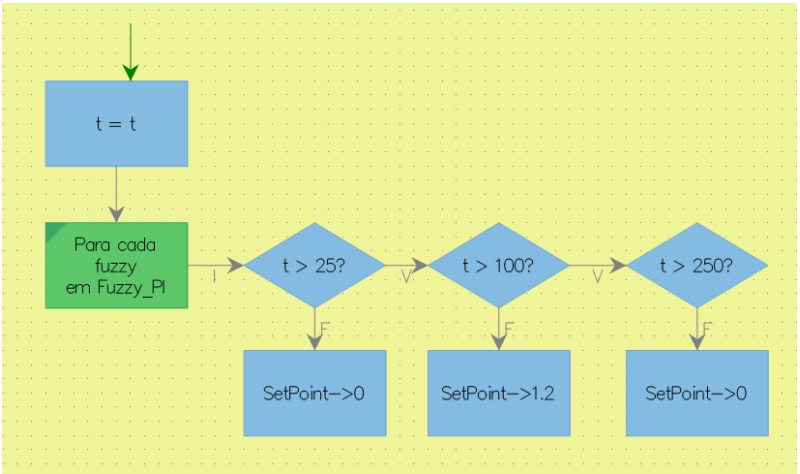

Figura 3.14: MPA fluxograma de Atualização de Setpoints.

3.2.4 Otimização

Nessa seção, é mostrada como é feita a implementação do algoritmo de busca por minimo local através de um fluxograma para a sintonia do controlador PI fuzzy. Como dito na revisão bibliográfica, pode ser utilizado um algoritmo de otimização para sintonizar um controlador, adotando-se uma função objetivo adequada. No

projeto foi usada a minimização da integral do erro quadrático e, como método de otimização, o Hooke & Jeeves.

Hooke & Jeeves

O método de Hooke & Jeeves, descrito em [21], é simples pois, só utiliza o cálculo da função objetivo para a minimização (não precisando calcular gradientes ou hessianas). O método precisa, como inicialização, um tamanho inicial para o incremento (δ), uma base inicial ($x(0)$) no eixo de coordenadas e uma tolerância (ϵ) para o critério de parada. Esse método de otimização tem como limitação a busca de soluções em direções cartesianas. Porém, ainda se mostra eficaz para diversas funções objetivo e, é de fácil implementação.

A partir do ponto inicial o método explora a vizinhança da base, usando o incremento inicial, em busca de uma direção provável de ótimo. A exploração é feita calculando-se a função objetivo em cada sentido (incrementos $+\delta$ e $-\delta$) de cada direção ao redor da base como mostrado na Figura 3.15, e escolhendo a direção provável de ótimo como a soma das direções que reduzirem o valor da função objetivo, ou seja, o algoritmo testa a direções cartesianas, no eixo das abscissas e no das ordenadas, e faz uma resultante entre as duas para gerar a direção de avanço. Com esta o algoritmo continua explorando até que haja um incremento da função objetivo e assim define uma nova base, onde volta a fazer o teste. Se em uma dada direção ele encontra um sentido que diminua o valor da função objetivo, ele não precisará calcular o valor da função no sentido oposto e usará apenas o sentido inicial para o cálculo da direção provável de ótimo.

Caso não encontre uma direção provável de ótimo, ou seja, todas as direções aumentam o valor da função objetivo em ambos os sentidos, então o algoritmo testará se já está com o incremento menor ou igual à tolerância e, se for o caso, o algoritmo irá encerar. Caso não esteja menor ou igual à tolerância, então o algoritmo irá dividir o valor do incremento (passo) por 2 e fará uma nova exploração.

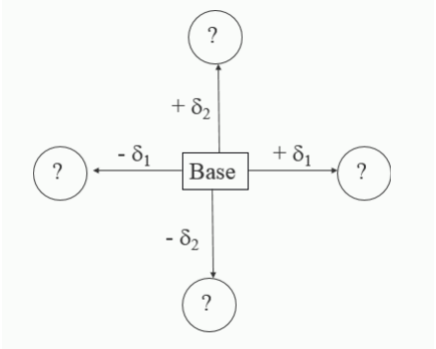

Figura 3.15: Exploração na Base.

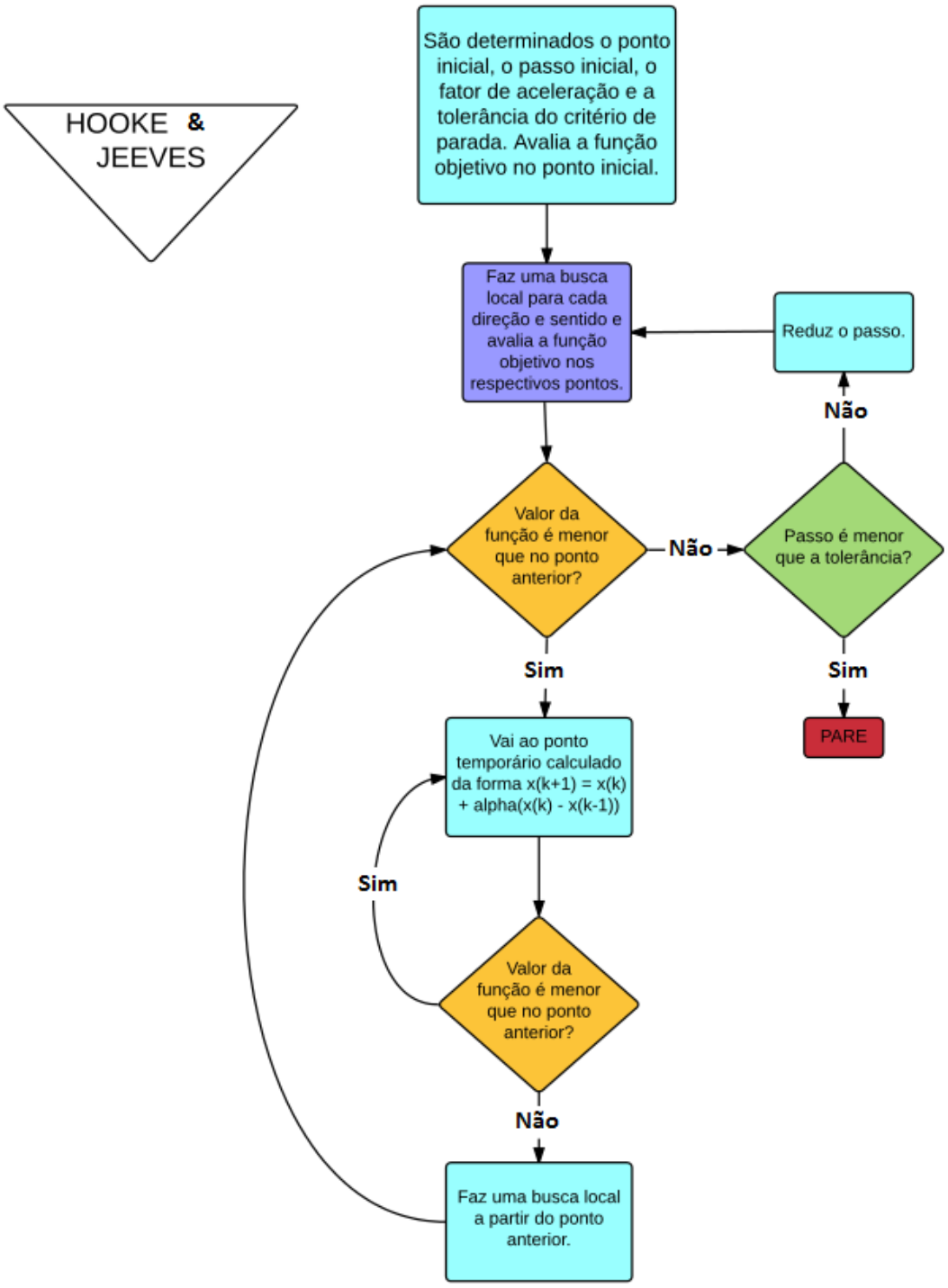

Figura 3.16: Fluxograma de descrição do método de Hooke & Jeeves.

Para implementar esse algoritmo no MPA, utiliza-se um fluxograma de Sintonia que pode ser visto na Figura 3.17, onde é feito cada passo do algoritmo, mostrado anteriormente, executa-se o fluxograma de Controle para calcular o funcional a ser minimizado, que é o somatório do erro quadrático em todo um ciclo de controle. Como esse fluxo roda o controle várias vezes, esse processo pode demorar muito

tempo, dependendo do tempo de um ciclo de controle.

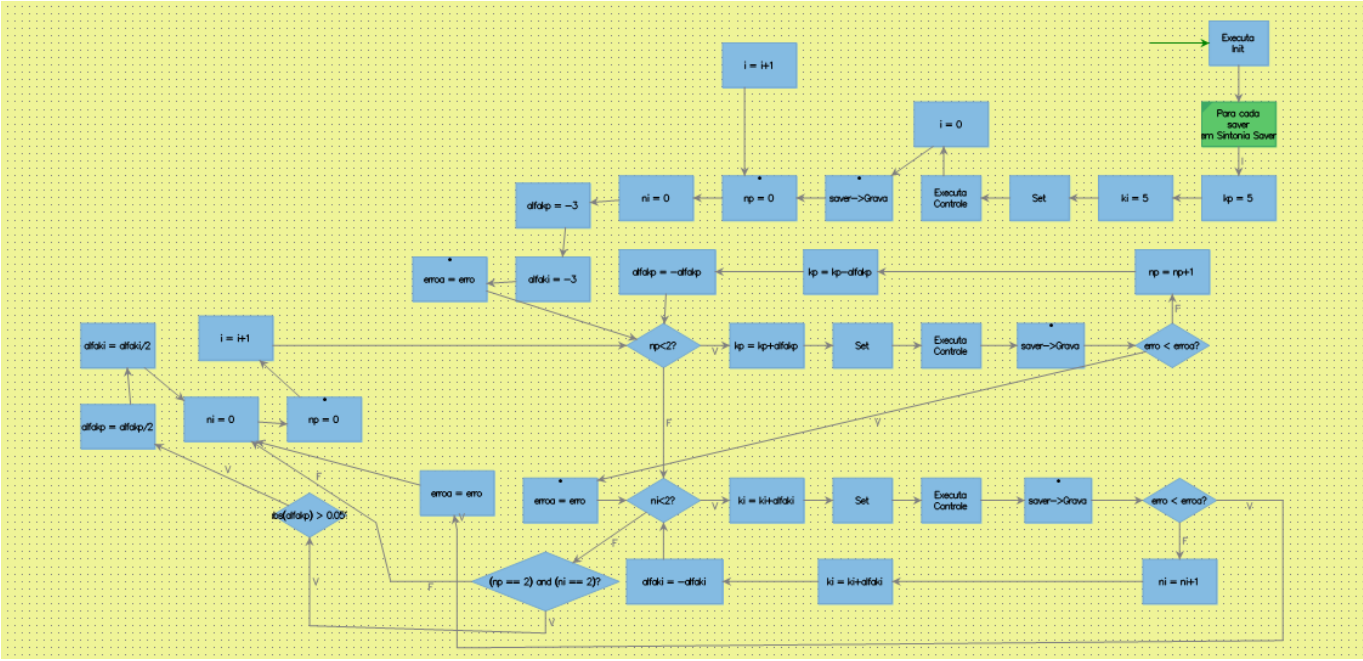

Figura 3.17: MPA fluxograma de sintonia.

Os resultados desses são gravados em 3 arquivos, um para o somatório do erro quadrático de cada sintonia , um para o kp e um para o ki da sintonia.

3.3 Aplicação 2: O Reator de Van der Vusse

Para o caso do Reator de Van der Vusse, existem algumas diferenças pequenas quanto à implementação em relação ao sistema de primeira ordem. Como a maior parte é similar, é mostrado a partir desse ponto apenas as diferenças.

3.3.1 Simulação

A simulação do Reator de Van der Vusse, descrito em [22], foi feita através de um fluxograma onde existem 3 reações $A \rightarrow B$, $B \rightarrow C$ e $2A \rightarrow D$. Para interesse de controle, utiliza-se como variável manipulada, a vazão de entrada no reator, que tem concentração pré-determinada de A, e como variável medida (controlada), a concentração de B(Cb) no reator.

As equações diferencias que regem as concentrações de cada componente são dadas por:

Balanço de massa para o componente A:

$$\tau \dot{C_a} = -\tau r_1 - \tau r_3 + C_{ain} - C_{aout}; \tag{3.4}$$

Balanço de massa para o componente B, que é a variável controlada:

$$\tau \dot{C_b} = \tau r_1 - \tau r_2 + C_{bin} - C_{bout}; \tag{3.5}$$

Balanço de massa para o componente C:

$$\tau \dot{C_c} = \tau r_2 + C_{cin} - C_{cout}; \tag{3.6}$$

E o balanço de massa para o componente D:

$$\tau \dot{C_d} = \tau r_3 + C_{din} - C_{dout}; \tag{3.7}$$

onde existem 3 taxas de reações:
r_1 é a taxa da reação $A \rightarrow B$ e é dada por:

$$k_1 e^{-E_1/T_{Out}} C_a \tag{3.8}$$

em que k_1 é o fator pré-exponencial da primeira reação e é igual a $1.287 * 10^{12} 1/h$, E_1 é a energia de ativação dessa reação e é igual a $9758.3K$, T_{Out} é a temperatura da vazão de saída do reator , e C_a é a concentração de A.

r_2 é a taxa da reação $B \rightarrow C$ e é dada por:

$$k_2 e^{-E_2/T_{Out}} C_b \tag{3.9}$$

em que k_2 é o fator pré-exponencial da segunda reação e é igual a $1.287 * 10^{12} 1/h$, E_2 é a energia de ativação dessa reação e é igual a $9758.3K$, e C_b é a concentração de B.

e r_3 é a taxa da reação $2A \rightarrow D$

$$k_3 e^{-E_3/T_{Out}} C_a^2 \tag{3.10}$$

em que k_3 é o fator pré-exponencial da segunda reação e é igual a $9.043 * 10^9 1/h$, E_3 é a energia de ativação dessa reação e é igual a $8560K$, e C_a^2 é a concentração de A ao quadrado.

O reator com esse esquema cinético tem características não lineares e, dependendo do setpoint, o sistema de controle tradicional PID não consegue controlar. Porém, um controle fuzzy, por ter características não lineares, pode ser capaz de controlar esse tipo de sistema. Embora neste projeto as características não lineares do controle fuzzy não foram exploradas em seu maior potencial, pois utiliza-se de um equivalente ao PI na lógica fuzzy, a sua não linearidade natural já traz alguma vantagem em relação ao PI tradicional.

Uma simulação do reator para a condição inicial na concentração de A não nula gera o mostrado na Figura 3.18, onde pode ser visto o valor de Cb no tempo para a concentração inicial de A no reator = 5,5 mol/L.

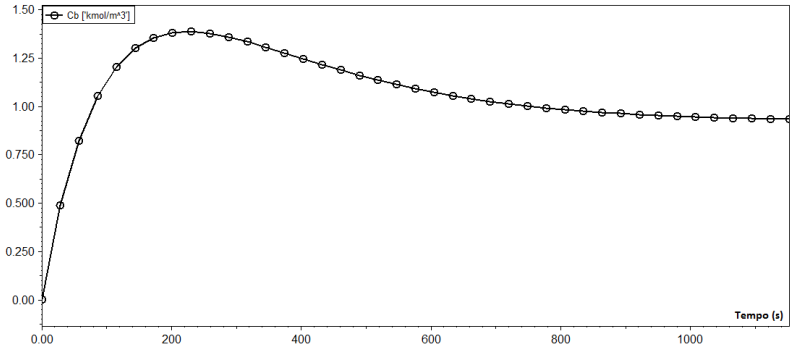

Figura 3.18: Concentração do Componente B para o reator de Van der Vusse para Valor inicial de C_a não nula.

E uma simulação para mostrar a resposta ao degrau no intervalo utilizado para setpoint neste trabalho pode ser mostrada em 3.19, em que tem, nessa região, uma resposta ainda bem linear, porém mais lenta que o sistema de primeira ordem. Em outras regiões esse sistema demonstra uma resposta de fase não mínima, ou até inversão de ganhos.

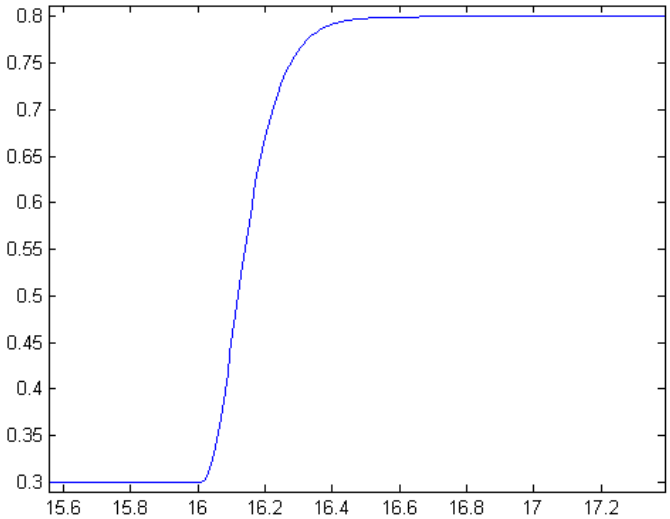

Figura 3.19: Resposta ao Degrau para a Concentração do Componente B para o reator de Van der Vusse.

Fora o modelo, o sistema de simulação é bem parecido com o caso do sistema de primeira ordem,ou seja, gera-se um flowsheet no EMSO, onde se comunica através do EMSO-OPC para os outros servidores. Não é mostrado o flowsheet do caso de

Van der Vusse, pois este é muito mais complexo e tem mais que um arquivo para a simulação, para maiores detalhes de como é feito o flowsheet consultar o apêndice A.

Além do simulado no EMSO, como no caso de primeira ordem, também é feita a simulação no MATLAB, para fins de comparação. No caso do Van der Vusse, como é um sistema mais complexo, não é possível simular apenas por uma função de transferência, então optou-se fazer um S-Function, onde pode-se simular todo o comportamento do sistema através das EDOs do reator. Nesse caso o simulink ficaria como na Figura 3.20, para maiores detalhes sobre a configuração do S-Function consultar o apêndice B.

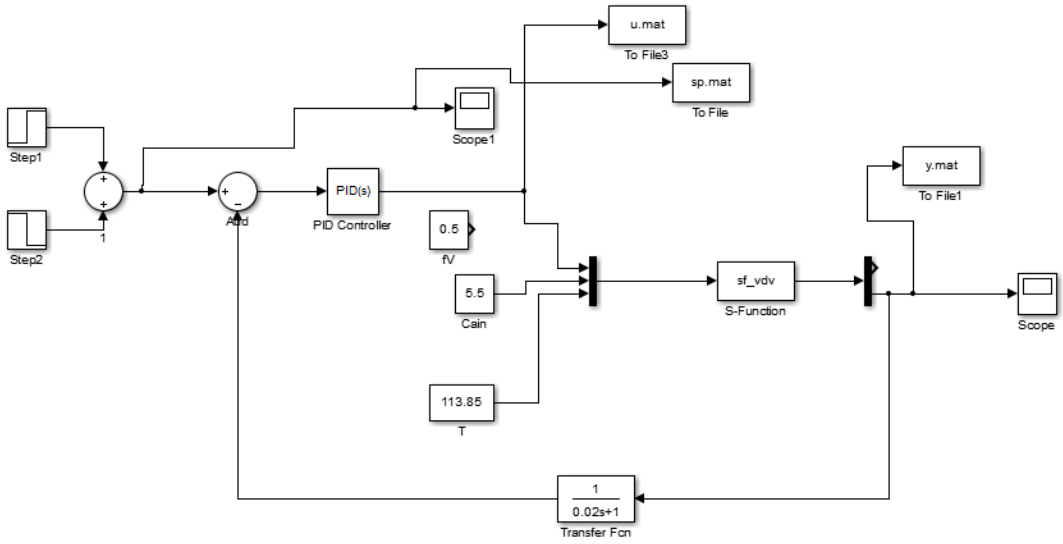

Figura 3.20: MatLab Simulink reator de Van der Vusse.

Na simulação do Van der Vusse como se pode ver pela Figura 3.20, admite-se uma dinâmica no sensor que mede o Cb, através de um filtro de primeira ordem. Isso foi feito também na simulação no EMSO, já que sempre há uma dinâmica não desprezível nos sensores e isso deve ser considerado.

3.3.2 Comunicação

A comunicação é equivalente ao sistema de primeira ordem. No EMSO-OPC, monta-se as variáveis como na Figura 3.21, onde define-se a variável controlada (Cb) e a manipulada (FVol) para a comunicação com o Top Server.

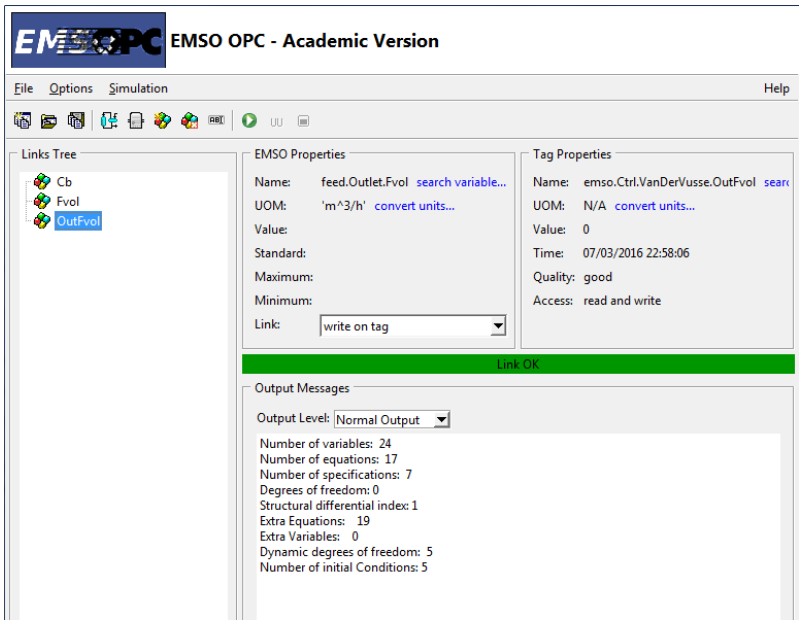

Figura 3.21: EMSO-OPC Reator de Van der Vusse.

Quanto ao Top Server, por ele ser um gerador de variáveis, simulam-se as variáveis do sistema de primeira ordem, juntamente com as do reator de Van der Vusse, então é utilizado o mesmo servidor para ambas simulações e este é configurado apenas uma vez com todas as variáveis num só servidor. Não pode ser feito o mesmo com o servidor do EMSO-OPC pois ele se conecta com o flowsheet e, porque não é possível carregar mais que um flowsheet ao mesmo tempo, então é preciso configurar servidores distintos para os dois casos.

No MPA o controle, fuzzy PI, é igual ao caso de primeira ordem, logo é preciso apenas fazer mudanças na planta, que é carregada no servidor e, então, usar o mesmo servidor que no caso anterior.

3.3.3 Controle

Assim como na comunicação, não há muita variação quanto ao controle do sistema para ambos os casos, isso se deve ao fato dele estar configurado para utilizar qualquer sistema. Ele simula um fuzzy PI, então teoricamente, pode ser utilizado para controlar qualquer sistema que um PI controlaria.

As únicas diferenças entre os controles para o caso do sistema de primeira ordem e do reator são os parâmetros utilizados. Esses parâmetros, como por exemplo, os valores máximo e mínimo de entrada e saída e valor do dt, são configurados na planta e no fluxograma de Set Point, apenas o valor de setpoint é diferente.

3.3.4 Otimização

A Otimização, assim como o controle, foi criada para sintonizar qualquer sistema. Por essa característica, ela não é configurado diferentemente para cada um dos casos, porém os valores iniciais e passos do fluxograma de Hooke & Jeeves devem ser ajustados corretamente para ser coerente com o sistema a ser executado.

Capítulo 4

Resultados e Discussões

Esse capítulo foi dividido em 3 partes. A primeira parte mostra os resultados quanto ao funcionamento puramente da lógica fuzzy no MPA. Para isso, cria-se um exemplo simples que testa a lógica fuzzy como mostrado na primeira seção do Capítulo 3, sem a parte de comunicação ou o PI implementado no MPA. Em sequência, são mostrados os resultados do controle da planta de primeira ordem, controlada pelo fuzzy PI, e, por último, o controle do reator de Van der Vusse, que é um sistema mais complexo.

4.1 Fuzzy no MPA

Essa sessão mostra o funcionamento do fuzzy no MPA e para isso, usa-se a IHM mostrada na Figura 3.3.

Com essa interface é possível simular vários casos e descobrir o risco para cada valor de pessoal e de dinheiro.

Algumas simulações foram feitas para identificar o comportamento das regras fuzzy propostas, com isso, monta-se a Tabela 4.1.

Tabela 4.1: Resultados do teste de fuzzy

Caso\Variável	Dinheiro	Pessoal	Risco
1	40 (Inadequado/Médio)	50 (Baixo/Alto)	35.8 (Baixo)
2	30 (Inadequado)	60(-Baixo/+Alto)	69.4 (Médio/Alto)
3	45(-Inadequado/+Médio)	10 (Baixo)	16.2 (Baixo)
4	50 (Médio)	80(Alto)	40 (Baixo/Médio)
5	80 (Adequado)	60 (-Baixo/+Alto)	6.4 (Baixo)

Com esses resultados, é possivel ver que a ação passa pelas regras fuzzy e defuzzifica através do método da centróide, de forma satisfatória, gerando um valor de risco que pode ser usado para controle ou qualquer outra ação que desejar.

Como previsto, pelas regras fuzzy criadas, quando se tem um valor de dinheiro inadequado, o risco é mais alto e quando se tem um valor de dinheiro adequado, o risco é mais baixo. Assim como quando o valor de pessoal é maior, o risco é mais alto. E quando o valor de pessoal é menor, o risco é mais baixo.

4.2 Sistema de Primeira Ordem

Para demonstrar os resultados do controle de primeira ordem foram feitas várias sintonias e mostra-se como o sistema age na melhor sintonização. Uma comparação entre o controle fuzzy e o PI tradicional, pode ser vista na Figura 4.1. O MPA segue o setpoint, assim como o controle tradicional, porém com um pequeno atraso e um overshoot um pouco maior. Esses resultados ligeiramente piores, no caso do controle no MPA são esperados, já que é um caso simples e o PI tradicional é bem eficaz para o controle de sistemas de primeira ordem, além disso, o atraso do fuzzy pela dinâmica de comunicação existe, mesmo sendo pequeno. A sintonia do MatLab foi feita através de uma minimização do erro quadrático por otimização, assim como a sintonia do MPA.

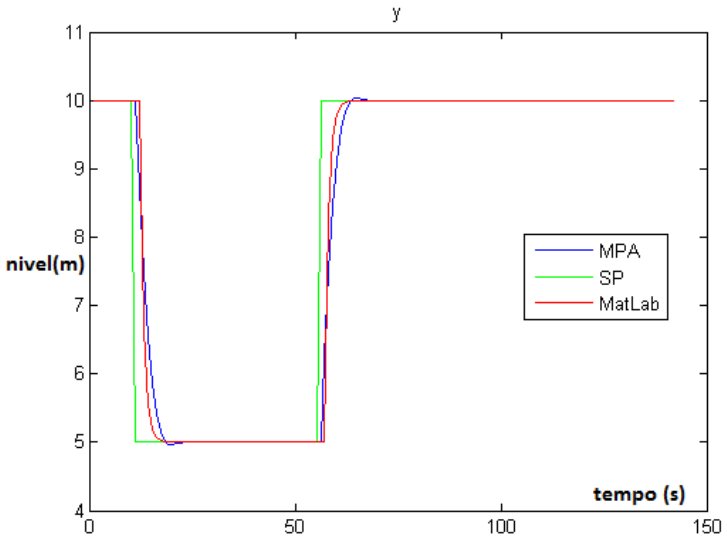

Figura 4.1: Comparação de Resultados Primeira Ordem.

Pode-se ver que se for colocado um atraso no MatLab, na entrada do sistema como na Figura 4.2, gera-se uma resposta muito pior que o sistema fuzzy, que seria instável para atrasos grandes. Para pequenos valores de atraso, podem-se ver os resultados da resposta na Figura 4.3.

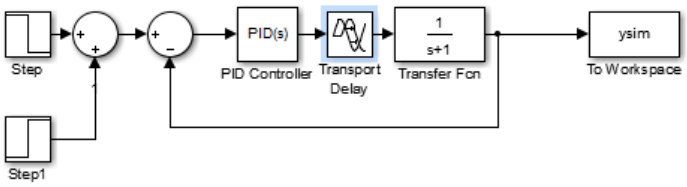

Figura 4.2: Sistema de Primeira Ordem com Atraso.

É visto na Figura 4.3 que o overshoot é maior no caso de atraso na simulação no MatLab em relação ao controle fuzzy aplicado para a mesma planta. Esse atraso da comunicação não é muito grande, porém como demonstrado, um atraso pequeno pode atrapalhar bastante a sintonia. Com isso a comparação fica dividida em com e sem atraso.

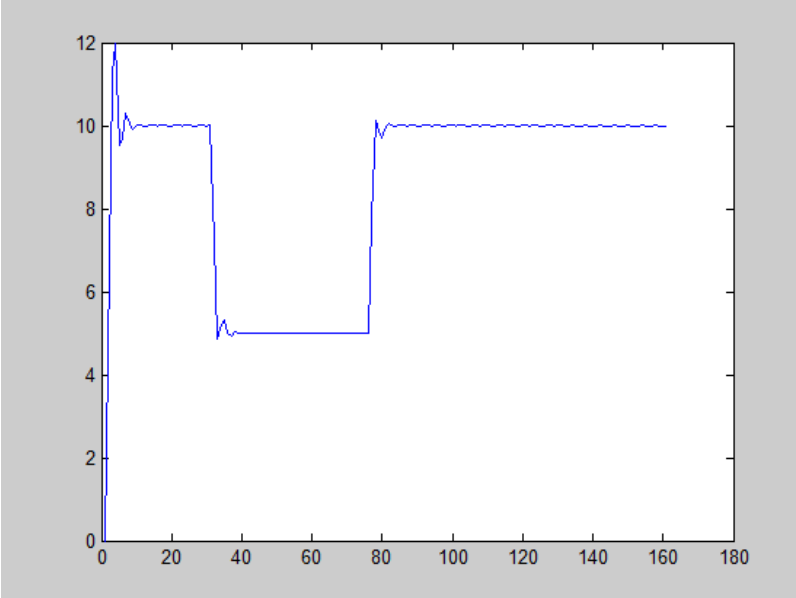

Figura 4.3: Resultado do PI tradicional para o caso de primeira ordem com atraso.

4.3 Sistema de Reator de Van der Vusse

Essa seção abrange não apenas a análise do PI aplicado em um reator com cinética de Van der Vusse, mas também descreve uma discussão sobre a melhoria da sintonia do controle através do sintonizador que utiliza o algoritmo de Hooke & Jeeves, descrito no item 3.2.4.

Para testar o sintonizador, implementado no MPA, foi colocada inicialmente uma sintonia ruim, valores de Kp e Ki muito altos, onde $Kp = 25$ e $Ki = 20$. Essa

sintonia foi escolhida para mostrar a melhoria gerada pelo fluxograma de sintonia no controle do sistema.

Com essa sintonia, foi gerada uma resposta no tempo mostrada pelo gráfico da Figura 4.4. É notável que o controle está se perdendo por seus altos ganhos gerando uma resposta oscilatória, embora não esteja instável.

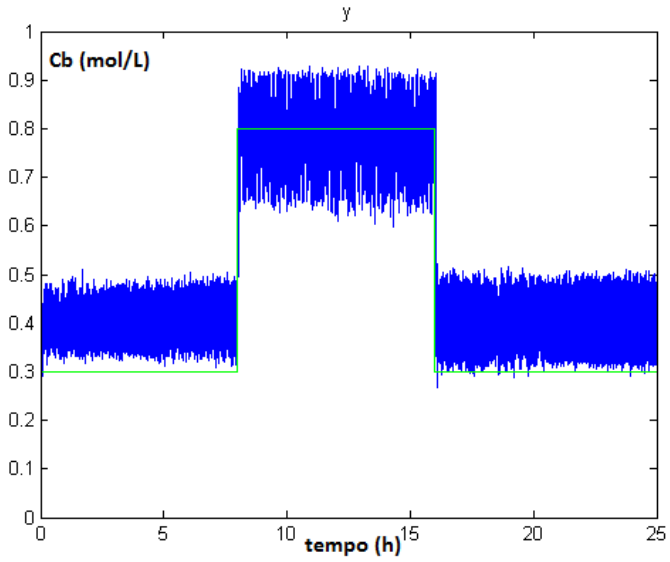

Figura 4.4: Van der Vusse MPA variável controlada: sintonia ruim.

Além da resposta ruim na variável controlada, pode-se ver também, na variável manipulada, um sistema muito agressivo mostrado pela Figura 4.5.

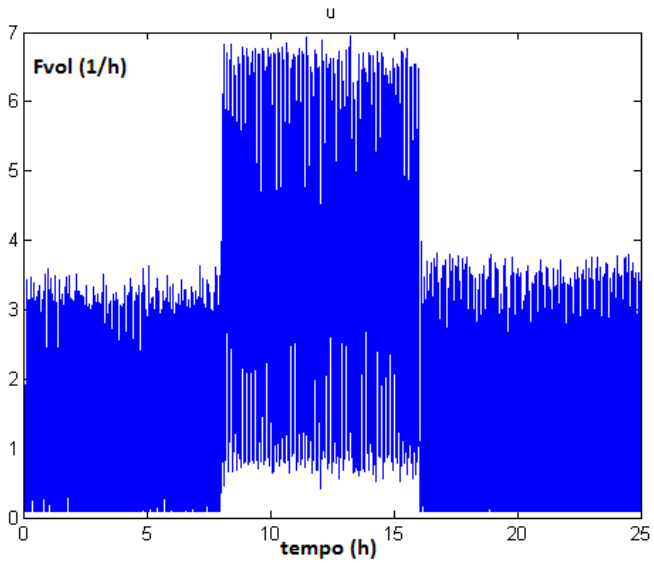

Figura 4.5: Van der Vusse MPA variável manipulada: sintonia ruim.

Com essa sintonia inicial ruim, é executado o algoritmo para minimizar o erro

quadrático, alterando os valores de Kp e de Ki. Como resultado, tem-se os valores de $Kp = 9.6875$ e de $Ki = 1.4375$, após 71 iterações.

Com essa nova sintonia, tem-se os resultados de u e de y representados na figura 4.6. Nesta, nota-se que a agressividade do sistema foi reduzida e que o sistema estabilizou-se, mostrando, então, que o processo de sintonia utilizando o Hooke & Jeeves consegue gerar uma sintonia muito superior à inicial.

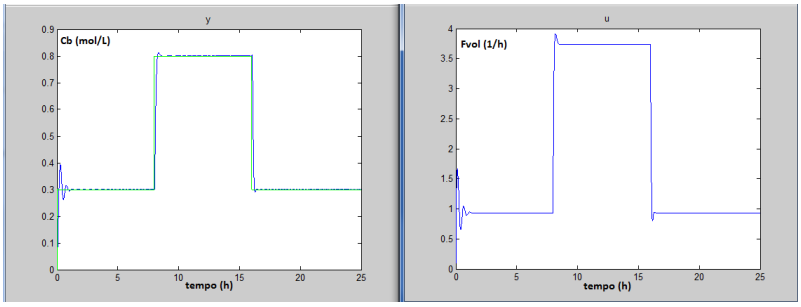

Figura 4.6: Van der Vusse: sintonia boa.

Similar aos resultados do caso de primeira ordem, os controles, fuzzy e PI tradicional, conseguem seguir a referência para o sistema do reator de Van der Vusse, para os SPs estudados. Porém, por ser um sistema de ordem maior, o controle PI no MatLab demonstra uma dificuldade para controlá-lo. O controlador fuzzy no MPA, por sua natureza não linear, consegue gerar ganhos em relação ao PI tradicional, também sintonizado por um algoritmo baseado em otimização. A comparação é demonstrada pela figura 4.7, onde vemos, que a resposta no Fuzzy PI não foi apenas mais rápida, como também teve um menor overshoot e oscilou menos.

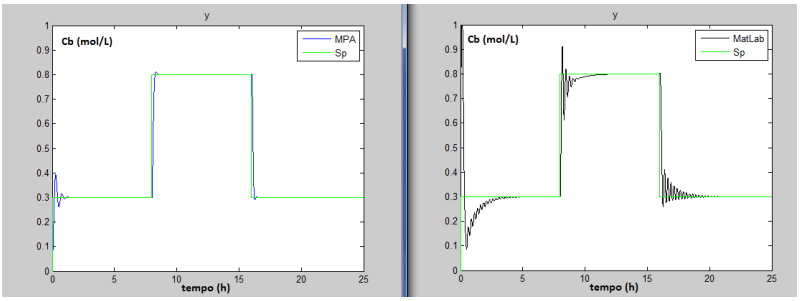

Figura 4.7: Resultado de comparação variável controlada Van der Vusse.

É notável, também, uma melhora quanto à variável manipulada do sistema, embora não seja o foco do trabalho e não seja considerada como parâmetro para minimização nesse projeto. Pode-se ver a diferença entre as variáveis manipuladas na Figura 4.8, onde a variável de controle para esse caso foi menor e menos oscilatória, gastando menos carga e comprometendo menos os equipamentos de atuação.

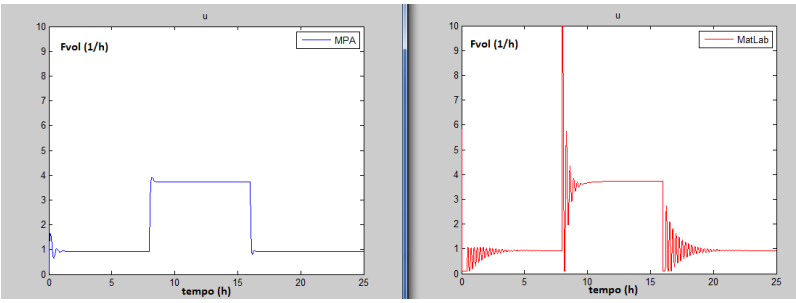

Figura 4.8: Resultado de comparação variável manipulada Van der Vusse.

Para a aquisição desses resultados, foram utilizadas sintonias do PI tradicional, após a otimização dos parâmetros para minimização do erro quadrático, assim como no caso de primeira ordem. Porém, é possível, gerar uma sintonia que não oscile tanto. A variável controlada de uma sintonia lenta no PI tradicional é mostrada na figura 4.9. Como essa sintonia é muito mais lenta que a gerada pelo PI fuzzy, temos um resultado melhor pelo controle fuzzy em relação ao PI tradicional em ambas as sintonias.

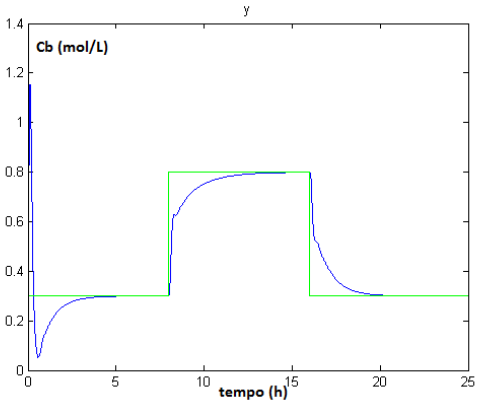

Figura 4.9: Van der Vusse variável controlada para uma sintonia lenta no PI tradicional.

É notável, pela figura 4.10, que essa diminuição da oscilação também é sentida na variável manipulada. Quanto a essa variável, é melhor que ela seja lenta, pois assim afeta menos os equipamentos de controle. Entretanto, essa melhora não compensa o pior resultado da variável controlada.

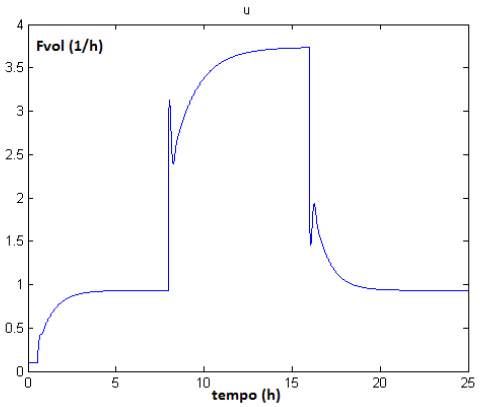

Figura 4.10: Variável manipulada para uma sintonia lenta no PI tradicional para o Reator de Van der Vusse.

Capítulo 5

Conclusões e Trabalhos Futuros

5.1 Conclusões

Quanto à implementação do fuzzy, o programa criado em MPA funciona como esperado. Como mostrado no item 4.1, é executada com êxito uma lógica fuzzy no MPA, que gera uma saída através do cálculo da centróide.

É mostrado em detalhes um sistema interno de comunicação entre simulação e controle em diferentes softwares dentro de um mesmo computador, gerando um ambiente propício para o desenvolvimento de novos controles.

O controle fuzzy consegue seguir a referência, gerando um resultado melhor que o PI tradicional para sistemas complexos. Entretanto, ainda possui um resultado ligeiramente pior que os controles tradicionais em sistemas de primeira ordem.

No que diz respeito à sintonia do fuzzy PI no MPA, houve ganhos significativos quando a sintonia inicial, e assim, com uma lógica de minimização do erro quadrático é gerado um controle sintonizado após a aplicação do Hooke & Jeeves.

Além disso, obteve-se um melhor desempenho na variável manipulada, o que não era esperado, já que não foi considerado nenhum fator no critério de minimização que levasse em conta essa variável. A verificação da variável manipulada é sempre interessante para casos reais, por isso, acrescentou-se também essa análise na parte dos resultados, no Capítulo 4, porém sem uma grande ênfase nessa variável, pois o foco desse trabalho é a variável controlada.

Pode-se concluir, então, que a lógica fuzzy implementada no MPA está funcionando como deveria. Porém, o controle PI fuzzy que foi implementado não teve tanto êxito, quando comparado a um controlador PI tradicional implementado no MatLab, no caso de sistema de primeira ordem. Ainda é notável, que a sintonia do controlador consegue melhoras significativas e que pode ser utilizada para melhorar o controle do sistema. Assim, o controle fuzzy, para sistemas mais complexos, caso do reator de Van der Vusse, consegue um desempenho melhor que o PI tradicional.

5.2 Sugestões para Trabalhos Futuros

Seria interessante estudar outros algoritmos de controle fuzzy, e não implementando um PI fuzzy, poderia-se também englobar um método de sintonia para esse tipo de sistema, que é muito mais complicado, pois haverá uma quantidade de parâmetros muito grande. Os ganhos desse controle diretamente pelo fuzzy deve ser interessante pelo fato de que introduziria um fator completamente não linear no controle e assim teria um potencial muito grande para melhorias. A sintonia de um controle desse tipo poderia ser uma extensão dessa sintonia por Hooke & Jeeves, englobando mais parâmetros.

Existem uma grande quantidade de trabalhos ainda a serem desenvolvidos como continuação desse projeto, essa implementação do fuzzy pode servir de alavanca para gerar um controle fuzzy muito mais completo para a melhoria de sistemas reais e com suas características não lineares, trazendo grandes ganhos para a industria.

Referências Bibliográficas

[1] ZADEH, L. "Fuzzy sets", *Information and Control*, v. 8, n. 3, pp. 338–353, 1965.

[2] ROISENBERG, M., RECH, L. "Lógica Fuzzy (Lógica Nebulosa)", Disponível em: <www.inf.ufsc.br/~alvares/INE5633/Fuzzy.ppt>.

[3] DAY, C. P., DUMMERMUTH, E. H. "Logical Industrial Controls Using Fuzzy Rules", *IEEE Conference Publications*, 1993.

[4] NOLASCO, D. H. S., PALMEIRA, E. S. "Controlador fuzzy para diagnóstico da qualidade de energia elétrica em níveis de baixa tensão - uma comparação entre as inferências de mandani clássico e de hamacher", *XVIII ENMC*, 2015.

[5] RIBEIRO, E. L. F., OLIVEIRA, F. B. S., MARANDUBA, H. L., et al. "Modelo dinâmico-fuzzy para análise da viabilidade do biodiesel de pinhão manso", *XVIII ENMC*, 2015.

[6] MACHADO, E. R. M. D., RAMOS, I. T. M., KOZAN, R. F., et al. "Modelagem e controle do sistema biológico da junção do joelho com o conjunto perna-pé com modelos fuzzy ts", *XVIII ENMC*, 2015.

[7] EKLUND, P. "Fuzzy logic in Northern Europe: industrial applications and software developments", *Proceedings of 1994 IEEE 3rd International Fuzzy Systems Conference*.

[8] PRECUP, R.-E., HELLENDOORN, H. "A survey on industrial applications of fuzzy control", *Computers in Industry*, v. 62, n. 3, pp. 213–226, 2011.

[9] MENDONÇA, E. S. *Monitoramento, Diagnóstico e otimização operacional de uma unidade de processamento de gáz natural*. Tese de Mestrado, Programa de pós-graduação em tecnologia de processos químicos e bioquímicos - UFRJ, 2007.

[10] DE CAMPOS, M. C. M. M., DE CARVALHO GOMES, M. V., PEREZ, J. M. G. T. *Controle Avançado e Otimização na Indústria do Petróleo*. INTERCIÊNCIA, 2013.

[11] WOLKENHAUER, O. "Fuzzy Control PI vs. Fuzzy PI-Control", Disponível em: <https://www.sbi.uni-rostock.de/uploads/tx_templavoila/SBI_Materials_Fuzzy-Control.pdf>.

[12] DING, Y., YING, H., SHAO, S. "Typical Takagi–Sugeno PI and PD fuzzy controllers: analytical structures and stability analysis", *Information Sciences*, 2002.

[13] ANDRZEJ, P. "Tunning fuzzy model parameters with stochastic algorithms", *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2011.

[14] DE ANDRADE, J. F. B. *Sintonia de controlador fuzzy por algoritmo genético em sistema de nível de líquidos.* Tese de Mestrado, Programa de Pós Graduação em Engenharia Elétrica Universidade federal do pará, 2014.

[15] PADILHA, P. C. C. *Desenvolvimento de uma metodologia de sintonia de controladores fuzzy utilizando redes neurais aplicações em processos petroquímicos.* Tese de Mestrado, IME, 2001.

[16] OGATA, K. *Modern control engineering.* Prentice-Hall, 1970.

[17] VODA, A., LANDAU, I. "A method for the auto-calibration of PID controllers", *Automatica*, v. 31, n. 1, pp. 41–53, 1995.

[18] TIZZO, L. M., OLIVEIRA-LOPES, L. C. "Otimização com multiobjetivos aplicada à sintonia de controladores preditivos", *VIII Congresso Brasileiro de Engenharia Química em Iniciação Científica*, 2015.

[19] DO P. NUNES, L. E. N., ROSADO, V. O. G., GRANDINETTI, F. J. "Ajuste dos parâmetros de um controlador proporcional, integral e derivativo através de algoritmos genéticos", *Revista de Ciências Exatas, Taubaté*, v. 9/10, n. 1-2, pp. 47–52, jan. 2004.

[20] DE ARRUDA, L. V. R., NEVES-JR, F., SWIECH, M. C. S., et al. "Um método evolucionário para sintonia de controladores PI/PID em processos multivariáveis", *Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial Universidade Tecnológica Federal do Paraná*, mar. 2008.

[21] "Hooke and Jeeves (Hooke Algorithm". 2016. Disponível em: <http://files.vlsi.uwindsor.ca/88-521/pdf/Hooke.PDF>.

[22] DO BRASIL MAGALHÃES, O. I. *Desenvolvimento de um sistema de otimização dinâmica em tempo real*. Tese de Mestrado, Programa de Pós-graduação em Engenharia Química, COPPE, UFRJ., 2010.

Apêndice A

Código de Fluxograma EMSO (reator de Van der Vusse)

A.1 vdv.mso

```
using "cstr_vdv_T";

FlowSheet VanDerVusse

DEVICES
feed  as Source_conc  (Brief = "Feed stream");
reactor  as cstr_vdv (Brief = "Cstr reactor");
product  as Sink_conc (Brief = "Product");

CONNECTIONS
feed.Outlet  to reactor.Inlet;
reactor.Outlet  to product.Inlet;

PARAMETERS
T as time_h;

VARIABLES
Cb as conc_mol  (Brief ="Molar concentration component B", Lower = 0, Symbol = "C_B
Fvol as frequency  (Brief="Flow rate", DisplayUnit='1/h');
# Fvol as flow_vol  (Brief="Flow rate", DisplayUnit='m^3/h');
```

```
SET
reactor.k1  = 1.287e12 * '1/h';
reactor.k2  = 1.287e12 * '1/h';
reactor.k3  = 9.043e9 * 'l/mol/h';
reactor.E1  = 9758.3 * 'K';
reactor.E2  = 9758.3 * 'K';
reactor.E3  = 8560 * 'K';
reactor.V  = 2 * 'm^3';
T = 0.05 * 'h';

SPECIFY
#"Inlet Molar Concentration Component A"
feed.Outlet.Ca = 5.5 * 'mol/l';

#"Inlet Molar Concentration Component B"
feed.Outlet.Cb = 0 * 'mol/l';

#"Inlet Molar Concentration Component C"
feed.Outlet.Cc = 0 * 'mol/l';

#"Inlet Molar Concentration Component D"
feed.Outlet.Cd = 0 * 'mol/l';

#"Temperature"
feed.Outlet.T = 387 * 'K';

#"Temperature"
feed.Outlet.P = 1 * 'atm';



#"Flow Rate"
Fvol = 5 * '1/h';
# Fvol = 10 * 'm^3/h';

EQUATIONS
feed.Outlet.Fvol = Fvol*reactor.V;
# feed.Outlet.Fvol = Fvol;

T*diff(Cb) = reactor.Cb - Cb;
```

```
INITIAL

Cb = 0 * 'mol/l';
#"Flow"
#feed.Outlet.Fvol = 1 * 'm^3/h';

"Molar Concentration Component A"
reactor.Ca = 5 * 'mol/l';

"Molar Concentration Component B"
reactor.Cb = 0 * 'mol/l';

"Molar Concentration Component C"
reactor.Cc = 0 * 'mol/l';

"Molar Concentration Component D"
reactor.Cd = 0 * 'mol/l';

OPTIONS
TimeStep = 0.008;
TimeEnd = 0.32;
TimeUnit = 'h';
Dynamic = true;
DAESolver(File="dasslc", RelativeAccuracy=1e-6, AbsoluteAccuracy=1e-6);
end
```

A.2 cstr_vdv_T.mso

```
using "simple_streams";

Model stream_conc as MaterialStream

ATTRIBUTES
Pallete  = false;
Brief  = "Material stream with molar concentration.";
Info  =
"== Contents ==
*Ca: Molar Concentration of Component A
*Cb: Molar Concentration of Component B
*Cc: Molar Concentration of Component C
*Cd: Molar Concentration of Component D
";

VARIABLES

Ca    as conc_mol
   (Brief = "Molar concentration component A", Lower = 0, Symbol = "C_A");
Cb    as conc_mol
   (Brief = "Molar concentration component B", Lower = 0, Symbol = "C_B");
Cc    as conc_mol
   (Brief = "Molar concentration component C", Lower = 0, Symbol = "C_C");
Cd    as conc_mol
   (Brief = "Molar concentration component D", Lower = 0, Symbol = "C_D");

end

Model Source_conc

ATTRIBUTES
Pallete  = true;
Icon  = "icon/stream_conc";
Brief  = "Source stream with molar concentration.";
Info  =
"== The user should specify ==
```

```
* Total Volumetric Flow
* Temperature
* Pressure
*Ca
*Cb
*Cc
*Cd
";

VARIABLES

out Outlet as stream_conc  (Brief = "Feed Stream", PosX=1, PosY=0.5256);

end

Model Sink_conc

ATTRIBUTES
Pallete  = true;
Icon  = "icon/stream_conc";
Brief  = "Sink stream with molar concentration.";
Info  =
"== Contents ==
* Total Volumetric Flow
* Temperature
* Pressure
*Ca
*Cb
*Cc
*Cd
";

VARIABLES

in Inlet as stream_conc  (Brief = "Product Stream", PosX=0, PosY=0.5256);

end

Model cstr_vdv
```

```
ATTRIBUTES
Pallete  = true;
Icon  = "icon/cstr";
Brief  = "Model of a cstr with Van Der Vusse Reaction.";
Info  =
"== Assumptions ==
* the reactor is a well mixed element;
* thermal and mechanical equilibrium;
* only designed for Van Der Vusse Reaction;

== Specify ==
* the Inlet stream;

== Setting Parameters ==
*Reaction rate constant (k1);
*Reaction rate constant (k2);
*Reaction rate constant (k3);
*Reactor Volume (V);

== Initial Conditions ==
* the components initial molar concentration inside reactor (Ca,Cb,Cc,Cd);
";

PARAMETERS

k1    as frequency
   (Brief="Reaction rate constant", DisplayUnit='1/h', Symbol = "k_1");
k2   as frequency
   (Brief="Reaction rate constant", DisplayUnit='1/h', Symbol = "k_2");
k3   as positive
   (Brief="Reaction rate constant", Unit='l/mol/h', Symbol = "k_3");
E1    as temperature  (Brief="Activation energy", Symbol = "E_1");
E2   as temperature  (Brief="Activation energy", Symbol = "E_2");
E3   as temperature  (Brief="Activation energy", Symbol = "E_3");
V as volume (Brief ="Reactor volume");
```

```
VARIABLES
Ca as conc_mol
   (Brief ="Molar concentration component A", Lower = 0, Symbol = "C_A^{cstr}");
Cb as conc_mol
   (Brief ="Molar concentration component B", Lower = 0, Symbol = "C_B^{cstr}");
Cc as conc_mol
   (Brief ="Molar concentration component C", Lower = 0, Symbol = "C_C^{cstr}");
Cd as conc_mol
   (Brief ="Molar concentration component D", Lower = 0, Symbol = "C_D^{cstr}");
tau as time_h
   (Brief ="Reactor mean residence time", Symbol = "\tau");
r1   as reaction_mol (Brief ="Reaction rate", Upper=1e20, Symbol = "r_1");
r2   as reaction_mol (Brief ="Reaction rate", Upper=1e20, Symbol = "r_2");
r3   as reaction_mol (Brief ="Reaction rate", Upper=1e20, Symbol = "r_3");

in   Inlet   as stream_conc
   (Brief="Inlet stream",PosX=0, PosY=0, Symbol = "^{inlet}");
out  Outlet  as stream_conc
   (Brief="Outlet stream", PosX=1, PosY=1, Symbol = "^{outlet}");

EQUATIONS

"Overall Mass Balance"
Inlet.Fvol = Outlet.Fvol;

"Reaction Rate - First Reaction"
r1 = k1*exp(-E1/Outlet.T)*Ca;

"Reaction Rate - Second Reaction"
r2 = k2*exp(-E2/Outlet.T)*Cb;

"Reaction Rate - Third Reaction"
r3 = k3*exp(-E3/Outlet.T)*Ca*Ca;

"A Component Mass Balance"
tau*diff(Ca) = -tau*r1 - tau*r3 + Inlet.Ca - Outlet.Ca;

"B Component Mass Balance"
tau*diff(Cb) = tau*r1 - tau*r2 + Inlet.Cb - Outlet.Cb;
```

```
"C Component Mass Balance"
tau*diff(Cc) = tau*r2 + Inlet.Cc - Outlet.Cc;

"D Component Mass Balance"
tau*diff(Cd) = tau*r3/2 + Inlet.Cd - Outlet.Cd;

"Well Mixed - A component"
Outlet.Ca = Ca;

"Well Mixed - B component"
Outlet.Cb = Cb;

"Well Mixed - C component"
Outlet.Cc = Cc;

"Well Mixed - D component"
Outlet.Cd = Cd;

"Residence Time"
tau * Inlet.Fvol = V;

"Thermal equilibrium"
Outlet.T = Inlet.T;

"Mechanical equilibrium"
Outlet.P = Inlet.P;

end
```

Apêndice B

Código de S-Function para o MatLab (Reator de Van der Vusse)

```
% Notação     :   x(1) = ca
%                 x(2) = cb
%                 u(1) = f = F/V
%                 u(2) = ca0
%                 u(3) = T_reator
%
%   Kr=[k1  EA1
%       k2  EA2
%       k3  EA3  ]
%z.B.:
%     Kr=[
%        1.287e12  -9758.3
%        1.287e12  -9758.3
%        9.043e09  -8560   ]
%
%    Konstante=[
%        Vr  = 2;
%    ]
%


function [sys, x0] = cstr(t,x,u,flag)

global Kr %V

if abs(flag) == 1
% derivatives
```

```
28
29 r=Kr(:,1).*exp(Kr(:,2)./(u(3)+273.15)).*[x(1); x(2);  x(1)^2];
30 sys(1) = u(1)*(u(2) - x(1)) - r(1) - r(3);
31 sys(2) = -u(1)*x(2) + r(1) - r(2);
32
33 %   dca = u(1).*(u(2) - x(1)) - k1*x(1) - k3*(x(1).^2);
34 %   dcb = -u(1).*x(2) + k1*x(1) - k2*x(2);
35
36
37 elseif flag == 3
38 % output
39 sys=x;
40
41 elseif flag == 0
42 % initialization
43 Kr=[
44 1.287e12  -9758.3
45 1.287e12  -9758.3
46 9.043e09  -8560   ];
47
48 sys = [2;0;2;3;0;0];
49
50 %     V = 20;
51 % cálculo do estacionário
52 x0 = [5,0];
53
54 else
55 sys = [];
56
57 end
```

Apêndice C

Códigos de implementação de fuzzy no MPA

C.1 MPA fuzzy

```
func{ id = "mpa_esperar", name = "MPA esperar",
description = [[Faz uma pausa na execução do fluxo]],
parameters = {
{ name = "Tempo em segundos para a espera do fluxo", type = "REAL" },
},
results = {
},
code = [=============================================================[
function(time)
-- TODO: check it with Cassino.
sleep(time)
end
]=============================================================],
}

func{ id = "centroid",
name        = "Centroid method",
description = [[Calculate the centroid given a fuzzyset]],
parameters = {{ name = "fs", type = "fuzzyset" },},
results = {
{ name = "cen", type = "REAL" },
},
code = [=============================================================[
```

```
function( fs )
local x = fs.x
local y = fs.y
local xval
local yval
if x:informar_tamanho() <= 0 then
error('invalid number of fuzzyset entries')
end
if x:informar_tamanho() ~= y:informar_tamanho() then
error('invalid number of fuzzyset entries')
end
local accxy = 0
local accy = 0
for i =1,x:informar_tamanho() do
local xval = tonumber(x:informar(i))
local yval = tonumber(y:informar(i))
if yval > 0 then
accxy = accxy + xval*yval
accy = accy + yval
end
end
if accy > 0 then
return accxy/accy
else
return (tonumber(x:informar(1))+tonumber(x:informar(x:informar_tamanho())))/2
end
end
]============================================================]
}

func{ id = "tmin", name = "tmin",
description = [[achar o minimo de dois valores]],
parameters = {{ name = "a1", type = "REAL" },
{ name = "a2", type = "REAL" },
},
results = {{ name = "min", type = "REAL" },},
code = [=[function(a1,a2)
if a2 < a1 then
return a2
```

```
else
return a1
end
end]=],
}

func{ id = "tmax", name = "tmax",
description = [[achar o maximo de dois valores]],
parameters = {{ name = "a1", type = "REAL" },
{ name = "a2", type = "REAL" },
},
results = {{ name = "max", type = "REAL" },},
code = [=[function(a1,a2)
if a2 < a1 then
return a1
else
return a2
end
end]=],
}

func{ id = "tsum", name = "tsum",
description = [[achar a soma de dois valores]],
parameters = {{ name = "a1", type = "REAL" },
{ name = "a2", type = "REAL" },
},
results = {{ name = "sum", type = "REAL" },},
code = [=[function(a1,a2)
local y = a1 + a2
return y -- stop in one(limit)?
end]=],
}

func{ id = "prod", name = "tprod",
description = [[achar o produto de dois valores]],
parameters = {{ name = "a1", type = "REAL" },
{ name = "a2", type = "REAL" },
},
results = {{ name = "sum", type = "REAL" },},
```

```
code = [=[function(a1,a2)
return a1*a2
end]=],
}

class{ id = "inferencefunction", name = "Inference Function", group = "Fuzzy",
attributes = {
{id = "tipo", name = "Type", type = "STRING"},
{id = "params_1", name = "Parametro 1" , type = "REAL"},
{id = "params_2", name = "Parametro 2" , type = "REAL"},
{id = "params_3", name = "Parametro 3" , type = "REAL"},
{id = "params_4", name = "Parametro 4" , type = "REAL"},
},
methods = {
{ id = "calc_inf",
name        = "Calculate Inference",
description = [[Calculate the inference for a given valor.]],
parameters = {{ name = "val", type = "REAL" },
},
results = {
{ name = "inf", type = "REAL" },
},
code = [============================================================[
function(self, val)
local x = val
local params = {self.params_1,self.params_2,self.params_3,self.params_4}
if self.tipo == "gauss" then
return math.exp( -((x - params[2])^2)/(2*params[1]^2))
elseif self.tipo == "trape" then
if x > params[1] and x < params[2] then
return (x-params[1])/(params[2]-params[1])
elseif x >= params[2] and x < params[3] then
return 1
elseif x >= params[3] and x < params[4] then
return (params[4]-x)/(params[4]-params[3])
else
return 0
end
elseif self.tipo == "triang" then
```

```
if x > params[1] and x < params[2] then
return (x-params[1])/(params[2]-params[1])
elseif x >= params[2] and x < params[3] then
return  (params[3]-x)/(params[3]-params[2])
else
return 0
end
else
return "error wrong type";
end
end
]============================================================],
},
},
}

class{ id = "lingvar", name = "Variavel Linguistica", group = "Fuzzy",
attributes = {
{ id = "value", name = "Value", type = "REAL",access = "gs"},
{ id = "name", name = "Name", type = "STRING",access = "g"},
{ id = "mf", name = "mf", type = "inferencefunction",access = "g"},
},
methods = {
},
}

class{ id = "input_var", name = "Variavel Input Fuzzy", group = "Fuzzy",
attributes = {
{ id = "value", name = "Value", type = "REAL",access = "gs"},
{ id = "name", name = "Name", type = "STRING",access = "g"},
{ id = "mn", name = "mn", type = "REAL",access = "g"},
{ id = "mx", name = "mx", type = "REAL",access = "g"},
{ id = "lingvar", name = "lingvar", type = "lingvar"},
{ id = "lingvar2", name = "lingvar2", type = "lingvar"},
{ id = "lingvar3", name = "lingvar3", type = "lingvar"},
{ id = "lingvar4", name = "lingvar4", type = "lingvar"},
},
methods = {
```

```
},
}

class{ id = "output_var", name = "Variavel Output Fuzzy", group = "Fuzzy",
attributes = {
{ id = "value", name = "Value", type = "REAL",access = "gs"},
{ id = "name", name = "Name", type = "STRING",access = "g"},
{ id = "fuzzyset", name = "fuzzyset", type = "fuzzyset",access = "g"},
{ id = "mn", name = "mn", type = "REAL",access = "g"},
{ id = "mx", name = "mx", type = "REAL",access = "g"},
{ id = "lingvar", name = "lingvar", type = "lingvar"},
{ id = "lingvar2", name = "lingvar2", type = "lingvar"},
{ id = "lingvar3", name = "lingvar3", type = "lingvar"},
{ id = "lingvar4", name = "lingvar4", type = "lingvar"},
{ id = "lingvar5", name = "lingvar5", type = "lingvar"},
},
methods = {
{ id = "calc",
name        = "Calcula Output",
description = [[Calcula um valor de output baseado nas listas]],
parameters = {
},
results = {
},
code = [===[ function(self)
self.value = centroid(self.fuzzyset)
end ]===],
},
},
}

class{ id = "fuzzyset", name = "Fuzzy Set", group = "Fuzzy",
attributes = {
{ id = "y", name = "y", type = "REAL_LIST",access = "g"},
{ id = "x", name = "x", type = "REAL_LIST",access = "g"},
},
methods = {
},
}
```

```
class{ id = "REAL_LIST", name = "Lista de REAIS", group = "Blocos de Cálculo",
bases = {},
description = [[Equipamento para armazenar uma lista de valores reais
como strings em disco para que possam ser carregados em uma nova sessão.
A lista é carregada automaticamente ao inicilaizar a instância da classe e
salva ao inserir ou remover um elemento.]],
attributes = {{ id = "at", name = "Atualizada", type = "BOOLEAN",access = "gs"},
{ id = "plot", name = "Is Plot", type = "BOOLEAN",access = "gs"},
},
methods = {
{ id = "inserir",
name        = "Inserir",
description = [[Insere um novo elemento no final da lista.
Os valores são armazenados como texto.]],
parameters = {
{ name = "Novo Valor", type = "REAL" },
},
results = {
},
code = [===[ function(self, valor)
val = tostring(valor)
if self.at then
x = tonumber(self:informar(1))
xf = tmax(valor,x)
self:remover()
val1 = tostring(xf)
table.insert(self._valores,val1)
else
table.insert(self._valores, val)
end

self:save()
end ]===],
},
{ id = "remover",
name        = "Remover",
description = [[Remove o elemento na posição informada e retorna.
Se nenhuma posição for informada, o primeiro elemento da lista será removido.]],
```

```
parameters = {
{ name = "Posição", type = "INTEGER" },
},
results = {
},
code = [===[ function(self, posicao)
local ret =  table.remove(self._valores, posicao or 1)
self:save()
return ret
end ]===],
},
{ id = "clear",
name        = "limpar",
description = [[deleta todos os valores da lista]],
parameters = {
},
results = {
},
code = [===[ function(self)
for i = 1,self:informar_tamanho() do
self:remover()
end
self.at = false
end ]===],
},
{ id = "informar",
name        = "Informar",
description = [[Informa o valor de determinada posição da lista.
Se nenhuma posição for passada, indica o valor na última posição.]],
parameters = {
{ name = "Posição", type = "INTEGER" },
},
results = {
{ name = "Valor", type = "STRING" },
},
code = [===[ function(self, posicao)
return self._valores[posicao or #self._valores]
end ]===],
},
```

```
{ id = "informar_tamanho",
name        = "Informar Tamanho",
description = [[Informa o tamanho da lista.]],
parameters = {
},
results = {
{ name = "Tamanho", type = "INTEGER" },
},
code = [===[ function(self)
return #self._valores
end ]===],
},
},
code = [==[
function _CLASS:load()
local ret, val = pcall(loadfile,self._filename)
self._valores = ret and val() or {}
end

function _CLASS:save()
local file = io.open(self._filename, "w")
file:write("return {\n")
for i, elem in ipairs(self._valores) do file:write(string.format("%q,\n",elem)) end
file:write("}\n")
file:close()
end

function _CLASS:init()
self._filename = tostring(self)..".lua"
self:load()
end
]==]
}

class{ id = "prem", name = "Premises", group = "Fuzzy",
attributes = {
{ id = "neg", name = "Neg", type = "BOOLEAN",access = "g"},
{ id = "ifpart", name = "ifpart", type = "input_var",access = "g"},
{ id = "ispart", name = "ispart", type = "lingvar",access = "g"},
```

```
},
methods = {
},
}

class{ id = "impl", name = "Implications", group = "Fuzzy",
attributes = {
{ id = "neg", name = "Neg", type = "BOOLEAN",access = "g"},
{ id = "thenpart", name = "thenpart", type = "output_var",access = "g"},
{ id = "ispart", name = "ispart", type = "lingvar",access = "g"},
},
methods = {

},
}

class{ id = "rule", name = "Rule", group = "Fuzzy",
attributes = {
{ id = "connmethod", name = "connmethod", type = "STRING",access = "g"},
{ id = "prem1", name = "prem1", type = "prem",access = "g"},
{ id = "prem2", name = "prem2", type = "prem",access = "g"},
{ id = "impl", name = "impl", type = "impl",access = "g"},
{ id = "step", name = "step", type = "REAL",access = "g"},
},
methods = {
{ id = "atualizarlista",
name        = "Atualizar lista",
description = [[Atualiza lista de x e y]],
parameters = {
},
results = {
},
code = [===[ function(self)

local defuzzmethod = centroid
local implicmethod = tmin
local conn
local prem1 = self.prem1
local prem2 = self.prem2
```

```
local rulevalue2
local rulevalue1

if self.connmethod == "dirmethod" then
conn = tmin
elseif self.connmethod =="andmethod" then
conn = tmin
elseif self.connmethod =="ormethod" then
conn = tmax
end
rulevalue1 = prem1.ispart.mf:calc_inf(prem1.ifpart.value)
if self.connmethod == "dirmethod" then
rulevalue2 = 1
else
rulevalue2 = prem2.ispart.mf:calc_inf(prem2.ifpart.value)
end
local rulevalue = conn(rulevalue1, rulevalue2)

local out = self.impl.thenpart
local lg = self.impl.ispart
local step = (out.mx - out.mn)*self.step

for i=out.mn,out.mx,step do

-- computes the mf value for i
local lgval = lg.mf:calc_inf( i )

-- compute the implication result
local v = implicmethod( lgval , rulevalue )

-- add the result to the fuzzy set
out.fuzzyset.x:inserir( i )
out.fuzzyset.y:inserir( v )



end
out.fuzzyset.x.at = true
out.fuzzyset.y.at = true
end]===],
```

```
},
},
}
```

C.2 Fuzzy PI

```
include('fuzzy.mpa')

class{ id = "fuzzy_pi", name = "Fuzzy_PI", group = "Fuzzy",
attributes = {
{id = "input", name = "Input", type = "REAL_POINT",access = "rw"},
{id = "output", name = "Output" , type = "REAL_POINT",access = "rw"},
{id = "max", name = "Output Max" , type = "REAL"},
{id = "min", name = "Output Min" , type = "REAL"},
{id = "sp", name = "SetPoint" , type = "REAL_POINT",access = "rw"},
{id = "pre", name = "Previous value" , type = "REAL",access = "gs"},
{id = "dt", name = "dt" , type = "REAL",access = "gs"},
{id = "maxin", name = "Input Max" , type = "REAL"},
{id = "minin", name = "Input Min" , type = "REAL"},
{id = "Kp", name = "Kp" , type = "REAL"},
{id = "Ki", name = "Ki" , type = "REAL"},
{id = "localsavein", name = "Save in", type = "fuzzyset",access = "g"},
{id = "localsaveout", name = "Save out", type = "fuzzyset",access = "g"},
{id = "localsavesp", name = "Save sp", type = "fuzzyset",access = "g"},
--{id = "inv", name = "Inverse", type = "BOOLEAN",access = "g"},

},
methods = {
{ id = "calc_error",
name        = "Calculate error",
description = [[Calculate the inference for a given valor.]],
parameters = {
},
results = {
{ name = "error", type = "REAL" },
},
code = [============================================================[
function(self)
local x = self.input:read();
```

```
local sp = self.sp:read();
local ge = self.Ki*0.5/((self.maxin + self.minin)/2);
--local ge = 3*self.Ki/((self.maxin + self.minin)/2-self.Ki*math.abs(sp-x));
local error = (sp-x)*ge;
if error > 1 then
error = 0.99;
elseif error < -1 then
error = -0.99;
end
return error
end
]============================================================],
},
{ id = "calc_rate",
name        = "Calculate rate",
description = [[Calculate the inference for a given valor.]],
parameters = { { name = "error", type = "REAL" },
},
results = {
{ name = "rate", type = "REAL" },
},

code = [============================================================[
function(self,error)
local x = self.input:read();
local sp = self.sp:read();
local dx =  error - self.pre;
local gr = self.Kp*0.5;
--local ge = 3*self.Ki/(((self.maxin + self.minin)/2)-self.Ki*math.abs(sp-x));
--local gr = self.Kp*(3-ge*math.abs(error))/((self.maxin + self.minin)/2);
local rate = dx*gr/self.dt;
if rate > 0.99 then
rate = 0.99;
elseif rate < -0.99 then
rate = -0.99;
end
return rate
end
]============================================================],
```

```
},
{ id = "atualiza_out",
name        = "Atualiza Output",
description = [[Calculate the inference for a given valor.]],
parameters = {{ name = "val", type = "REAL" },
},
results = {
},
code = [============================================================[
function(self,val)
local deriv = (val*((self.maxin + self.minin)/2));
local x = self.input:read();
local sp = self.sp:read();
local out = self.output:read();
out = out+deriv;
if out > self.max then
out = self.max;
elseif out < self.min then
out = self.min;
end
self.output:write(out)
end
]============================================================],
},

{id = "grava_out",
name = "Grava Output",
description = [[Calculate the inference for a given valor.]],
parameters = {{name = "t", type = "REAL"},
},
results = {
},
code = [============================================================[
function(self,t)
local varin = self.localsavein;
local valin = self.input:read();
varin.x:inserir( t );
varin.y:inserir( valin );
local varout = self.localsaveout;
```

```
local valout = self.output:read();
varout.x:inserir( t );
varout.y:inserir ( valout );
local varsp = self.localsavesp;
local valsp = self.sp:read();
varsp.x:inserir( t );
varsp.y:inserir ( valsp );
end
]============================================================],
},
{ id = "changesp",
name        = "Change Sp",
description = [[Calculate the inference for a given valor.]],
parameters = {
{ name = "new", type = "REAL" },
},
results = {
},
code = [============================================================[
function(self,val)
self.sp:write(val);
end
]============================================================],
},
},
}
```

C.3 Sintonia Fuzzy

```
include('fuzzyPI.mpa')

class{ id = "sintonia", name = "Sintonia Saver", group = "Fuzzy",
attributes = {
{id = "fuzzy", name = "Fuzzy", type = "fuzzy_pi",access = "gs"},
{id = "localsavekp", name = "Save Kp", type = "fuzzyset",access = "g"},
{id = "localsaveki", name = "Save Ki", type = "fuzzyset",access = "g"},
{id = "localsaveerro", name = "Save Erro", type = "fuzzyset",access = "g"},
},
methods = {
```

```
{id = "set",
name = "Set",
description = [[set values for Kp and Ki.]],
parameters = { { name = "kp", type = "REAL" },{ name = "ki", type = "REAL" },
},
results = {
},
code = [============================================================[
function(self,kp,ki)
local fuzzy = self.fuzzy;
fuzzy.kp = kp;
fuzzy.ki = ki;
end
]============================================================],
},
{id = "grava",
name = "Grava",
description = [[Calculate the inference for a given valor.]],
parameters = { { name = "erro", type = "REAL" },
},
results = {
},
code = [============================================================[
function(self,erro)
local kpsave = self.localsavekp;
local kisave = self.localsaveki;
local errosave = self.localsaveerro;
local kp = self.fuzzy.kp;
local ki = self.fuzzy.ki;
kpsave.y:inserir( kp );
kisave.y:inserir( ki );
errosave.y:inserir( erro );
end
]============================================================],
},

},
}
class{ id = "sint", name = "Sintonia", group = "Fuzzy",
```

```
attributes = {
{id = "saver", name = "Sintonia Saver", type = "sintonia",access = "gs"},
{id = "kp", name = "Kp", type = "REAL",access = "g"},
{id = "ki", name = "Ki", type = "REAL",access = "g"},
},
methods = {
{id = "set",
name = "Set",
description = [[set values for Kp and Ki.]],
parameters = {
},
results = {
},
code = [============================================================[
function(self)
local fuzzy = self.saver.fuzzy;
fuzzy.kp = self.kp;
fuzzy.ki = self.ki;
end
]============================================================],
},
},
}
class{ id = "fix", name = "Fixer", group = "Fuzzy",
attributes = {
{id = "fuzzy", name = "Fuzzy", type = "fuzzy_pi",access = "gs"},
{id = "minman", name = "valor inicial", type = "REAL",access = "gs"},
{id = "min", name = "valor min", type = "REAL",access = "g"},
{id = "max", name = "valor max", type = "REAL",access = "g"},
},
methods = {
{id = "set",
name = "Set",
description = [[set values]],
parameters = {
},
results = {
{ name = "valido", type = "BOOLEAN" },
},
```

```
code = [============================================================[
function(self)
--self.fuzzy.output:write(minman);
local valid = true;
local con = self.fuzzy.input:read();
if  con < self.min or con > self.max then
valid = false;
end
return valid
end
]============================================================],
},
},
}
```