UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

SISTEMATIZAÇÃO DE PROJETOS DE CONTROLADORES FUZZY ATRAVÉS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

ÁLVARO LUIS DOS REIS LEITÃO

DM 025 / 2009

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2009

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ÁLVARO LUIS DOS REIS LEITÃO

SISTEMATIZAÇÃO DE PROJETOS DE CONTROLADORES FUZZY ATRAVÉS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

DM 025 / 2009

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2009

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ÁLVARO LUIS DOS REIS LEITÃO

SISTEMATIZAÇÃO DE PROJETOS DE CONTROLADORES FUZZY ATRAVÉS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

Dissertação submetida à Banca Examinadora do Programa de Pós-Graduação em Engenharia Elétrica da UFPA para a obtenção do Grau de Mestre em Engenharia Elétrica

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2009

L533s Leitão, Álvaro Luís dos Reis

Sistematização de projetos de controladores fuzzy através de programação orientada a objetos / Álvaro Luís dos Reis Leitão; orientador, Carlos Tavares da Costa Júnior.-2009.

Dissertação (mestrado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-graduação em Engenharia Elétrica, Belém, 2009.

1. Programação orientada a objetos (computação). 2. Sistemas de controle inteligente. 3. Sistemas fuzzy. I. orientador. II. título.

CDD 22. ed. 005.117

UNIVERSIDADE FEDERAL DO PARÁ

INSTITUTO DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

SISTEMATIZAÇÃO DE PROJETOS DE CONTROLADORES FUZZY ATRAVÉS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

AUTOR: ÁLVARO LUIS DOS REIS LEITÃO

DISSERTAÇÃO DE MESTRADO SUBMETIDA À AVALIAÇÃO DA BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA DA UNIVERSIDADE FEDERAL DO PARÁ E JULGADA ADEQUADA PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE SISTEMAS DE ENERGIA.

APROVADA EM 06/10/2009

BANCA EXAMINADORA:

Prof. Dr. Carlos Tavares da Costa Junior
(ORIENTADOR – UFPA)

Prof. Dr. Walter Barra Junior
(MEMBRO – UFPA)

Prof. Dr. Jorge Roberto Brito de Souza
(MEMBRO – UFPA)

Prof. Dr. José Augusto Furtado Real
(MEMBRO – IESAM)

VISTO:

Prof. Dr. Marcus Vinícius Alves Nunes
(COORDENADOR DO PPGEE/ITEC/UFPA)

UFPA / ITEC / PPGEE

Aos meus filhos Daniel, Felipe e Álvaro Filho.

AGRADECIMENTOS

Agradeço aos professores do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Pará, cujo empenho me levou a esta conquista.

Agradeço ao meu amigo Fernando Gomes por estar sempre disposto a auxiliar nas dificuldades, sempre com ótimas sugestões.

Agradeço aos membros da Banca Examinadora pelas críticas de grande valor e muito construtivas.

Agradeço ao Professor Carlos Tavares da Costa Junior, meu orientador, e ao Professor Walter Barra Junior, pelos conhecimentos que me foram passados na área de Sistemas Fuzzy e Controle Fuzzy.

Agradeço ao Professor Jorge Brito de Souza, pelos ensinamentos na área dos Sistemas Lineares e Controle Robusto de Sistemas Multivariáveis.

E também, um agradecimento bastante especial à minha esposa Lauricéia, pela paciência durante toda esta jornada.

A todos vocês, muito obrigado!

Álvaro Luís dos Reis Leitão.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E SÍMBOLOS

AmpOp	Amplificador Operacional
A_{MF}	Matriz A do modelo em espaço de estados em malha fechada
AOO	Análise Orientada a Objetos
C++	"C plus plus" ou C mais mais
CAv	"Centers' Average"
CAverage	Nome da Classe do Defuzificados por Média Ponderada dos Centros
CjFuzzy	Nome da Classe dos Conjuntos Fuzzy
COG	"Center Of Gravity"
COGravity	Nome da Classe do Defuzificador por Centro de Gravidade
Controlador	Nome da Classe do Controlador
EDO	Equação Diferencial Ordinária
G	Ganho
G_1	Ganho do Erro da Posição do Pêndulo
G_2	Ganho da Variação do Erro da Posição do Pêndulo
G_3	Ganho do Erro da Posição da Base
G_4	Ganho da Variação do Erro da Posição da Base
G_b	Ganho da Saída do Controlador da Base
G_e	Ganho da Saída do Controlador de Equilíbrio do Pêndulo
G_{SW}	Ganho da Saída do Controlador de "Swing Up"
icp	início do controle do pêndulo
icb	início do controle da base
IndxEntr	Tabela de Índices das Variáveis de Entrada
IndxSai	Tabela de índices das Variáveis de Saída
isw	início do “swing up”
K_{AK}	Vetor de ganhos calculado pela fórmula de Ackermann
LIFE	"Laboratory for International Fuzzy Engineering"
MInferencia	Nome da Classe da Máquina de Inferência
$\mathcal{M}_C$	Matriz de Controlabilidade
$\mathcal{M}_O$	Matriz de Observabilidade
M_P	Sobre-sinal
PLC	“*Programmable Logic Controller”*

POO	Programação Orientada a Objetos
PRI	Pêndulo Rotacional Invertido
PVI	Problema do Valor Inicial
R_f	"Feedback Resistor"
R_{in}	"Input Resistor"
RK1	Método de Runge-Kutta de Primeira Ordem
RK2	Método de Runge-Kutta de Segunda Ordem
SDinamico	Nome da Classe do Simulador de Sistemas Dinâmicos
T_S	Tempo de Acomodação (s)
TSK	Takagi-Sugeno-Kang
VarLing	Nome da Classe das Variáveis Linguísticas
Vin	"Input Voltage"
Vout	"Output Voltage"
w_n	Freqüência natural (rad/s)
ξ	Coeficiente de Amortecimento

RESUMO

Através do uso da programação em linguagem orientada a objetos e, aplicando-se uma técnica de programação específica, é possível gerar um conjunto de classes genéricas cujos objetos representam cada bloco de um controlador fuzzy e também suas variáveis linguísticas. Tais classes, sendo aplicadas de forma sistemática, facilitam a programação de uma variedade de controladores desta natureza. Este trabalho apresenta a referida técnica e mostra os resultados obtidos através de um modelo simulado de um pêndulo rotacional invertido que é controlado por um sistema de controle composto por três controladores fuzzy, projetados e construídos sob este ponto de vista.

PALAVRAS-CHAVES: Controle, Fuzzy, Programação, Orientado, Objeto, Sistema, Inferência, Simulação.

ABSTRACT

By using object oriented language programming and, applying a specific programming technique, it's possible to generate a set of generic classes whose objects represent each block of a fuzzy controller as well its linguistic variables. Such classes, being applied in a systematic way, make easy the programming of a set of controllers of this nature. This work presents this technique and shows the results obtained via a model simulation of a inverted rotational pendulum which is controlled by a control system composed by three fuzzy controllers designed and built under this point of view.

KEYWORDS: Control, Fuzzy, Programming, Oriented, Object, System, Inference, Simulation.

1 – INTRODUÇÃO

O final da década de 80 e o início da década de 90 foram marcados, do ponto de vista dos Sistemas de Lógica Fuzzy, pelo aparecimento de uma variedade de programas computacionais com a proposta de análise e projeto de Sistemas Fuzzy. A maioria desses programas, ainda disponíveis no mercado e em versões mais atualizadas, apresenta restrições no seu uso por não se tratarem de "softwares" cujo código seja "aberto", e com restrições relativas a esta utilização ou mesmo à sua distribuição.

A título de ilustração, podem ser citados alguns "softwares" que fazem aplicações com o uso de Sistemas de Lógica Fuzzy:

O Fuzzy Logic ToolBox do MatLab® é um dos mais utilizados pela comunidade acadêmica, permitindo programação de aplicações com base em simulação e também com base em interface de "hardware" para controle externo ao micro-computador.

O FuzzyTECH™ é um dos mais evoluídos do ponto de vista de interface com o usuário, apresentando um ambiente de desenvolvimento integrado que permite aplicações industriais; oferece um pacote que incorpora um "Programmable Logic Controller" (PLC), com aplicações bem definidas.

O CubiCalc é uma ferramenta de análise e projeto para sistemas Fuzzy, recomendado para o desenvolvimento do sentimento ou da experiência sobre os métodos que se utilizam dos Sistemas Fuzzy.

O Fuzz-C é um gerador de código em linguagem C, partindo de uma linguagem própria e simplificada, para aplicações em Sistemas Fuzzy. Segundo a sua proposta, o código em linguagem C gerado pode ser compilado por uma variedade de compiladores C existentes no mercado.

O FIDE™ é um "software" que apresenta um ambiente de desenvolvimento integrado para o projeto de aplicações em Sistemas Fuzzy, possuindo um conjunto de geradores de códigos capazes de gerar código de algoritmos Fuzzy nas linguagens Java, ANSI C, MatLab® M-file e código assembly para uma variedade de microcontroladores (aplicações em sistemas embarcados); apresenta ainda uma característica que o difere dos demais "softwares" citados anteriormente: A sua base de regras é introduzida no formato matricial, formato adotado neste trabalho.

O elevado custo de alguns desses "softwares" é outro fator que reduz o interesse no seu uso, uma vez que, dependendo da aplicação, a relação custo-benefício pode ser elevada. Para o caso da comunidade científico-acadêmica, sistemas cujos códigos sejam livres e abertos facilitam a sua distribuição, auxiliando na difusão da informação e do conhecimento, além de permitir que esses programas sejam mais bem adaptados às suas necessidades.

1.1 – MOTIVAÇÃO

A ideia de um trabalho aberto e, consequentemente, com pleno acesso a todas as suas informações, técnicas e métodos, tornando-o assim útil à comunidade acadêmica no sentido de contribuir com o seu aprendizado, além de permitir, através do acesso ao seu código seu próprio aprimoramento ou ainda sua utilização, sem restrição alguma, no desenvolvimento de aplicações em Sistemas de Controle baseados em Lógica Fuzzy, constituem a principal motivação para o desenvolvimento deste trabalho.

1.2 – OBJETIVO

O presente trabalho tem por objetivo principal subsidiar o estudo e desenvolver um conjunto de classes, compondo-se uma biblioteca estática, via programação em linguagem orientada a objetos, que sirvam para sistematizar a programação de controladores fuzzy em microcomputadores do tipo PC. Esta sistematização deve facilitar o trabalho de programação destes controladores visando reuso de código com sua consequente redução, podendo refletir-se até no aumento da facilidade de sua manutenção como será visto ao longo deste desenvolvimento.

Para a sua validação, também é desenvolvido um simulador de sistemas dinâmicos contínuos sob o paradigma da orientação a objetos, tornando possível a análise por intermédio de interface gráfica, do comportamento destes sistemas através dos seus sinais. Tal simulador é de suma importância para este trabalho na implementação e análise do estudo de caso apresentado.

1.3 – METODOLOGIA DE TRABALHO

Este trabalho apresenta a seguinte linha de desenvolvimento:

- Especificação das classes via AOO, cujos objetos representem os blocos básicos de um controlador fuzzy (sistema do tipo Mamdani), com o objetivo de desenvolver sua programação em linguagem orientada a objetos.

- Programação, em MatLab®/Simulink®, das classes especificadas e do modelo do pêndulo rotacional invertido (estudo de caso) para a simulação e validação dos controladores que são desenvolvidos, sob este paradigma, para este modelo.

- Programação, em C++, das classes especificadas; programação do simulador de sistemas dinâmicos contínuos e inscrição do modelo do pêndulo rotacional invertido com o objetivo de implementar a simulação conforme especificado para este trabalho.

- Análise e comparação dos resultados obtidos através do MatLab®/Simulink® e dos resultados obtidos via Borland C++ Builder®.

1.4 – ORGANIZAÇÃO

Para que os objetivos propostos no presente trabalho sejam alcançados, essa dissertação está estruturada em seis capítulos, sendo este primeiro introdutório e mais cinco específicos conforme segue a descrição.

No capítulo dois é feita uma abordagem teórica sobre os conceitos da programação orientada a objetos, sendo apresentado um exemplo, escrito em linguagem C++, que é mais relacionado com a área de Engenharia Elétrica e fugindo dos exemplos que são usualmente apresentados na literatura especializada no ensino da programação computacional. Seu objetivo é estabelecer um conhecimento básico da programação orientada a objetos, que é de fundamental importância ao entendimento das técnicas empregadas para o desenvolvimento deste trabalho. Uma abordagem, explicando a definição dos sistemas fuzzy, também é feita neste capítulo, onde todos os conceitos relacionados a este tópico são apresentados com o objetivo de se construir o embasamento teórico que também é de grande relevância para este trabalho.

No capítulo três é mostrada uma forma, bastante prática, do emprego da programação orientada a objetos, em C++, na implementação de controladores fuzzy, cujos programas rodam em microcomputadores do tipo PC. O emprego especializado deste tipo de programação para a criação das classes, as quais servem para construir os objetos que representam os blocos distintos que, reunidos, compõem estes tipos de controladores.

No capítulo quatro, o mesmo conceito de orientação a objetos é usado para desenvolver uma classe especializada para criar objetos que sejam simuladores de sistemas dinâmicos contínuos lineares ou não lineares. Tal classe permite, em tempo de programação, a inscrição do modelo matemático em espaço de estados, da planta que se pretende simular. Um objeto desta classe, que possua um sistema dinâmico contínuo inscrito, pode simulá-lo, através de um algoritmo de integração numérica, que é programado como um dos seus métodos, desde um tempo inicial t_0, por n passos h, até um tempo final t_f, fornecendo o

estado do sistema a cada um desses passos. Dois parâmetros importantes também podem ser adicionados ao modelo em tempo de programação: entrada de controle, para um sistema que não seja autônomo e variável de tempo, caso o sistema seja variante no tempo.

No capítulo cinco é apresentada, através da simulação descrita no capítulo quatro, uma aplicação completa da técnica descrita no capítulo três. Tal aplicação trata de um sistema de controle de um pêndulo rotacional invertido, que é desenvolvida sob a técnica apresentada neste trabalho e consiste de um projeto que reúne três controladores fuzzy distintos, um controlador de balanço ou "swing up", que é responsável por lançar o pêndulo para bem próximo do ponto de equilíbrio instável superior, um controlador de equilíbrio, que é responsável por estabilizar o pêndulo nesta posição e um controlador da posição da base, que é responsável por retornar a base rotativa do pêndulo para a origem. É importante observar que esta aplicação controla duas variáveis de estado a partir de uma única entrada de controle. São ainda apresentados os gráficos referentes à simulação deste sistema de controle em ação.

No capítulo seis são levantadas as discussões e as conclusões alcançadas. São apresentadas também sugestões de futuros trabalhos que podem ser desenvolvidos, ou a partir do próprio pêndulo, ou da técnica aqui estudada.

2 – CONCEITOS UTILIZADOS

2.1- ORIENTAÇÃO A OBJETOS

2.1.1 – Introdução

O termo "Orientação a Objetos" foi criado por Alan Key, autor da linguagem SmallTalk, no final dos anos 60. Sua idéia foi baseada no conceito introduzido pela linguagem Simula67, criada em 1967, por Ole Johan Dahl e Kristen Nygaard, a qual é considerada, na história da computação, como sendo a primeira linguagem a utilizar o conceito de Orientação a Objetos.

Foi durante os anos 90 que a Programação Orientada a Objetos(POO) teve seu maior impulso através das utilizações em software, com o aparecimento da linguagem Java, criada em 1995 a partir da linguagem Oak, desenvolvida através de um projeto, denominado "Star Seven" (*7), de James Gosling, Mike Sheridan e Patrick Naughton.

Atualmente o paradigma da Programação Orientada a Objetos é considerado um conceito moderno do ponto de vista da engenharia de software, influindo na análise, no projeto e na programação.

Ao contrário da programação estruturada, que é implementada através de seqüências de passos e tarefas denominadas de código, o qual atua sobre os dados, na programação orientada a objetos, os dados controlam o acesso ao código, sendo o programa organizado por meio de um conjunto de interfaces bem definidas e em torno destes dados, os quais são chamados de objetos.

2.1.2 – Classes e Objetos

Para melhor compreensão do paradigma da programação orientada a objetos, é necessária uma definição clara do conceito do que vem a ser "objeto", bem como venha a ser feita a sua construção.

Objetos são variáveis criadas a partir de tipos complexos de dados denominados de classes.

As classes são declarações estáticas (tipos) definidas pelo usuário que, quando implementadas, servem como uma espécie de molde para estes objetos [2]. Sua definição vai além do que é visto nos tipos de estruturas que são usualmente empregados na programação estruturada. As classes definem, para os seus objetos, os atributos, que são os dados pertencentes a eles e os métodos, que são códigos de programa dentro destes objetos, e servem para fazer a "comunicação" do meio exterior com o objeto propriamente dito. Os métodos tem fundamental importância na implementação das funcionalidades de cada objeto. Dentre os diversos métodos definidos por uma classe, há um que é obrigatório em todas as classes: o método criador, que é responsável pela alocação do espaço, na memória, o qual será ocupado pelos atributos e pelos métodos definidos na classe. O método criador também inicializa os atributos com valores, que poderão ser modificados de forma direta ou através dos métodos do próprio objeto, ao longo da sua "existência" (período compreendido desde a criação de um objeto até a sua destruição). Esta ação de alocar espaço em memória e dar valores aos atributos é denominada de "instanciação de uma classe", ou seja, instanciar uma classe é o mesmo que criar um objeto pertencente à referida classe. Portanto um objeto é a instância de uma classe. Outro método, também importante é o destrutor, que tem a função de desalocar o espaço de memória criado pelo método construtor, apagando o referido objeto.

Para um melhor entendimento destas definições, é utilizado como exemplo a seguinte situação: Necessita-se desenvolver um programa para simular um sistema que será montado com base em componentes eletrônicos, onde se utilizam diversos amplificadores operacionais para implementar as sucessivas etapas de amplificação do sistema e, em alguns casos, utilizam-se etapas com amplificação inversora Fig.1 e, em outros, não inversora Fig.2. Cada etapa também possui um ganho diferente e, portanto, cada ganho deve ser ajustado independentemente.

Da forma como o problema é colocado, tornam-se necessários "objetos" da classe "Amplificador", com parâmetros Rf (resistor de realimentação), Rin (resistor de entrada) e G (ganho), e que tenham métodos (funcionalidades) de "inversor" e "não-inversor", além dos ajustes dos parâmetros Rf e Rin , uma vez que, para o amplificador da Fig.1, tem-se como ganho $G = -\frac{R_f}{R_{in}}$ e, para o amplificador da Fig.2, o ganho é $G = +\frac{R_f}{R_{in}}$.

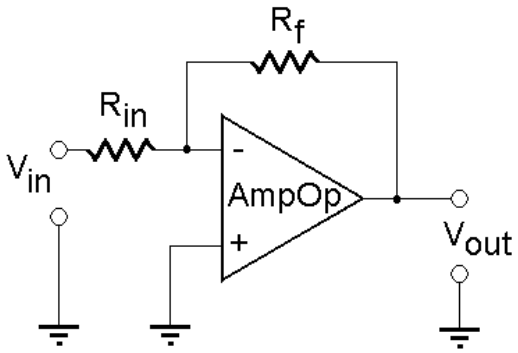


Figura 1 – Amplificador inversor.

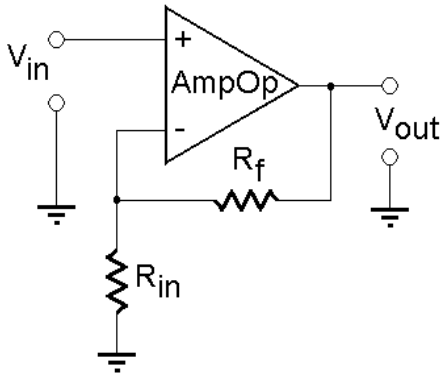


Figura 2 – Amplificador não-inversor

É mostrado, na Fig.3, um diagrama que representa a classe Amplificador. Este diagrama é dividido em três partes onde, a parte de cima contém o nome da classe, a parte intermediária é o espaço onde ficam dispostas as propriedades, ou atributos da classe e a parte inferior os métodos que implementam as funcionalidades e as interfaces da classe. Este modelo de representação de classe é o que será utilizado ao longo de todo este trabalho.

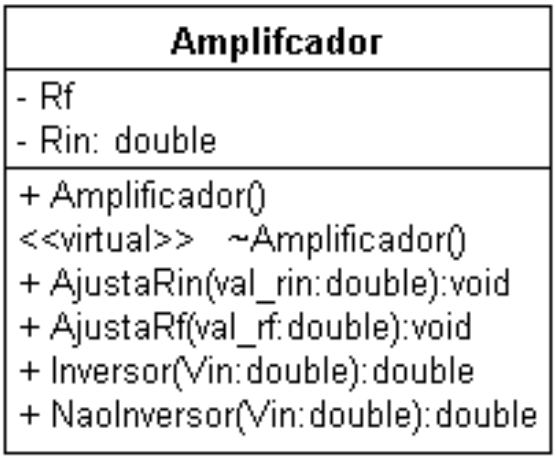


Figura 3 – Classe Amplificador

O objetivo de se descrever uma classe conforme a Fig.3 é facilitar o seu entendimento, separando a informação conforme o seu tipo (atributo ou método) e facilitar a organização da programação, conforme é mostrado no seguinte trecho, escrito em C++, o qual define a classe Amplificador :

```
class Amplificador{
        private:
                double          Rf, Rin;
        public:
                Amplificador ( ){Rin=1;Rf=0;};
                void    Ajusta_Rf ( double val_rf ) { Rf = val_rf;};
                void    Ajusta_Rin ( double val_rin ) { Rin = val_rin;};
                double Inversor ( double Vin ) { return (-Vin*Rf/Rin);};
                double NaoInversor ( double Vin ){ return (Vin*(1+Rf/Rin));};
                ~Amplificador ( );
        };
```

É importante observar que, como métodos, a classe Amplificador tem um criador e um destrutor (Amplificador ( ) e ~Amplificador( ) respectivamente) que, pela concepção das linguagens C++, estes dois métodos são obrigatórios; o primeiro para a inicialização dos objetos da classe, o qual não retorna parâmetros, e o segundo, para o cancelamento dos objetos da classe quando não são mais necessários.

Os métodos Ajusta_Rf (.) e Ajusta_Rin (.) são para alterar os valores dos resistores Rf e Rin respectivamente e, os métodos Inversor (.) e NaoInversor (.) dão aos objetos da classe Amplificador as funcionalidades de inversor e não-inversor, retornando os respectivos valores multiplicado-se os valores de entrada pelos ganhos correspondentes.

Uma vez definida a classe Amplificador, pode-se então, em tempo de programação, criar os seus objetos que serão os amplificadores do sistema em questão. Esta criação de objetos amplificadores pode ser feita na forma direta, conforme mostrado no quadro abaixo:

```
Amplificador Amp01( ); // Criação do objeto Amp01.

Amplificador Amp02( ); // Criação do objeto Amp02.
```

Cada um destes comandos invoca o criador da classe Amplificador que aloca espaço em memória para conter os dados de cada um destes objetos além de atribuir às variáveis Amp01 e Amp02 as referências aos endereços de memória onde foram alocados estes espaços. Observa-se ainda que, o uso do criador desta classe, inicializa Rf com 0 e Rin com 1, significando ganho inicial para cada amplificador igual a 0. Tais ganhos podem ser ajustados com o uso dos métodos Ajusta_Rf(.) e Ajusta_Rin(.).

Para exemplificar a utilização destes objetos, considera-se o seguinte código:

```
double Vout;  // declara a variável Vout.

Amp01.Ajusta_Rf( 10000 );  // Rf = 10000 para Amp01.

Amp01.Ajusta_Rin ( 1000 ); // Rin=1000 para Amp01.

Vout = Amp01.Inversor ( 1 ); // Uso de Amp01 no modo inversor. Vout= -10.

Vout = Amp01.NaoInversor ( 1 ); // Modo não-inversor. Vout = 11.
```

2.1.3 – Principais Características da Programação Orientada a Objetos

●Abstração: É o processo de identificação dos objetos e de seus relacionamentos, onde a definição de cada objeto faz com que o programador possa concentrar seu foco apenas naquilo que o objeto faz sem se importar na maneira como é feita pelo objeto.

●Encapsulamento: É a faculdade que um objeto tem de "esconder" as informações ou características que são de menor relevância ao usuário. Por exemplo, para o caso do objeto Amp01, não se consegue atribuir valores a Rf e a Rin de forma direta, ou seja, não funcionaria o comando "Amp01.Rf=10000;". Esta atribuição só pode ser feita através do método Ajusta_Rf(.).

●Herança: É o mecanismo que permite com que uma determinada classe, denominada de classe filha, herde características de outra classe, denominada de classe pai. Outra denominação comum para classe pai é o termo “superclasse” (mais genérica), que é usado para a classe base da qual as demais classes filhas, também ditas “derivadas” ou “subclasses” (mais especializadas) herdam características.

Quando esta herança ocorre a partir de uma única classe, ela é dita herança simples e, quando ocorre a partir de mais de uma classe, é dita herança múltipla.

A herança ocorre, em geral, quando são definidas novas classes, com atributos que são objetos de outras classes. Nem todas as linguagens de programação orientadas a objetos permitem herança múltipla. A linguagem C++, por exemplo, permite, porém, Java não permite.

●Polimorfismo: Implementado por intermédio de herança e da reescrita dos métodos (“overriding”) das superclasses nas subclasses, que conferem aos objetos das subclasses uma forma de tratar seus dados de maneira mais específica que os objetos das superclasses.

2.1.4 – Conclusão

Nesta seção procurou-se definir o conceito de orientação a objetos, mostrando de forma sucinta suas principais características e apresentando um breve exemplo, de cunho bastante prático. O modelo de representação de classe também foi apresentado.

É importante ressaltar que, uma diferença entre a técnica de programação estruturada e a técnica de programação orientada a objetos deve ser considerada, pois na estruturada, o foco da programação ocorre em torno da solução, já com a orientação a objetos este foco ocorre em torno do problema, o qual é separado em partes e cada uma destas partes é convertida em um objeto (abstração).

2.2 – SISTEMAS FUZZY

2.2.1 – Introducao

A lógica fuzzy foi proposta em 1965 por Lotfi A. Zadeh, da Universidade de Berkeley, Califórnia, EUA, que definiu o conceito de conjuntos fuzzy, definindo também, para estes conjuntos, noções de inclusão, união, interseção, relação, convexidade, bem como várias propriedades associadas a estas [1].

Em 1973, Zadeh introduziu o conceito de variável linguística, caracterizando ainda, as relações simples entre estas variáveis através de declarações condicionais fuzzy e as relações complexas através de algorítimos fuzzy [2].

Seguindo a linha de raciocínio de Zadeh, várias pesquisas sucederam-se e, com elas, o surgimento da primeira aplicação industrial com uso de controle com lógica fuzzy: um forno para a fabricação de cimento, que foi construído na Dinamarca em 1975.

Apesar deste fato, durante os anos seguintes, os sistemas de lógica fuzzy foram grandemente ignorados nos Estados Unidos, por serem considerados do campo da inteligência artificial, que era um campo no qual, ocasionalmente, se apresentava uma idéia de resultados que ia além do que cumpria, principalmente na metade da década de 80, recebendo, nesta época, pouca credibilidade do ponto de vista comercial.

Ao contrário dos americanos, os japoneses investiram nos sistemas fuzzy e, em 1985, Seiji Yasunobu e Soyoji Miyamoto, da Hitachi, através de simulações, demonstraram a eficiência dos sistemas de controle fuzzy na ferrovia de Sendai, onde os sistemas foram implantados, na época da inauguração da linha em 1987, para aceleração, frenagem e parada.

Outro feito, diretamente relacionado aos sistemas de controle fuzzy, foi o de Takeshi Yamakawa que, em 1987, durante o Encontro Internacional de Pesquisadores fuzzy, em Tókyo, apresentou uma demonstração do uso do controle fuzzy em um experimento com um pêndulo invertido (sistema carro-poste), que era controlado por um sistema de lógica fuzzy composto por chips dedicados. Como resultado, os espectadores ficaram impressionados, mesmo porque Yamakawa aumentou o grau de perturbação no topo do pêndulo usando de uma taça com água e, até mesmo um camundongo vivo, conseguindo manter o pêndulo estável. A partir destas demonstrações, engenheiros japoneses passaram a desenvolver uma variedade de sistemas fuzzy com aplicações industriais e em produtos de consumo. Já em

1988 o Japão implantou o “Laboratory for International Fuzzy Engineering (LIFE)”, uma cooperativa que reunia em torno de 48 empresas que desenvolviam pesquisas na área dos sistemas fuzzy. Com isso, é possível citar alguns bens de consumo que hoje fazem uso de sistemas fuzzy para o seu funcionamento: Aspiradores de pó (Matsushita) – ajuste de sucção, condicionadores de ar (Mitsubishi) – sensores de temperatura e controle térmico, câmeras fotográficas (Canon) - controle de autofoco, máquinas de lavar (Hitachi) – peso de carga e mistura de sabão.

Atualmente, as pesquisas relacionadas aos sistemas fuzzy concentram interesses em duas linhas. A primeira tenta explicar o porquê dos resultados práticos serem tão bons e, a segunda visa sistematizar as abordagens existentes e o desenvolvimento de novos tipos de abordagens.

Neste capítulo é definido o conceito de sistema fuzzy juntamente com a teoria relacionada.

2.2.2 – Definição de Sistemas Fuzzy

Para que se possa definir um conceito acerca dos sistemas fuzzy, inicialmente deve ser esclarecido o que, de fato, venha a significar o termo “fuzzy”.

Fuzzy é uma palavra originada na língua inglesa e que é usada para fazer referência ao que é vago ou definido imprecisamente ou nebuloso. Sem considerar o significado denotativo do termo fuzzy, os sistemas fuzzy são precisamente definidos, bem como o controle fuzzy, que é um tipo especial de controle não-linear. Procura-se enfatizar que, embora os sistemas fuzzy se caracterizem por um fenômeno de definição imprecisa, a sua teoria é precisa, a qual é justificada por:

(1) Vaguesa (Definição aproximada), que é introduzida para que se obtenha um modelo de raciocínio, observando-se que esta não é a única forma de se aproximar o conhecimento.

(2) Maneira sistemática de formular, através de uma teoria, o conhecimento humano, colocando-o em sistemas de engenharia, juntamente com fórmulas matemáticas e

medidas de sensores. Esta é a principal essência da teoria dos sistemas fuzzy, justificando-a como um campo independente da engenharia.

Os sistemas fuzzy se baseiam no conhecimento ou em regras, onde o núcleo de um sistema fuzzy é a base do conhecimento que consiste das regras fuzzy SE-ENTÃO. Estas regras são declarações nas quais constam palavras caracterizadas por funções de pertinência contínuas. Considerando como exemplo:

SE a_velocidade_do_carro é **alta**, ENTÃO aplicar **menos** força ao_acelerador (2.1)

Os termos **alta** e **menos**, na declaração acima, se caracterizam por funções de pertinência, conforme mostrado na Fig.4.

Para que se possa construir um sistema fuzzy, deve-se partir de uma coleção de regras fuzzy SE-ENTÃO. A obtenção da coleção de regras fuzzy é feita a partir do conhecimento que é desenvolvido pelos especialistas humanos, o qual é organizado na forma regras fuzzy que são combinadas como um único sistema. Se um sistema fuzzy é usado como controlador, denomina-se este de controlador fuzzy.

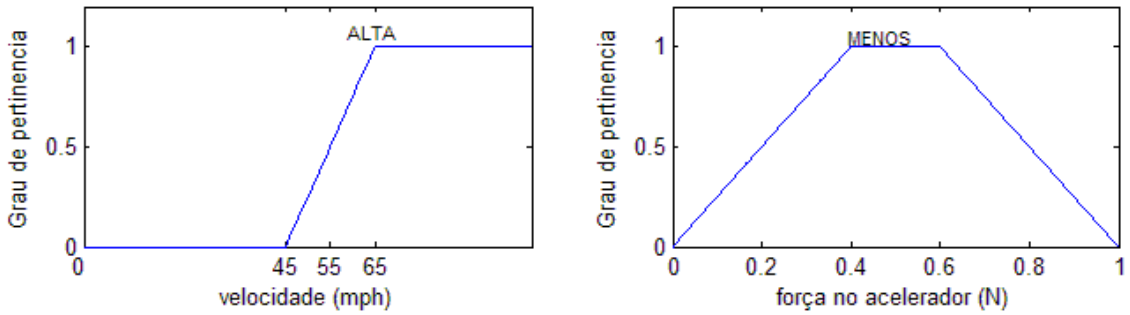


Figura 4 – Funções de pertinência para os termos alta e menos.

Dependendo de como o sistema fuzzy é organizado, ele pode ser de três tipos:

(1) Sistema fuzzy puro: Neste sistema as entradas e as saídas são conjuntos fuzzy, ou seja, palavras em língua natural. Tais sistemas não se aplicam de forma direta em sistemas de engenharia, o que é apontado como problema pela literatura [6], pois nos sistemas de engenharias as entradas e as saídas são variáveis com valores reais. Sua máquina de inferência

combina as regras fuzzy SE-ENTÃO em um mapeamento dos conjuntos fuzzy no espaço de entrada $U \subset \mathbb{R}^n$ para os conjuntos fuzzy no espaço de saída $V \subset \mathbb{R}^n$ com base nos princípios da lógica fuzzy. No caso da existência da linha pontilhada, o sistema é chamado de sistema fuzzy dinâmico. A Fig. 5 mostra o diagrama deste tipo de sistema.

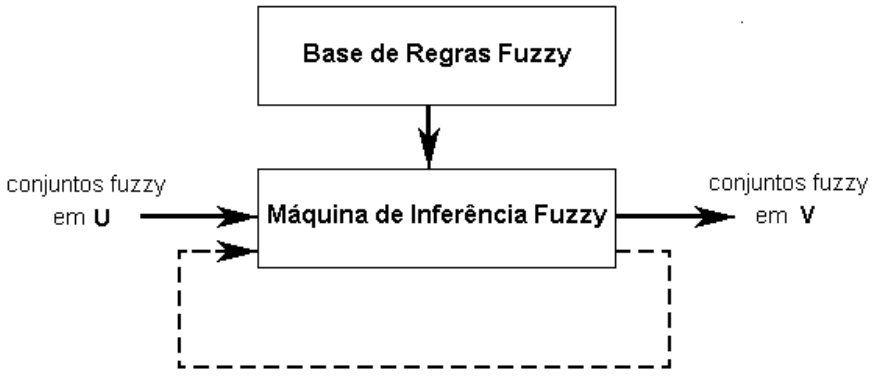


Figura 5 – Sistema fuzzy puro.

(2) Sistema fuzzy Takagi-Sugeno-Kang (TSK): No sistema Takagi-Sugeno-Kang, ao invés de se usar regras fuzzy SE-ENTÃO na forma apresentada na Eq.(2.1), usa-se regras fuzzy SE-ENTÃO conforme:

SE a_velocidade_do_carro é **alta**, ENTÃO a_força_para_o_acelerador é y = Cx (2.2)

onde a palavra **alta** tem o mesmo significado anterior e C é uma constante.

A parte ENTÃO da regra deixa de ser uma descrição que usa palavras em linguagem natural para ser uma simples fórmula matemática.

Esta mudança, em relação ao sistema fuzzy puro, facilita a combinação de regras, apresentando-se, o sistema fuzzy TSK, como uma média ponderada dos valores nas partes SE-ENTÃO das regras. A Fig. 6 mostra o diagrama para o sistema fuzzy TSK.

Por apresentar a parte ENTÃO das regras como fórmula matemática, este sistema não oferece uma estrutura natural para representar o conhecimento humano. Não há também muita liberdade para se aplicar princípios diferentes de lógica fuzzy, portanto as versatilidades dos sistemas fuzzy não são bem representadas nesta estrutura.

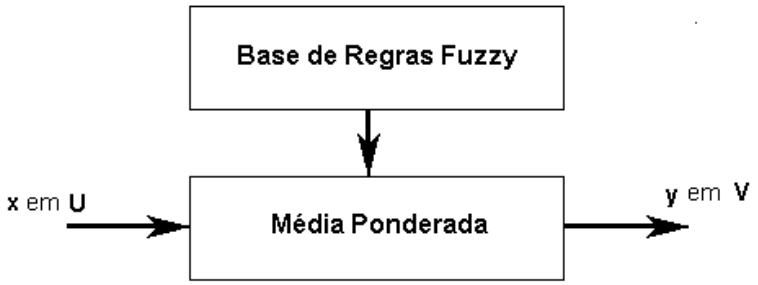


Figura 6 – Sistema fuzzy Takagi-Sugeno-Kang

Quando for o caso, para se contornar as dificuldades apresentadas pelo sistema fuzzy puro e pelo sistema TSK, é recomendada a aplicação da terceira opção de sistema, onde há a implementação do fuzificador e o defuzificador.

(3) Sistema fuzzy com fuzificador e defuzificador: Os sistemas de engenharia podem fazer uso dos sistemas fuzzy puros desde se adicione um fuzificador, que transforma o valor real da entrada em um conjunto fuzzy e um defuzificador, que transforma o conjunto fuzzy em um valor real de saída. A Fig. 7 apresenta o diagrama deste sistema.

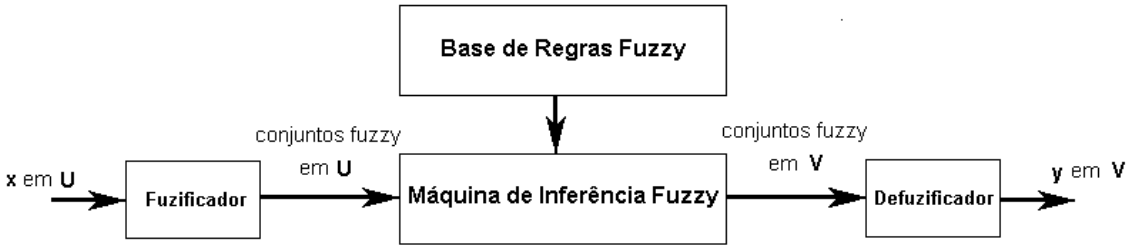


Figura 7- Sistema fuzzy com fuzificador e defuzificador.

O presente trabalho dá ênfase aos sistemas que empregam o sistema com fuzificador e defuzificador.

2.2.3– Conjuntos Fuzzy

Considere U o universo de discurso, ou o conjunto universo, onde estão contidos todos os elementos que podem ser relacionados com cada aplicação particular.

Pela teoria clássica de conjuntos é possivel se definir um conjunto A de duas formas distintas: Listando-se os seus elementos (forma conhecida como método da lista), ou através da especificação das propriedades comuns aos seus membros (forma conhecida como método da regra), conforme:

$$A = \{x \in U \mid x \ satisfaz\ algumas\ condições\} \tag{2.3}$$

Pode-se aplicar ainda, um terceiro método para a definição de um conjunto A. Denominado de método da pertinência, onde se introduz uma função de pertinência, com valor que vai de zero a um, também chamada de função característica, função de discriminação ou função indicador para A, que é denotada por $\mu_A\ (x)$, como segue:

$$\mu_A(x) = \begin{cases} 1\ se\ x \in A \\ 0\ se\ x \notin A \end{cases} \tag{2.4}$$

Para uma melhor percepção, com relação à definição de conjuntos sob certas condições, é dado o seguinte exemplo:

Definindo-se o conjunto universo U como "todos os carros da cidade de Berkeley, Califórnia, EUA", também é possível definir outros conjuntos de carros em U, tais como:

(a) Número de cilindros: Conjunto A em U que contenha todos os carros com motor de quatro cilindros. Segundo o método da regra A pode ser definido como:

$$A = \{x \in U \mid x \ \text{tem quatro cilindros}\ \} \tag{2.5}$$

E através do método da pertinência A é definido como:

$$\mu_A(x) = \begin{cases} 1 & se\ (x \in U)\ e\ x\ tem\ quatro\ cilindros \\ 0 & se\ (x \notin U)\ e\ x\ não\ tem\ quatro\ cilindros \end{cases} \tag{2.6}$$

(b) Carros americanos e carros não americanos: Esta definição apresenta uma complexidade maior, uma vez que muitos componentes que integram os carros americanos são fabricados fora dos EUA, acrescentando-se ainda que, muitos carros não americanos são produzidos nos EUA.

A dificuldade apresentada pelo problema do exemplo citado é devido a não existência de uma fronteira clara para se separar carros americanos e carros não americanos e, na teoria clássica de conjuntos a exigência de propriedades bem definidas é fundamental para a definição desses conjuntos.

Para superar esta limitação, imposta pela teoria clássica, foi introduzido o conceito de conjunto fuzzy, o qual põe de lado a idéia de que esta limitação seja fundamental, passando a ser necessária uma nova teoria – a teoria dos conjuntos fuzzy [1].

Define-se um conjunto fuzzy em um universo de discurso U caracterizando-o por uma função de pertinência $\mu_A\ (x)$ que possui valores no intervalo [0,1], sendo, este conjunto fuzzy, uma generalização de um conjunto clássico onde é permitido, à função de pertinência, assumir qualquer valor no intervalo [0,1].

Por conta desta generalização, é possível verificar que, para um conjunto clássico, a função de pertinência só pode assumir dois valores – zero ou um, enquanto que, para um conjunto fuzzy a função de pertinência é contínua cujo intervalo de valores possíveis é [0,1].

A representação de um conjunto fuzzy A em U é feita como um conjunto de pares ordenados de um elemento genérico x e seu valor de pertinência, conforme segue:

$$A = \{(x, \mu_A(x)) | x \in U\} \tag{2.7}$$

Para o caso em que U seja contínuo ($U = \mathbb{R}$), A pode ser representado como:

$$A = \int_U \mu_A(x)/x \tag{2.8}$$

E para quando U for discreto, a representação pode ser feita da seguinte maneira:

$$A = \sum_U \mu_A(x)/x \tag{2.9}$$

Onde o sinal de integral na Eq.(2.8) não representa a operação de integração e sim a reunião de todos os pontos $x \in U$ com a função de pertinência $\mu_A(x)$ associada e, a somatória

na Eq.(2.9), não significa a soma aritmética e sim a reunião de todos os pontos $x \in U$ com a respectiva função de pertinência $\mu_A(x)$ associada.

Com o conceito de conjunto fuzzy definido, agora é possível se voltar à questão de todos os carros de Berkeley e representá-la através de dois conjuntos fuzzy: carros americanos (μ_D) e carros não americanos (μ_F), graficamente representados na Fig.8.

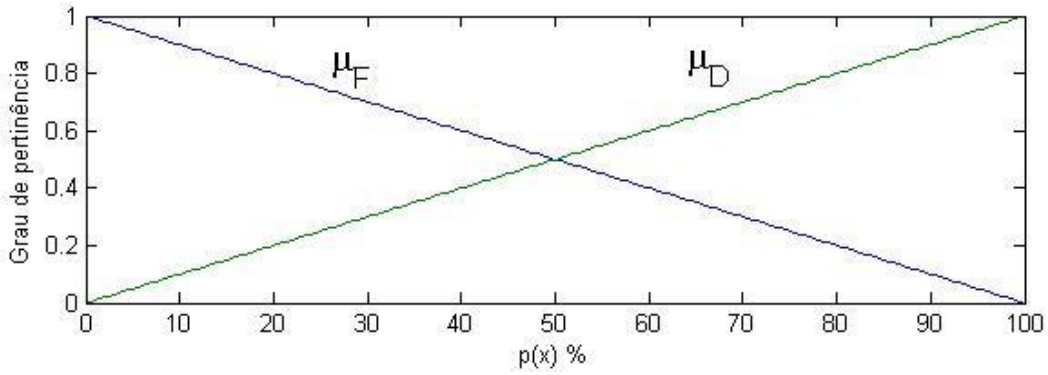


Figura 8 – Todos os carros de Berkeley.

Na Fig.8, as funções de pertinência são funções do percentual de peças $p(x)$ americanas existentes nos carros, conforme:

$$\mu_D = p(x) \tag{2.10}$$

$$\mu_F = 1 - p(x) \tag{2.11}$$

Observa-se que um elemento pode fazer parte de mais de um conjunto com os mesmos ou diferentes graus de pertinência.

Vários conceitos básicos relacionados aos conjuntos fuzzy também foram desenvolvidos, alguns são extensões dos conceitos dos conjuntos clássicos, também referidos na teoria dos conjuntos fuzzy pelo termo “crisp”, cujo significado é o contrário do termo “fuzzy”, significando “bem definido”. Para os referidos conceitos, são definidos:

● Suporte: para um conjunto fuzzy A em U, é o conjunto crisp que contém todos os elementos de U cujos valores de pertinência sejam diferentes de zero em A, com seguinte representação:

$$supp(A) = \{x \in U \,|\mu_A(x) > 0\} \tag{2.12}$$

●Singleton fuzzy: é um conjunto fuzzy cujo suporte é um único ponto em U.

●Centro de um conjunto fuzzy: Após calcular-se o valor médio de todos os pontos onde a função de pertinência de um dado conjunto fuzzy alcança o seu valor máximo e, se este valor for finito, então este valor médio é definido como o centro deste conjunto fuzzy. Se o valor médio for $+\infty$, então o centro do conjunto será o menor de todos os pontos que alcançam o valor máximo de pertinência e, caso este valor seja $-\infty$, seu centro será o maior destes pontos.

●Ponto de cruzamento: é o ponto em U onde o valor da função de pertinência em A é igual a 0.5.

●Altura de um conjunto fuzzy: é o maior valor da função de pertinência atingido por qualquer ponto. Os conjuntos que têm altura igual a 1, são denominados de "conjuntos fuzzy normalizados".

●Corte-α de um conjunto fuzzy: Dado um valor de pertinência α, o corte-α é o conjunto crisp A_α de todos os elementos em U, dos quais os valores de pertinência são maiores ou iguais ao valor de α.

A Fig.9 mostra em gráfico estes conceitos básicos relacionados aos conjuntos fuzzy.

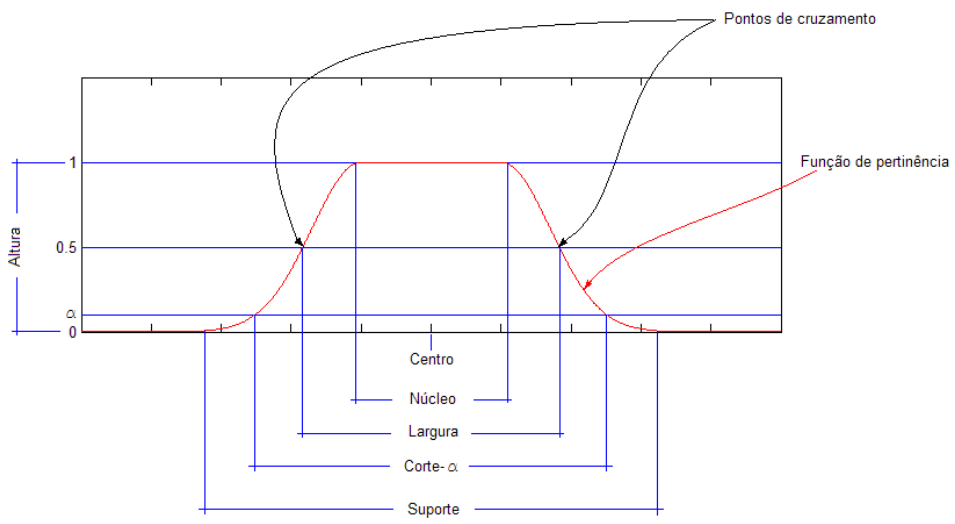


Figura 9 - Conceitos básicos relacionados aos conjuntos fuzzy

●Convexidade: Sendo U um espaço Euclidiano n-dimensional $\mathbb{R}^n$, o conceito de convexidade de conjunto pode ser generalizado ao conjunto fuzzy.

Um conjunto fuzzy é convexo se e somente se $A_\propto$ é um conjunto convexo no intervalo (0,1]. A definição para conjunto fuzzy A em $\mathbb{R}^n$ convexo é apresentada conforme:

$$\mu_A[\lambda x_1 + (1-\lambda)x_2] \geq min\,[\mu_A(x_1), \mu_A(x_2)] \tag{2.13}$$

para todo $x_1, x_2 \in \mathbb{R}^n$ e todo $\lambda \in [0,1]$.

Os conceitos vistos até o presente momento dizem respeito a apenas um conjunto fuzzy, em seguida, são apresentadas operações relacionadas com mais de um conjunto fuzzy.

2.2.4– Operações Com Conjuntos Fuzzy

Sejam A e B dois conjuntos fuzzy definidos no mesmo universo de discurso U.

São definidas as operações de igualdade, inclusão, complemento, união e interseção entre dois conjuntos fuzzy A e B de acordo com o que segue:

● Igualdade: Dois conjuntos fuzzy são iguais se e somente se os seus valores de pertinência forem iguais para todo elemento correspondente pertencente ao universo de discurso U, conforme definido por:

$$A = B \Leftrightarrow \mu_A(x) = \mu_B(x)\,, \forall\, x \in U \tag{2.14}$$

● Inclusão: Um conjunto fuzzy está contido em outro se e somente se os valores de pertinência de um conjunto são sempre menores ou iguais aos valores de pertinência do outro para todo elemento correspondente pertencente ao universo de discurso U, conforme a seguinte definição:

$$A \subset B \Leftrightarrow \mu_A(x) \leq \mu_B(x)\,, \forall\, x \in U \tag{2.15}$$

● Complemento: O complemento de um conjunto fuzzy A é um conjunto fuzzy $\bar{A}$ em U. Sua função de pertinência é definida por:

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \tag{2.16}$$

● União: O conjunto fuzzy $A \cup B$ em U representa a união entre os conjuntos fuzzy A e B. É importante observar que o uso do “max" indica a obtenção do maior dos valores de pertinência indicados para cada elemento do domínio, mostrado como segue:

$$A \cup B \Rightarrow \mu_{A \cup B}(x) = max\ [\mu_A(x), \mu_B(x)] \tag{2.17}$$

● Interseção: O conjunto fuzzy $A \cap B$ em U representa a interseção de dois conjuntos fuzzy A e B. Na interseção o uso do “min” indica a obtenção do menor dos valores de pertinência indicados para cada elemento do domínio, conforme:

$$A \cap B \Rightarrow \mu_{A \cap B}(x) = min\ [\mu_A(x), \mu_B(x)] \tag{2.18}$$

● Leis de DeMorgan: Também se aplicam ao caso dos conjuntos fuzzy. As seguintes equações mostram suas definições para dois conjuntos fuzzy A e B em U:

$$\overline{A \cup B} = \bar{A} \cap \bar{B} \tag{2.19}$$

$$\overline{A \cap B} = \bar{A} \cup \bar{B} \tag{2.20}$$

2.2.5- Operações Avançadas Com Conjuntos Fuzzy

Na seção anterior, foram introduzidas as operações mais comuns entre conjuntos fuzzy. Estas operações são o complemento, a união e a interseção. O estudo agora é referente a outros tipos de operadores que realizam as mesmas operações, os quais têm sua proposição fundamentada em axiomas. Para tal, é considerado $a = \mu_A(x)$ e $b = \mu_B(x)$.

● Complemento fuzzy: Seja $c: [0,1] \rightarrow [0,1]$ um mapeamento que transforma a função de pertinência de um conjunto fuzzy A na função de pertinência de $\overline{A}$, conforme:

$$c: [\mu_A(x)] = \mu_{\bar{A}}(x) \tag{2.21}$$

Considerando a Eq.2.16 e, para que a função c seja caracterizada como complemento, os seguintes axiomas dem ser satisfeitos:

(c1)$c(0) = 1$ e $c(1) = 0$ (condição de fronteira).

(c2) $\forall\, a, b \in [0,1]$ se $a < b \Rightarrow c(a) \geq c(b)$ (condição de não crescimento)

Qualquer função c: $[0,1] \rightarrow [0,1]$ que satisfaça os axiomas (c1) e (c2) é chamada de complemento fuzzy. A Tabela 1 apresenta algumas funções que implementam o complemento fuzzy, cujas propostas recebem denominações diferentes.

Tabela 1 – Funções que implementam o complemento fuzzy (onde $a = \mu_A(x)$).

Denominação	$c: [0,1] \rightarrow [0,1]$	Eq.
Classe Sugeno	$c_\lambda(a) = \dfrac{1-a}{1+\lambda a}\,, \qquad \lambda \in (-1, \infty)$	(2.22)
Classe Yager	$c_w(a) = (1-a^w)^{1/w}\,, \qquad w \in (0, \infty)$	(2.23)

A Fig.10 mostra os gráficos para a operação de complemento fuzzy.

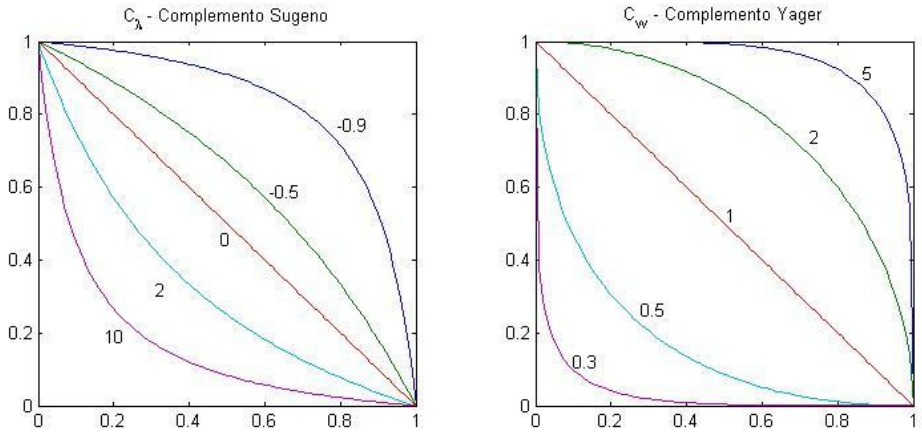


Figura 10 – Complementos fuzzy: Classes Sugeno e Yager.

É importante observar que para as duas classes relacionadas, em cada valor de λ ou w, dependendo do caso, é obtido um complemento fuzzy particular.

● União fuzzy – norma-s: Seja $s: [0,1] \times [0,1] \rightarrow [0,1]$ um mapeamento que transforma as funções de pertinência dos conjuntos fuzzy A e B na função de pertinência de $A \cup B$, conforme:

$$s: [\mu_A(x), \mu_B(x)] = \mu_{A \cup B}(x) \tag{2.24}$$

Considerando a Eq. 2.17 e, para que a função s possa se caracterizar como uma união, os quatro axiomas seguintes devem ser satisfeitos:

(s1) $s(1,1) = 1$, $s(0, a) = s(a, 0) = a$ (condição de fronteira).

(s2) $s(a, b) = s(b, a)$ (condição de comutatividade).

(s3) Se $a \leq a'$ e $b \leq b'$, então $s(a, b) \leq s(a', b')$ (condição de não decrescimento).

(s4) $s[s(a, b), c] = s[a, s(b, c)]$ (condição associativa).

Qualquer função $s: [0,1] \times [0,1] \rightarrow [0,1]$ que satisfaça os axiomas de (s1) a (s4) é chamada de norma–s. A Tabela 2 mostra algumas funções que implementam a norma-s com base nestes axiomas. Algumas dessas normas-s são apresentadas na Fig.11.

Tabela 2 – Funções que implementam a norma-s (onde $a = \mu_A(x)$ e $b = \mu_B(x)$).

Denominação	$s: [0,1] \times [0,1] \rightarrow [0,1]$	Eq.
Classe Dombi	$s_\lambda(a, b) = \dfrac{1}{1 + [\left(\frac{1}{a} - 1\right)^{-\lambda} + \left(\frac{1}{b} - 1\right)^{-\lambda}]^{-\frac{1}{\lambda}}}$, $\lambda \in (0, \infty)$	(2.25)
Classe Dubois-Prade	$s_\alpha(a, b) = \dfrac{a + b - ab - \min(a, b, 1 - a)}{\max(1 - a, 1 - b, \alpha)}$, $\alpha \in [0,1]$	(2.26)
Classe Yager	$s_w(a, b) = \min[1, (a^w + b^w)^{1/w}]$, $w \in (0, \infty)$	(2.27)
Soma drástica	$s_{ds}(a, b) = \begin{cases} a \ se \ b = 0 \\ b \ se \ a = 0 \\ 1 \ outro \ caso \end{cases}$	(2.28)
Soma Einstein	$s_{es}(a, b) = \dfrac{a + b}{1 + ab}$	(2.29)
Soma Algébrica	$s_{as}(a, b) = a + b - ab$	(2.30)
Máximo	$s_{max}(a, b) = \max[a, b]$	(2.31)

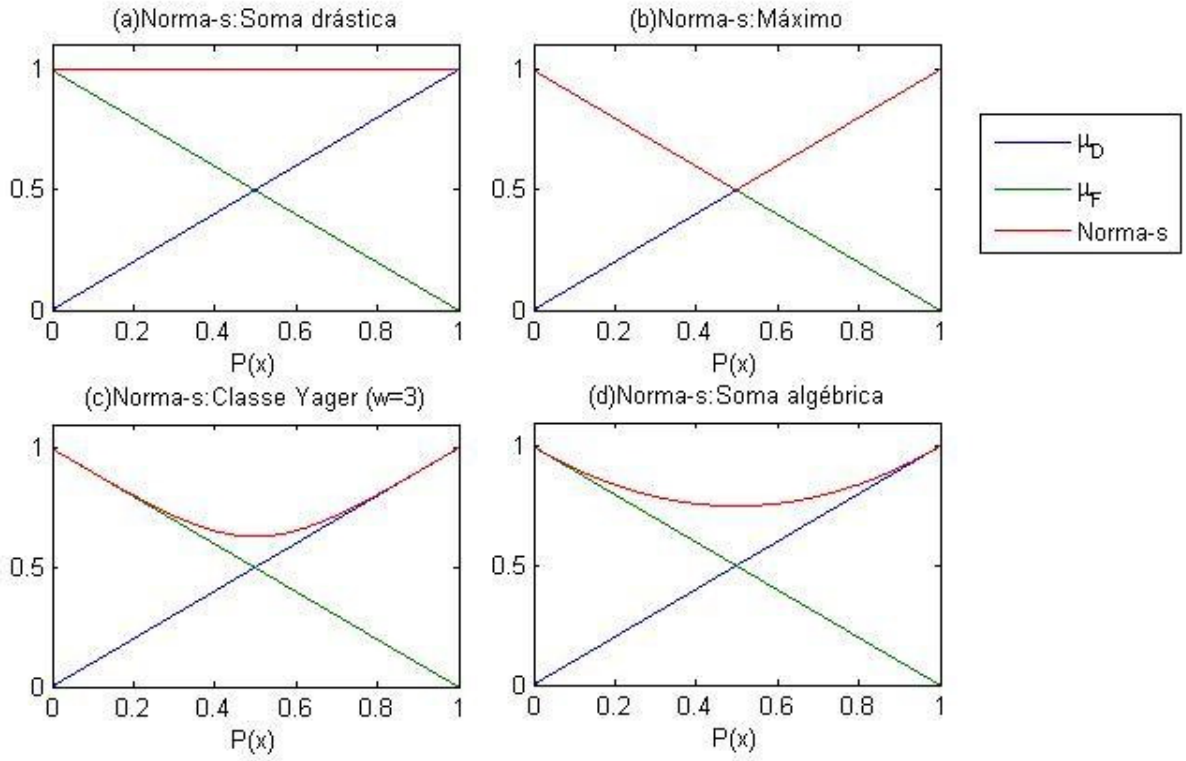


Figura 11- Gráficos para a norma-s

● Interseção fuzzy – norma-t: Seja $t: [0,1] \times [0,1] \rightarrow [0,1]$ um mapeamento que transforma as funções de pertinência dos conjuntos fuzzy A e B na função de pertinência de $A \cap B$, conforme:

$$t: [\mu_A(x), \mu_B(x)] = \mu_{A \cap B}(x) \tag{2.32}$$

Considerando a Eq. 2.18 e, para que a função t possa se caracterizar como uma união, os quatro axiomas seguintes devem ser satisfeitos:

(t1) $t(0,0) = 0,\ t(a,1) = t(1,a) = a$ (condição de fronteira).

(t2) $t(a,b) = t(b,a)$ (condição de comutatividade).

(t3) Se $a \leq a'$ e $b \leq b'$, então $t(a,b) \leq t(a',b')$ (condição de não decrescimento).

(t4) $t[t(a,b),c] = t[a,t(b,c)]$ (condição associativa).

Qualquer função $t: [0,1] \times [0,1] \rightarrow [0,1]$ que satisfaça os axiomas de (t1) a (t4) é chamada de norma–t. A Tabela 3 mostra algumas funções que implementam a norma-t com base nos respectivos axiomas. Alguns gráficos para a norma-t são apresentados na Fig.12.

Tabela 3 – Funções que implementam a norma-t (onde $a = \mu_A(x)$ e $b = \mu_B(x)$).

Denominação	$t: [0,1] \times [0,1] \rightarrow [0,1]$	Eq.
Classe Dombi	$t_\lambda(a,b) = \dfrac{1}{1 + [\left(\frac{1}{a} - 1\right)^\lambda + \left(\frac{1}{b} - 1\right)^\lambda]^{\frac{1}{\lambda}}}, \lambda \in (0, \infty)$	(2.33)
Classe Dubois-Prade	$t_\alpha(a,b) = \dfrac{ab}{\max(a,b,\alpha)}, \quad \alpha \in [0,1]$	(2.34)
Classe Yager	$t_w(a,b) = 1 - \min[1, ((1-a)^w + (1-b)^w)^{1/w}], w \in (0, \infty)$	(2.35)
Produto drástico	$t_{dp}(a,b) = \begin{cases} a \ se \ b = 1 \\ b \ se \ a = 1 \\ 0 \ outro \ caso \end{cases}$	(2.36)
Produto Einstein	$t_{ep}(a,b) = \dfrac{ab}{2 - (a + b - ab)}$	(2.37)
Produto Algébrico	$t_{ap}(a,b) = ab$	(2.38)
Mínimo	$t_{min}(a,b) = min\,[a,b]$	(2.39)

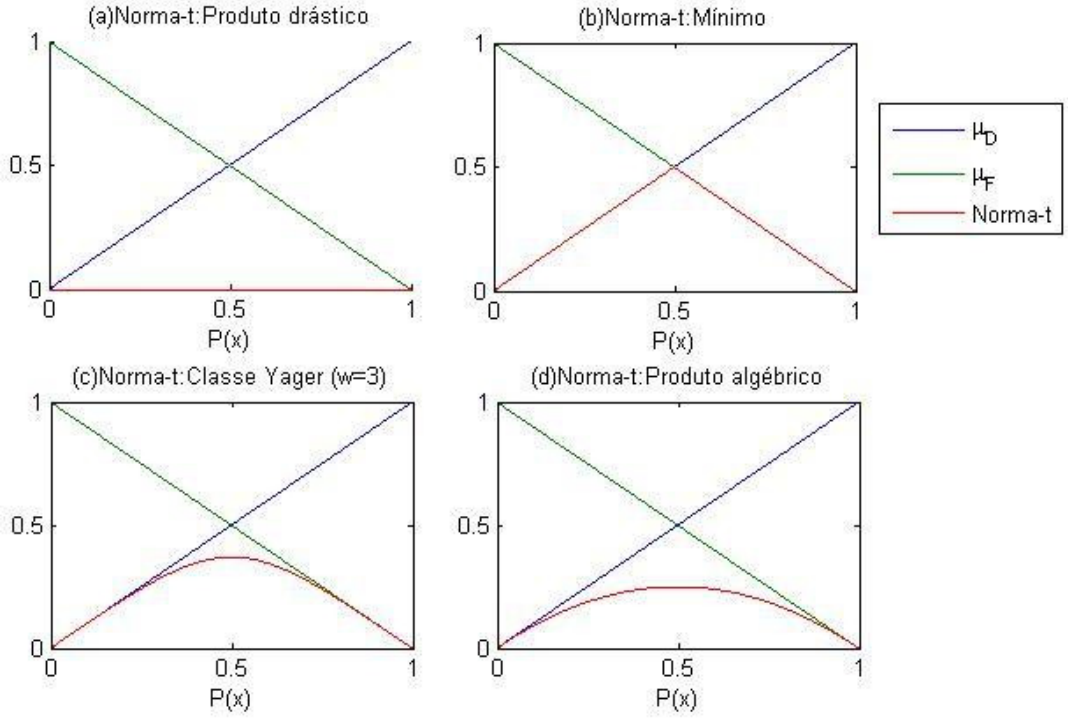


Figura 12 – Gráficos para a norma-t.

● Operadores de média: A união entre dois conjuntos fuzzy arbitrários A e B $(A \cup B)$, que venha a ser definida por qualquer norma-s, ocorre no intervalo $[\max(a,b), S_{ds}(a,b)]$. Já para a interseção $(A \cap B)$, definida por qualquer norma-t, ocorre no intervalo $[t_{dp}(a,b), \min(a,b)]$, ou seja, as operações de união e interseção não conseguem cobrir o intervalo entre $\min(a,b)$ e $\max(a,b)$. Para que se consiga cobrir o intervalo $[\min(a,b), \max(a,b)]$ são definidos os operadores de média, representados por v e, como nas norma-s e na norma-t, é uma função de $[0,1] \times [0,1]$ para $[0,1]$. São apresentadas na Tabela 4, algumas funções que implementam a média.

Tabela 4 – Funções que implementam a média (onde $a = \mu_A(x)$ e $b = \mu_B(x)$).

Denominação	v: $[0,1] \times [0,1] \rightarrow [0,1]$	Eq.
Médias Max-min	$v_\lambda(a,b) = \lambda \max(a,b) + (1-\lambda)\min(a,b)$, $\lambda \in [0,1]$	(2.40)
Meio generalizado	$v_\alpha(a,b) = \left(\frac{a^\alpha + b^\alpha}{2}\right)^{\frac{1}{\alpha}}$, $\alpha \in \mathbb{R}\ (\alpha \neq 0)$	(2.41)
"E" fuzzy	$v_p = p\min(a,b) + \frac{(1-p)(a+b)}{2}$, $p \in [0,1]$	(2.42)
"Ou" fuzzy	$v_\gamma(a,b) = \gamma \max(a,b) + \frac{(1-\gamma)(a+b)}{2}$, $\gamma \in [0,1]$	(2.43)

2.2.6 – Relações Fuzzy

Para que se possa ter a idéia do que venha a ser uma relação fuzzy, considera-se o conceito de relação entre conjuntos da lógica clássica.

Considerando dois conjuntos clássicos (não fuzzy) U e V, e o seu produto cartesiano $U \times V$, definido como segue:

$$U \times V = \{(u,v) \mid u \in U \ e \ v \in V\} \tag{2.44}$$

O conjunto $U \times V$ da Eq.(2.44) relaciona todos os pares ordenados de forma que $u \in U$ e $v \in V$. É importante observar que se a dimensão de U for diferente da dimensão de V, os produtos $U \times V$ e $V \times U$ serão diferentes conforme:

$$U \neq V \Rightarrow U \times V \neq V \times U \tag{2.45}$$

O produto cartesiano generalizado para n conjuntos $U_1, U_2, \cdots, U_n$ é mostrado conforme o seguinte:

$$U_1 \times U_2 \times \cdots \times U_n = \{(u_1, u_2, \cdots, u_n) \mid u_1 \in U_1, u_2 \in U_2, \cdots, u_n \in U_n\} \tag{2.46}$$

Uma relação (não fuzzy) entre conjuntos (não fuzzy) $U_1, U_2, \cdots, U_n$ é um subconjunto do produto cartesiano $U_1 \times U_2 \times \cdots \times U_n$, ou seja, se o conjunto $Q(U_1, U_2, \cdots, U_n)$ representar uma relação entre $U_1, U_2, \cdots, U_n$, então $Q(U_1, U_2, \cdots, U_n)$ será um subconjunto de $U_1 \times U_2 \times \cdots \times U_n$, conforme:

$$Q(U_1, U_2, \cdots, U_n) \subset U_1 \times U_2 \times \cdots \times U_n \tag{2.47}$$

Como exemplo, considere os conjuntos $U = \{1,2,3\}$, $V = \{2,3,4\}$ e o seu produto cartesiano $U \times V = \{(1,2), (1,3), (1,4), (2,2), (2,3), (2,4), (3,2), (3,3), (3,4)\}$.

Seja $Q(U,V)$ uma relação entre U e V onde "**o primeiro elemento é maior ou igual ao segundo elemento**". Então $Q(U,V) = \{(2,2), (3,2), (3,3)\}$ e pode ser representada pela função de pertinência μ_Q da Eq.(2.48) que, para este caso, define uma relação binária, cujos valores podem ser reunidos em uma matriz relacional conforme a Tabela 5.

$$\mu_Q = \begin{cases} 1 \ \ se \ (u_1, u_2, \cdots, u_n) \in Q(U_1, U_2, \cdots, U_n) \\ 0 \ \ para \ outros \ valores \end{cases} \tag{2.48}$$

Tabela 5 - Matriz relacional que representa $Q(U,V)$

		V		
		2	3	4
U	1	0	0	0
	2	1	0	0
	3	1	1	0

Analisando a Tabela 5, conclui-se que, numa relação clássica entre conjuntos, obtem-se uma relação crisp, que significa a existência da relação (valor "1") ou a sua inexistência (valor "0").

Como segundo exemplo, apresenta-se uma situação em que a relação binária não consiga ser feita, ou seja, é difícil fazer avaliação zero-um conforme apresentada na Tabela 5. Considere os conjuntos de cidades $U =$ {São Francisco, Honk-Kong, Tóquio} e $V =$ {Boston, Honk Kong}. Com base em Ue V, é definida uma relação, em $U \times V$, que classifique as distâncias entre as cidades dos dois conjuntos como "**muito longe**". Para a estrutura dos conjuntos clássicos, como definir o que é "muito longe"? Devido o conceito de "muito longe" não ser bem definido para conjuntos clássicos, não se pode aplicar relações clássicas, necessitando-se de um sistema numérico especial para caracterizar esta relação.

Como proposta, pode-se usar um número no intervalo [0,1] que represente o grau de "muito longe" e, com isso obter uma matriz relacional conforme a Tabela 6.

Tabela 6 – Matriz relacional (fuzzy) para "muito longe".

	Boston	Honk Kong
São Francisco	0.3	0.9
Honk Kong	1	0
Tóquio	0.95	0.1

Com este exemplo, é mostrada a generalização do conceito de relação clássica, o que permite a formulação de mais problemas do mundo real. Isso introduz o conceito de relação fuzzy [6].

Uma relação fuzzy é um conjunto fuzzy definido no produto cartesiano de conjuntos "cirsp" $U_1, U_2, \cdots, U_n$. Uma relação fuzzy Q em $U_1, U_2, \cdots, U_n$ é definida como:

$$Q = \{((u_1, u_2, \cdots, u_n), \mu_Q(u_1, u_2, \cdots, u_n)) | (u_1, u_2, \cdots, u_n) \in U_1, U_2, \cdots, U_n\} \quad (2.49)$$

Onde $\mu_Q: U_1, U_2, \cdots, U_n \rightarrow [0,1]$.

2.2.7 – Projeções e Extensões Cilíndricas

Considere Q uma relação fuzzy em $U \times V$. A projeção de Q sobre U é a relação fuzzy Q_1 em U que é definida pela função de pertinência μ_{Q_1}, conforme o seguinte:

$$\mu_{Q_1}(x) = \max_{y \in V} \mu_Q(x, y) \tag{2.50}$$

De forma semelhante, a projeção de Q sobre V é a relação fuzzy Q_2 em V que é definida pela função de pertinência μ_{Q_2}, como segue:

$$\mu_{Q_2}(y) = \max_{x \in U} \mu_Q(x, y) \tag{2.51}$$

Considere Q_3 um conjunto fuzy em U, sua extensão cilíndrica para $U \times V$ é a relação fuzzy Q_{3E} em $U \times V$, definida pela função de pertinência $\mu_{Q_{3E}}$, conforme o seguinte:

$$\mu_{Q_{3E}}(x, y) = \mu_{Q_3}(x) \tag{2.52}$$

2.2.8- Composições de Relações Fuzzy

Sejam $P(U,V)$ e $Q(V,W)$ duas relações binárias crisp que compartilham um conjunto V em comum. A composição de $P\ e\ Q$, denotada por $P \circ Q$ é uma relação definida em $U \times W$ tal que $(x,z) \in P \circ Q$ se e somente se existe pelo menos um $y \in V$ tal que $(x,y) \in P$ e $(y,z) \in Q$. Usando a representação das relações por função de pertinência, conforme a Eq.2.48, tem-se a seguinte definição para a composição: $P \circ Q$ é a composição de $P(U,V)\ e\ Q(V,W)$ se e somente se:

$$\mu_{P \circ Q}(x,z) = \max_{y \in V} t[\mu_P(x,y), \mu_Q(y,z)] \tag{2.53}$$

para qualquer $(x,z) \in U \times W$, onde t é qualquer norma-t.

Generalizando o conceito de composição para as relações fuzzy, e usando a Eq.2.53 para definir a composição $P \circ Q$ das relações fuzzy, supondo $P\ e\ Q$ duas relações fuzzy. Devido ao operador norma-t na Eq.(2.53), pode-se obter uma variedade de fórmulas e, para cada norma-t (ver Tabela 3), se consegue uma composição particular. Em especial, é possível

citar as duas composições mais comuns na literatura, que são a composição max-min e a composição max-prod, definidas pelas funções de pertinência conforme:

$$\mu_{P\circ Q}(x,z) = \max_{y\in V} \min\left[\mu_P(x,y)\,,\mu_Q(y,z)\right] \tag{2.54}$$

$$\mu_{P\circ Q}(x,z) = \max_{y\in V}\ \left[\mu_P(x,y)\,\mu_Q(y,z)\right] \tag{2.55}$$

onde $(x,z) \in U \times W$.

2.2.9- O Princípio da Extensão

É o princípio, no qual se constitui uma identidade básica, que permite com que o domínio de uma função seja extendido de pontos crisp em U para conjuntos fuzzy em U.

Seja, uma função do conjunto crisp U para o conjunto crisp V ($f: U \to V$). Suponha que um conjunto fuzzy A em U seja dado e deseja-se determinar o conjunto fuzzy $B = f(A)$ em V induzido por f. Se f for um mapeamento um-para-um, então pode-se definir:

$$\mu_B(y) = \ \mu_A[f^{-1}(y)] \ \text{ onde } y \in V \tag{2.56}$$

Na Eq.(2.56) $f^{-1}(y)$ é a função inversa de f, ou seja, $f[f^{-1}(y)] = y$. Se f não é um-para-um, então uma ambigüidade acontece quando dois ou mais pontos distintos em U com diferentes valores de pertinência em A são mapeados no mesmo ponto em V. Para resolver esta ambigüidade, atribuímos o maior dos valores de pertinência para $\mu_B(y)$. De forma mais geral, a função de pertinência de B é definido como:

$$\mu_B(y) = \ \max_{x\in f^{-1}(y)} \mu_A\,(x), \quad y \in V \tag{2.57}$$

Onde $f^{-1}(y)$ significa o conjunto de todos os pontos $x \in U$ de maneira que $f(x) = y$. A identidade (2.57) é chamada de princípio da extensão.

2.2.10- Variáveis Linguísticas e as Regras Fuzzy SE-ENTÃO

Quando é atribuído a uma determinada variável um valor que seja um número, ela é definida como variável numérica. Isto é muito comum quando se desenvolve uma formulação precisa de um sistema.

Sob outro aspecto, é muito comum se estimar valores e associar, a estes valores, termos que possam expressar aproximadamente o seu quantitativo, como por exemplo: “O tanque está cheio” (ou seja: volume_do_tanque = cheio), “O homem é baixo” (ou seja: altura_do_homem = baixo). Pode-se assim, por definição intuitiva, estabelecer que, quando as variáveis passam a assumir palavras em linguagem natural, elas são chamadas de variáveis linguísticas.

Para que se possa conseguir formular tais palavras em termos matemáticos, deve-se inicialmente caracterizá-las através de conjuntos fuzzy, os quais são definidos no universo de discurso onde a variável lingüística é definida.

Em 1975, Zadeh formalizou a definição de variável lingüística, a qual se caracteriza pelos parâmetros (X, T, U, M) [2][3][6], onde X é o nome da variável lingüística, T é o conjunto de valores lingüísticos (rótulos) que X pode assumir, U é o intervalo numérico (domínio físico) para o qual X tem seus valores quantitativos (valores crisp) e M que é uma regra semântica para relacionar T com U. O presente trabalho usa esta formalização na implementação da classe que cria os objetos que representam as variáveis lingüísticas.

As variáveis lingüísticas são consideradas extensões das variáveis numéricas, uma vez que elas podem assumir como seus valores conjuntos fuzzy.

Por serem os elementos mais fundamentais na representação do conhecimento humano, as variáveis lingüísticas e o seu conceito assumem um papel importante. Um exemplo que ilustra este fato é a estimativa feita por um ser humano acerca da velocidade de um automóvel deslocando-se em uma estrada, sua resposta certamente será algo que descreva o valor da velocidade: alta, ou seja, uma palavra, enquanto que, se for usado um sensor (radar) para medir esta velocidade, obtém-se um número: 137 Km/h.

Com a definição de variável lingüística, pode-se então usar palavras como valores para as mesmas. Estes valores geralmente são termos compostos, por exemplo: $x = x_1 x_2 \dots x_n$, que é a concatenação de termos chamados de “atômicos” $x_1, x_2, \dots, x_n$, os quais se classificam em três tipos:

1- Termos primários (rótulos dos conjuntos fuzzy): “baixo”, “médio”, “alto”.

2- Complemento: “Não” e conexões: “e” e “ou”.

3- Limites (“Hedges”): “Muito”, “Ligeiramente”, “Mais ou menos”.

Pode-se aplicar agora o conceito de variável lingüística na formulação das regras fuzzy SE-ENTÃO, que constituem a base do conhecimento dos sistemas fuzzy.

Uma regra SE-ENTÃO é uma declaração condicional expressa conforme a Eq.(2.58).

SE <proposição fuzzy>, ENTÃO <proposição fuzzy> (2.58)

Uma proposição fuzzy pode ser de dois tipos:

1- Proposição fuzzy atômica: é uma declaração do tipo: x é A, onde x é a variável lingüística e A é o valor lingüístico (A é um conjunto fuzzy definido no domínio físico de x).

2- Proposições fuzzy compostas: É a composição de proposições fuzzy atômicas com o uso dos conectivos "e", "ou" e "não", que representam a interseção fuzzy ("∧"), união fuzzy ("∨") e o complemento fuzzy ("$\bar{\ }$") respectivamente. Nas proposições fuzzy compostas, cada proposição fuzzy atômica é independente, significando que cada variável lingüística x seja diferente, como no geral.

As proposições fuzzy devem ser entendidas como relações fuzzy, então é importante saber como determinar as funções de pertinência destas relações fuzzy:

● Para o conectivo "e" usa-se a interseção fuzzy:

Considere que x, y sejam variáveis lingüísticas em U e V. A e B dois conjuntos fuzzy em U e V. Se, como proposição fuzzy composta tem-se: "x é A e y é B", então esta proposição é interpretada como a relação fuzzy $A \cap B$ em $U \times V$ com função de pertinência $\mu_{A\cap B}(x,y) = t[\mu_A(x), \mu_B(y)]$, onde t: $[0,1] \times [0,1] \rightarrow [0,1]$ é qualquer norma-t.

● Para o conectivo "ou" usa-se a união fuzzy:

Se a proposição fuzzy composta é: "x é A ou y é B", sua interpretação é como a relação fuzzy $A \cup B$ em $U \times V$, com função de pertinência $\mu_{A\cup B}(x,y) = s[\mu_A(x), \mu_B(y)]$, onde $s[0,1] \times [0,1] \rightarrow [0,1]$ é qualquer norma-s.

● Para o conectivo "não" usa-se o complemento fuzzy:

Se a proposição fuzzy composta é: "$FP = (x \text{ é } S \; e \; x \; não \text{ é } F) ou \; x \text{ é } M$", que é uma relação no espaço produto $[0, V_{max}]^3$, a sua função de pertinência será $\mu_{FP}(x_1, x_2, x_3) = s\{t[\mu_S(x_1), c(\mu_F(x_2))]\mu_M(x_3)\}$, onde s é norma-s, t é norma-t e c é o complemento fuzzy.

Devido à interpretação das proposições fuzzy como relações fuzzy e, considerando que as mesmas fazem parte das regras fuzzy SE-ENTÃO, é necessário saber como interpretar a operação destas regras fuzzy SE-ENTÃO.

Partindo do cálculo proposicional clássico, uma declaração condicional do tipo "SE p ENTÃO q" é representada simbolicamente por "$p \rightarrow q$", sendo p e q duas variáveis proposicionais. Esta representação, feita pelo conectivo "→", denominada de implicação, cuja operação é definida pela Tabela 7, onde V significa verdadeiro e F falso.

Tabela 7 – Implicação $p \rightarrow q$

p	q	$p \rightarrow q$
V	V	V
V	F	F
F	V	V
F	F	V

Com base na Tabela 7, é possível verificar que $p \rightarrow q$ equivale ao seguinte:

$$\bar{p} \vee q \tag{2.59}$$

$$(p \wedge q) \vee \bar{p} \tag{2.60}$$

onde "¯", "∨" e "∧" significam as operações lógicas "não", "ou" e "e" respectivamente.

Como é possível substituir p e q das regras SE-ENTÃO por proposições fuzzy, tornando-as regras SE-ENTÃO fuzzy, também é possível substituir os operadores "¯", "∨" e "∧" das Eqs.(2.59) e (2.60) pelos operadores complemento fuzzy, união fuzzy e interseção fuzzy. Conforme foi estudado nas seções 2.2.4 e 2.2.5, há uma variedade de operadores fuzzy que podem ser substituídos nas Eqs.(2.59) e (2.60), o que dá margem a uma variedade de interpretações diferentes para as regras fuzzy SE-ENTÃO.

Para que sejam mostradas essas diferentes interpretações, a Eq.(2.58) será escrita conforme mostrado abaixo, onde FP_1 e FP_2são proposições fuzzy

$$\text{SE } <FP_1>, \text{ ENTÃO } <FP_2> \tag{2.61}$$

Também serão substituídos p e q, nas Eqs.(2.59) e (2.60), por FP_1 e FP_2 respectivamente.

Considera-se que FP_1 seja uma relação fuzzy definida em $U = U_1 \times U_2 \times ... \times U_n$ e FP_2 uma relação fuzzy definida em $V = V_1 \times V_2 \times ... \times V_n$ e x e y variáveis lingüísticas (vetores) em U e V, respectivamente. A Tabela 8 relaciona alguns tipos de implicação conhecidas com os seus métodos de obtenção [6].

Tabela 8 – Implicações fuzzy propostas.

Implicação	Método	Formulação	Eq.
Dienes-Rescher	Substitui na Eq.(2.31) o "$^-$" e o "$\vee$" pelo complemento fuzzy básico e pela união fuzzy	$\mu_{Q_D}(x,y) = \max\left[1 - \mu_{\mathrm{FP}_1}(\mathrm{x}), \mu_{\mathrm{FP}_2}(\mathrm{y})\right]$	(2.62)
Lukasiewicz	Substitui na Eq.(2.31) o "$^-$" pelo complemento fuzzy básico e o e $\vee$ pela norma-s Yager com w=1,	$\mu_{Q_L}(x,y) = \min\left[1,1 - \mu_{\mathrm{FP}_1}(\mathrm{x}) + \mu_{\mathrm{FP}_2}(\mathrm{y})\right]$	(2.63)
Zadeh	Obtida a partir da Eq.(2.32) pelo uso do complemento fuzzy básico, da união fuzzy básica e da interseção	$\mu_{Q_Z}(x,y) = \max\left[\min\left(\mu_{\mathrm{FP}_1}(\mathrm{x}), \mu_{\mathrm{FP}_2}(\mathrm{y})\right),1 - \mu_{\mathrm{FP}_1}(\mathrm{x})\right]$	(2.64)
Gödel	É uma implicação muito conhecida na lógica clássica. É obtida a partir da sua generalização às proposições fuzzy.	$\mu_{Q_G}(x,y) = \begin{cases} 1 & se\ \mu_{FP_1}(x) \le \mu_{FP_2}(y) \\ \mu_{FP_2}(y) & para\ outros\ valores \end{cases}$	(2.65)

Analisando a equivalência e o que esta possa significar nas Eqs.(2.59) e (2.60) para as duas situações, uma em que p e q são proposições crisp, e a outra, em que p e q são proposições fuzzy, observa-se que, para p e q com proposições crisp, ou seja, só podem assumir valores verdadeiro ou falso, $p \rightarrow q$ é uma implicação global e a Tabela 7 deve cobrir

todos os casos possíveis. Porém, para p e q com proposições fuzzy, $p \rightarrow q$ poderá ser somente uma implicação local, sendo que $p \rightarrow q$ terá um grande valor verdadeiro apenas quando p e q tiverem grandes valores.

Exemplificando, considere a seguinte regra SE-ENTÃO fuzzy:

SE velocidade é alta, ENTÃO resistência é alta.

Cria-se aqui o contexto de uma situação local, pois esta regra não faz consideração alguma com relação à "velocidade é média" ou à "velocidade é lenta" e, com isso a regra fuzzy SE $< FP_1 >$, ENTÃO $< FP_2 >$ deve ser interpretada como SE $< FP_1 >$, ENTÃO $< FP_2 >$ SENÃO $< NADA >$, onde o $NADA$ significa que não existe regra e, sob o ponto de vista lógico, $p \rightarrow q = p \wedge q$. Usando-se min ou o produto algébrico para a interseção ($\wedge$), obtêm-se as implicações de Mamdani, onde a regra fuzzy SE $< FP_1 >$, ENTÃO $< FP_2 >$ como uma relação fuzzy Q_{MM} ou Q_{MP} em $U \times V$ com funções de pertinência como segue:

$$\mu_{Q_{MM}}(x, y) = \min\left[\mu_{FP_1}(x), \mu_{FP_2}(y)\right] \tag{2.66}$$

$$\mu_{Q_{MP}}(x, y) = \mu_{FP_1}(x).\mu_{FP_2}(y) \tag{2.67}$$

As implicações de Mamdani são as mais usuais em aplicações com sistemas fuzzy e controle fuzzy, sob a justificativa de que as regras fuzzy SE-ENTÃO sejam locais. Entretanto, há quem discorde, podendo argumentar, para o caso do exemplo da velocidade que, quando se diz: "SE velocidade é alta, ENTÃO resistência é alta", fica indicado implicitamente que "SE velocidade é lenta, ENTÃO resistência é baixa". Se a idéia for criar as regras neste sentido, então as regras fuzzy SE-ENTÃO passam a ser não locais. Com isso, deve ser observado que, quando se representa o conhecimento humano em termos de regras fuzzy SE-ENTÃO, pessoas diferentes têm interpretações diferentes. Isso sugere a necessidade de implicações diferentes, então, quando os especialistas humanos pensarem que suas regras sejam locais, as implicações de Mamdani se aplicam e, quando não locais, as implicações globais, apresentadas na Tabela 8, são aplicadas.

2.2.11- Lógica Fuzzy e Raciocínio Aproximado

A palavra “lógica” é empregada para definir o estudo dos métodos e princípios do raciocínio, onde o raciocínio, por sua vez significa obter, a partir das proposições existentes, novas proposições. Na lógica clássica, as proposições podem assumir ou o valor 0, que significa falso, ou o valor 1, que significa verdadeiro. Na lógica fuzzy, que é uma generalização da lógica clássica, estes valores pertencem ao intervalo [0,1]. Esta generalização apresenta a particularidade de permitir o desenvolvimento de um raciocínio aproximado onde se podem deduzir conclusões imprecisas (proposições fuzzy) partindo de premissas imprecisas (proposições fuzzy).

Na lógica clássica, a tabela verdade é a representação das relações entre as proposições. Um exemplo de tabela verdade fundamental com algumas operações lógicas mais usuais, como conjunção “$\vee$”, disjunção “$\wedge$”, implicação “$\rightarrow$”, equivalência “$\leftrightarrow$” e negação “$\bar{}$”, é apresentado na Tabela 9.

A partir de n proposições é possível definir uma nova proposição, também denominada de função lógica, como uma função, a qual, combinando as n proposições, atribui um valor verdade particular à nova proposição. Dependendo do valor de n, pode-se ter um número muito grande de combinações (2^{2^n}), o que torna bastante difícil apresentar todas estas combinações na forma de uma tabela verdade. Por isso, é muito comum se representar as fórmulas lógicas através de um número reduzido de operações lógicas básicas formando um conjunto completo de primitivas.

Tabela 9 – Operações lógicas mais usuais

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\bar{p}$
T	T	T	T	T	T	F
T	F	F	T	F	F	F
F	T	F	T	T	F	T
F	F	F	F	T	T	T

O conjunto completo de primitivas mais comumente utilizado reúne operações de conjunção “$\vee$”, disjunção “$\wedge$”, e negação “$\bar{}$”. Combinando estas primitivas nas fórmulas lógicas, pode-se formar qualquer outra função lógica. Recursivamente pode-se definir da seguinte forma:

(a) Os valores verdade 0 e 1 são fórmulas lógicas.

(b) Se p é uma proposição, então p e $\bar{p}$ são fórmulas lógicas.

(c) Se p e q são fórmulas lógicas, então $p \vee q$ e $p \wedge q$ também são fórmulas lógicas.

(d) Apenas são fórmulas lógicas aquelas definidas por (a), (b) e (c).

Chama-se tautologia para a proposição representada por uma fórmula lógica, a qual é sempre verdadeira, sem considerar o valor verdade das proposições básicas que fazem parte desta fórmula. Caso esta proposição se apresente sempre falsa, é chamada de contradição.

Como exemplos de tautologias considerem-se as seguintes fórmulas lógicas:

$$(p \rightarrow q) \leftrightarrow (\bar{p} \vee q) \tag{2.68}$$
$$(p \rightarrow q) \leftrightarrow ((p \wedge q) \vee \bar{q}) \tag{2.69}$$

Para que se prove as Eqs.(2.68) e (2.69) é usado o método da tabela verdade – Tabela 10, na qual são listados todos os valores possíveis para p e q e, aplicando-os nas respectivas fórmulas lógicas, são levantados os seus resultados com o objetivo de comprovar as tautologias.

Tabela 10 – Prova das Eqs.(2.68) e (2.69)

p	q	$p \rightarrow q$	$\bar{p} \vee q$	$(p \wedge q) \vee \bar{q}$	$(p \rightarrow q) \leftrightarrow (\bar{p} \vee q)$	$(p \rightarrow q) \leftrightarrow ((p \wedge q) \vee \bar{q})$
T	T	T	T	T	T	T
T	F	F	F	F	T	T
F	T	T	T	T	T	T
F	F	T	T	T	T	T

Para que se façam inferências dedutivas, lançar-se mão de várias tautologias, às quais se chamam de regras de inferência. As mais utilizadas são o Modus Ponens, o Modus Tollens e o Silogismo Hipotético.

O Modus Ponens estabelece que, para duas proposições p e $p \rightarrow q$ (premissas), o valor verdade da proposição q (conclusão) deve ser inferido. A sua representação simbólica é a seguinte:

$$(p \wedge (p \rightarrow q)) \rightarrow q \tag{2.70}$$

Outra forma de se visualizar o que estabelece o Modus Ponens é através da seguinte representação intuitiva:

Premissa 1: x é A
Premissa 2: SE x é A, ENTÃO y é B
Conclusão: y é B

O Modus Tollens estabelece que, dadas duas proposições $\bar{q}$ e $p \rightarrow q$, o valor verdade da proposição $\bar{p}$ deve ser inferido. A sua representação simbólica é a seguinte:

$$(\bar{q} \wedge (p \rightarrow q)) \rightarrow \bar{p} \tag{2.71}$$

Que através da representação intuitiva fica:

Premissa 1: y não é B
Premissa 2: SE x é A, ENTÃO y é B
Conclusão: x não é A

O Silogismo Hipotético estabelece que dadas duas proposições $p \rightarrow q$ e $q \rightarrow r$, o valor verdade para $p \rightarrow r$ poderia ser inferido. A sua representação simbólica é a seguinte:

$$((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r) \tag{2.72}$$

Que intuitivamente pode ser representado como segue:

Premissa 1: SE x é A, ENTÃO y é B
Premissa 2: SE y é B, ENTÃO z é C
Conclusão: SE x é A, ENTÃO z é C

Na lógica fuzzy têm-se proposições fuzzy, as quais são representadas por conjuntos fuzzy. Seus princípios fundamentais se baseiam nas propostas do Modus Ponens generalizado, Modus Tollens generalizado e do Siligismo Hipotético generalizado, com o objetivo de fornecer fundamentos para um raciocínio aproximado, baseado em proposições imprecisas, através do uso de conjuntos fuzzy.

No Modus Ponens generalizado é estabelecido que, dadas duas proposições fuzzy x é A' e SE x é A ENTÃO y é B, deve-se inferir uma nova proposição fuzzy y é B', tal que

quanto mais próximo A' é de A, mais próximo B' é de B, onde A', A, B' e B são conjuntos fuzzy. Portanto:

Premissa 1: x é A'
Premissa 2: SE x é A, ENTÃO y é B
Conclusão: y é B'

O Modus Tollens generalizado estabelece que dadas duas proposições fuzzy y é B' e SE x é A ENTÃO y é B, deve-se inferir uma nova proposição fuzzy y é A' tal que quanto maior a diferença entre B' e B, maior a diferença entre A' e A, onde A', A, B' e B são conjuntos fuzzy. Logo:

Premissa 1: y é B'
Premissa 2: SE x é A, ENTÃO y é B
Conclusão: x é A'

No Silogismo Hipotético generalizado é estabelecido que, dadas duas proposições fuzzy SE x é A ENTÃO y é B e SE y é B' ENTÃO z é C, é possível inferir uma nova proposição fuzzy SE x é A ENTÃO z é C', tal que quanto mais próximo B é de B', mais próximo C' é de C, onde A, B, B', C e C' são conjuntos fuzzy. Então:

Premissa 1: SE x é A, ENTÃO y é B
Premissa 2: SE y é B', ENTÃO z é C
Conclusão: SE x é A, ENTÃO z é C'

Considerando a Fig.13 e, supondo que se tenha a curva $y = f(x)$ de $x \in U$ para $y \in V$, sendo dado $x = a$, então, partindo de $x = a$, pode-se inferir $y = b = f(a)$.

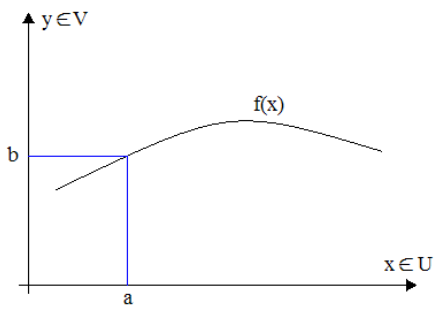


Figura 13 – Inferência de $y = b$, partindo de $x = a$ e $y = f(x)$.

Este processo de inferência de y=b, quando generalizado, é definido como a regra composicional de inferência.

Para esta generalização considere a Fig.14. Assumindo que a seja um intervalo e $f(x)$ uma função intervalar. Para se encontrar o intervalo b, que é inferido a partir de a e de $f(x)$, primeiro se constrói o conjunto cilíndrico a_E com base a e encontra-se a interseção I com a curva intervalar. Então se projeta I em V obtendo-se o intervalo b.

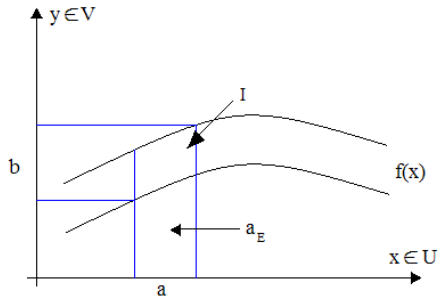


Figura 14 – Inferência do intervalo b, partindo do intervalo a e da função intervalar $f(x)$.

Assumindo que A' seja um conjunto fuzzy em U e Q uma relação fuzzy em $U \times V$. Seguindo o mesmo processo feito para a situação da Fig.14 e ilustrado na Fig.15, forma-se a extensão cilíndrica ${A'}_E$ de A' e, da sua interseção com Q, obtem-se um conjunto fuzzy ${A'}_E \cap Q$, análogo à interseção I na Fig.14. Fazendo-se a projeção de ${A'}_E \cap Q$ no eixo y, obtem-se o conjunto fuzzy B'.

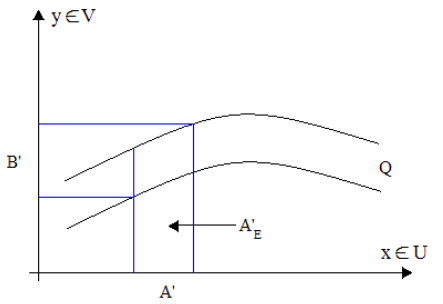


Figura 15 – Inferência de um conjunto fuzzy B' a partir de um conjunto fuzzy A' e uma relação fuzzy Q.

Mais especificamente, dado $\mu_{A'}(x)$ e $\mu_Q(x,y)$, tem-se:

$$\mu_{{A'}_E}(x,y) = \mu_{A'}(x) \tag{2.73}$$

Logo:

$$\mu_{A'_E \cap Q}(x,y) = t[\mu_{A'_E}(x,y), \mu_Q(x,y)] \\ = t[\mu_{A'}(x), \mu_Q(x,y)] \tag{2.74}$$

$$\mu_{B'}(y) = \sup_{x \in U} t[\mu_{A'}(x), \mu_Q(x,y)] \tag{2.75}$$

A Eq.(2.75) é chamada de regra composicional de inferência. Na literatura, geralmente usa-se o símbolo "$\star$" para o operador norma-t, então a Eq.(2.75) também é escrita na forma, chamada de composição sup-star, como segue:

$$\mu_{B'}(y) = \sup_{x \in U}[\mu_{A'}(x) \star \mu_Q(x,y)] \tag{2.76}$$

Já foi visto que uma regra fuzzy SE-ENTÃO é interpretada como uma relação fuzzy no produto cartesiano nos domínios de x e y. Portanto, diferentes princípios de implicação resultam em diferentes relações fuzzy.

Com base na Eq.(2.75), é possível determinar as fórmulas para os cálculos das conclusões para o Modus Ponens generalizado, para o Modus tollens generalizado e para o Silogismo Hipotético generalizado.

● Modus Ponens generalizado: Seja a premissa x é A', representada pelo conjunto fuzzy A' em U e a premissa SE x é A ENTÃO y é B, representada pela relação $A \rightarrow B$ em $U \times V$, e a conclusão y é B', representa pelo conjunto fuzzy B' em V, que é inferido como:

$$\mu_{B'}(y) = sup_{x \in U}\, t[\mu_{A'}(x), \mu_{A \rightarrow B}(x,y)] \tag{2.77}$$

●Modus Tollens generalizado: Seja a premissa y é B', representada pelo conjunto B' em V, e a premissa SE x é A ENTÃO y é B, representada pela relação fuzzy $A \rightarrow B$ em $U \times V$ e a conclusão x é A, representada pelo conjunto fuzzy A' em U que e é inferido como:

$$\mu_{A'}(x) = sup_{x \in U}\, t[\mu_{B'}(y), \mu_{A \rightarrow B}(x,y)] \tag{2.78}$$

●Silogismo Hipotético: Seja a premissa SE x é A ENTÃO y é B, representada pela relação fuzzy $A \rightarrow B$ em $U \times V$ e a premissa SE y é B' ENTÃO z é C, representada pela relação $B' \rightarrow C$ em $V \times W$, a conclusão SE x é A ENTÃO z é C', representada pela relação $A \rightarrow C'$ em $U \times W$, que é inferida como:

$$\mu_{A \rightarrow C'}(x, z) = sup_{y \in V}\, t[\mu_{A \rightarrow B}(x, y), \mu_{B' \rightarrow C}(y, z)] \tag{2.79}$$

2.2.12- Base de Regras Fuzzy e Máquina de Inferência Fuzzy.

Serão estudados agora os detalhes da máquina de inferência de um sistema fuzzy e da sua base de regras.

Considere um sistema fuzzy com múltiplas entradas e uma saída, uma vez que para os sistemas com múltiplas saídas é verdadeira a sua decomposição em um conjunto de sistemas de uma saída.

2.2.12.1- Base de Regras Fuzzy

É formada por um conjunto regras fuzzy SE-ENTÃO. É considerada a parte mais importante do sistema fuzzy, pois os demais componentes trabalham em função de se implementar essas regras de forma racional e eficiente. É possível de se ver a forma como se apresentam as regras fuzzy SE-ENTÃO de uma base de regras, como segue:

$$Ru^{(l)}: SE < x_1 \text{ é } A_1^l > e < x_2 \text{ é } A_2^l > e \ldots e < x_n \text{ é } A_n^l >, ENTÃO < y \text{ é } B^l > \tag{2.80}$$

onde A_i^l e B_i^l são conjuntos fuzzy em $U_i \subset R$ e $V \subset R$, respectivamente e $x = (x_1, x_2, \ldots, x_n)^T \in U$ e $y \in V$ são as variáveis linguísticas de entrada e de saída do sistema fuzzy, respectivamente. Para uma base de regras com M regras, tem-se $l = 1,2,\ldots, M$. As regras no formato da Eq.(2.80) recebem a denominação de regras fuzzy SE-ENTÃO canônicas, pelo fato de que estas regras podem incluir outros tipos regras fuzzy e proposições fuzzy (casos especiais) que são apresentados através do seguinte lema:

Lema L1: As regras fuzzy SE-ENTÃO canônicas, na forma da Eq.(2.80), incluem os seguintes casos:

(a) “Regras parciais”:

$$SE < x_1 \text{ é } A_1^l > e < x_2 \text{ é } A_2^l > e \ldots e < x_m \text{ é } A_m^l >, ENTÃO < y \text{ é } B^l > \quad (2.81)$$

(b) “Regras Ou”:

$$SE < x_1 \text{ é } A_1^l > e \ldots e < x_m \text{ é } A_m^l > ou < x_{m+1} \text{ é } A_{m+1}^l > e \ldots e < x_n \text{ é } A_n^l >, ENTÃO < y \text{ é } B^l > \quad (2.82)$$

onde $m < n$.

(c) “Declarações fuzzy simples”:

$$y \text{ é } B^l \quad (2.83)$$

(d) “Regras graduais”:

$$Quanto\ menor\ x, maior\ y \quad (2.84)$$

(e) “Regras não fuzzy”: Produção de regras convencionais, ou seja, situações em que A_i^l e B^l só podem assumir valores ou 0 ou 1, então as regras da forma da Eq.(2.80) se tornam regras não fuzzy.

2.2.12.2- Propriedades do Conjunto de Regras

Devido à base de regras fuzzy ser formada por um conjunto de regras, pelo relacionamento entre essas regras e, pelas regras como um todo, questionamentos são levantados: As regras fuzzy cobrem todas as situações que podem ocorrer com o sistema fuzzy? Existem conflitos entre as regras?

Definição 2.1: Uma base de regras fuzzy é considerada completa se, para qualquer $x \in U$, existe pelo menos uma regra na base que satisfaça:

$$\mu_{A_i^l}(x_i) \neq 0 \quad (2.85)$$

Para todo $i = 1, 2, \ldots, n$.

Definição 2.2: Um conjunto de regras fuzzy SE-ENTÃO é consistente se não há regras com a mesma parte SE e diferentes partes ENTÃO.

Definição 2.3: Um conjunto de regras fuzzy SE-ENTÃO é contínuo se não exitem regras de vizinhança cujos conjuntos fuzzy da parte ENTÃO têm interseção vazia.

2.2.12.3- Máquina de Inferência Fuzzy

A máquina de inferência fuzzy combina as regras SE-ENTÃO, da base de regras fuzzy, através do uso da lógica fuzzy em um mapeamento do conjunto fuzzy A' em U para o conjunto fuzzy B' em V.

Se a base de regras fuzzy consiste de apenas uma regra, este mapeamento é especificado pelo Modus Ponens generalizado Eq.(2.77), porém, na prática, bases de regras fuzzy geralmente são constituídas por mais de uma regra, portanto, inferir um conjunto de regras merece outra análise.

Para o caso do conjunto de regras, há duas maneiras de se inferir: inferência baseada em composição e inferência baseada em regra individual.

Na inferência baseada em composição, combinam-se todas as regras da base de regras fuzzy em uma única relação $U \times V$, que passa a ser considerada uma única regra fuzzy SE-ENTÃO. Esta combinação é feita através de operadores lógicos apropriados que são aplicados de acordo com a interpretação intuitiva que é dada ao conjunto de regras. Essa interpretação se dá com base em dois argumentos opostos, o primeiro considera as regras como declarações condicionais independentes e, sob este ponto de vista, o operador razoável para a combinação é a união. O segundo argumento considera as regras como declarações condicionais fortemente acopladas de maneira que, todas as condições destas regras devam ser satisfeitas afim de que, o conjunto de regras, como um todo, tenha impacto. Sob este ponto de vista, deve-se usar o operador de interseção para combinar as regras. São apresentados, em seguida, os detalhes destes dois esquemas.

Seja uma relação $Ru^{(l)}$ uma relação fuzzy em $U \times V$ que representa a regra fuzzy SE-ENTÃO na forma da Eq.(2.80), ou seja, $Ru^{(l)} = A_1^l \times A_2^l \times ... \times A_n^l \rightarrow B^l$, sendo que, $A_1^l \times A_2^l \times ... \times A_n^l$ é uma relação fuzzy em $U = U_1 \times U_2 \times ... \times U_n$ definida por:

$$\mu_{A_1^l \times ... \times A_n^l}(x_1, ..., x_n) = \mu_{A_1^l}(x_1) \star ... \star \mu_{A_n^l}(x_n) \tag{2.86}$$

onde "$\star$" representa qualquer operador norma-t e a implicação $\rightarrow$ em $Ru^{(l)}$ é definida de acordo com as implicações apresentadas na Tabela 8. Se for relevante o primeiro ponto de vista (regras independentes), então as M regras na forma da Eq.(2.80) são entendidas como uma única relação fuzzy Q_M em $U \times V$ conforme:

$$Q_M = \bigcup_{l=1}^{M} Ru^{(l)} \tag{2.87}$$

Esta combinação é chamada de combinação Mamdani, que também pode ser escrita na forma:

$$\mu_{Q_M}(x,y) = \mu_{Ru^{(1)}}(x,y) \dot{+} \ldots \dot{+} \mu_{Ru^{(M)}}(x,y) \tag{2.88}$$

onde "$\dot{+}$" representa a norma-s.

Se for considerado o segundo ponto de vista (regras acopladas), as M regras fuzzy SE-ENTÃO na forma da Eq.(2.80) são entendidas como uma relação fuzzy Q_G em $U \times V$ e definida como segue:

$$Q_G = \bigcap_{l=1}^{M} Ru^{(l)} \tag{2.89}$$

Esta combinação é chamada de combinação Gödel, e também pode ser escrita na seguinte forma:

$$\mu_{Q_G}(x,y) = \mu_{Ru^{(1)}}(x,y) \star \ldots \star \mu_{Ru^{(M)}}(x,y) \tag{2.90}$$

onde "$\star$" significa norma-t.

Considerando que na entrada de uma máquina de inferência fuzzy haja um conjunto fuzzy arbitrário A' em U. E considerando Q_G e Q_M como uma única regra fuzzy SE-ENTÃO e, fazendo uso do Modus Ponens generalizado (2.77), pode-se obter a saída fuzzy da máquina de inferência, usando a combinação Mamdani, da seguinte forma:

$$\mu_{B'}(y) = sup_{x \in U}\, t[\mu_{A'}(x), \mu_M(x,y)] \tag{2.91}$$

ou ainda, usando a combinação Gödel:

$$\mu_{B'}(y) = sup_{x \in U}\, t[\mu_{A'}(x), \mu_G(x,y)] \tag{2.92}$$

Com base nesta análise, um procedimento computacional é formulado para a obtenção da inferência baseada em combinação, portanto tem-se:

Passo 1: Para as M regras fuzzy SE-ENTÃO da Eq.(2.80), determinar as funções de pertinência $\mu_{A_1^l \times \ldots \times A_n^l}(x_1, \ldots, x_n)$ para $l = 1,2, \ldots, M$, conforme a Eq.(2.86).

Passo 2: Considerar $A_1^l \times ... \times A_n^l = FP_1$ e $B^l = FP_2$ ao usar as implicações mostradas na Tabela 8 e determinar $\mu_{Ru^{(l)}}(x_1, ..., x_n, y) = \mu_{A_1^l \times ... \times A_n^l \to B^l}(x_1, ..., x_n, y)$ para $l = 1,2, ..., M$ através de qualquer uma destas implicações.

Passo 3: Determinar $\mu_{Q_M}(x, y)$ conforme a Eq.(2.88) ou $\mu_{Q_G}(x, y)$ conforme a Eq.(2.90).

Passo 4: Para uma dada entrada fuzzy A', a máquina de inferência fuzzy fornece uma saída fuzzy B' conforme as Eqs.(2.91) ou (2.92).

Na inferência baseada em regra individual, um conjunto fuzzy de saída é determinado para cada regra fuzzy da base, rendendo um total de M conjuntos fuzzy. A saída total da máquina de inferência fuzzy é a combinação destes M conjuntos fuzzy individuais. Esta combinação pode ser feita tanto por união como por interseção.

Para a inferência baseada em regra individual, o procedimento computacional é o seguinte:

Passos 1 e 2: Exatamente os mesmos para a inferência baseada em composição.

Passo 3: Um conjunto fuzzy de saída B'_l em V é calculado para cada regra individual $Ru^{(l)}$, de acordo com o Modus Ponens generalizado, considerando na entrada da máquina de inferência fuzzy um conjunto fuzzy A' em U. Logo:

$$\mu_{B'_l}(y) = sup_{x \in U}\, t[\mu_{A'}(x), \mu_{Ru^{(l)}}(x, y)] \tag{2.93}$$

para $l = 1,2, ..., M$.

Passo 4: Combinar os M conjuntos fuzzy $\{B'_1, ..., B'_M\}$:

Por união, a combinação é:

$$\mu_{B'}(y) = \mu_{B'_1}(y) \dot{+} ... \dot{+} \mu_{B'_M}(y) \tag{2.94}$$

E por interseção, a combinação é:

$$\mu_{B'}(y) = \mu_{B'_1}(y) \star ... \star \mu_{B'_M}(y) \tag{2.95}$$

Onde $\dot{+}$ e $\star$ representam operadores de norma-s e norma-t respectivamente.

2.2.12.4- Características de Algumas Máquinas de Inferência Fuzzy

Devido às várias alternativas já mostradas, é possível de se ver que uma máquina de inferência pode ser montada de diversas maneiras.

Partindo das regras da base de regras fuzzy, podem-se ter cinco tipos de implicações diferentes (Tabela 8). Para a sua combinação tem-se a combinação Mamdani e a combinação Gödel, além do que, estes processos envolvem operações de normas-t e normas-s, que também possuem uma variedade de propostas de implementação. Diante desta diversidade de recursos, cabe perguntar sobre como proceder para fazer a sua seleção?

Devem ser considerados alguns critérios importantes. O apelo intuitivo é um dos critérios. Nele a escolha deve fazer sentido sob um ponto de vista também intuitivo. A eficiência computacional é outro critério. Neste caso, a escolha deve resultar numa fórmula que, ao relacionar A' com B', facilite os cálculos e, o critério das propriedades especiais, onde as características que se apresentam como mais importantes na escolha devem prevalecer.

Algumas máquinas de inferência, propostas na literatura [6], são apresentadas na seqüência.

- **Máquina de Inferência Produto**:

  - Características:
    (1) Inferência baseada em regra individual com combinação por união.
    (2) Implicação produto Mamdani.
    (3) Normas-t: Produto Algébrico e normas-s: máximo.

Usando as Eqs.(2.93),(2.94),(2.67) e (2.86) obtém-se:

$$\mu_{B'}(y) = max_{l=1}^{M}[sup_{x\in U}(\mu_{A'}(x)\prod_{i=1}^{n}\mu_{A_i^l}(x_i)\mu_{B^l}(y))] \qquad 2.96)$$

Para a Eq.(2.96) é dado um conjunto fuzzy A' em U e é obtido um conjunto fuzzy B' em V.

- **Máquina de Inferência Mínimo**:

  - Características:
    (1) Inferência baseada em regra individual com combinação por união.

(2) Implicação mínimo Mamdani.

(3) Normas-t: mínimo e norma-s: máximo.

Usando as Eqs.(2.93),(2.94),(2.66) e (2.86) obtem-se:

$$\mu_{B'}(y) = max_{l=1}^{M}[sup_{x\in U} min(\mu_{A'}(x), \mu_{A_1^l}(x_1), \dots, \mu_{A_n^l}(x_n), \mu_{B^l}(y))] \quad (2.97)$$

Para a Eq.(2.97) é dado um conjunto fuzzy A' em U e é obtido um conjunto fuzzy B' em V.

- **Máquina de Inferência Lukasiewicz**:

  - Características:

    (1) Inferência baseada em regra individual com combinação por interseção.

    (2) Implicação Lukasiewicz.

    (3) Normas-t: mínimo.

Usando as Eqs.(2.95),(2.93),(2.63) e (2.86) obtem-se:

$$\mu_{B'}(y) = min_{l=1}^{M}\left\{sup_{x\in U} min\left[\mu_{A'}(x), 1 - min_{i=1}^{n}(\mu_{A_i^l}(x_i)) + \mu_{B'}(y)\right]\right\} \quad (2.98)$$

Para a Eq.(2.98) é dado um conjunto fuzzy A' em U e é obtido um conjunto fuzzy B' em V.

- **Máquina de Inferência Zadeh**:

  - Características:

    (1) Inferência baseada em regra individual com combinação por interseção.

    (2) Implicação Zadeh.

    (3) Normas-t: mínimo.

Usando as Eqs.(2.95),(2.93),(2.64) e (2.86) obtem-se:

$$\mu_{B'}(y) = min_{l=1}^{M}\{sup_{x\in U} \min\left[\mu_{A'}(x), \max(\min(\mu_{A_1^l}(x_1), \dots, \mu_{A_n^l}(x_n), \mu_{B^l}(y)), 1 - \min_{i=1}^{n}(\mu_{A_i^l}(x_i)))\right]\} \quad (2.99)$$

Para a Eq.(2.99) é dado um conjunto fuzzy A' em U e é obtido um conjunto fuzzy B' em V.

- **Máquina de Inferência Dienes-Rescher**:

  - Características:
    (1) Inferência baseada em regra individual com combinação por interseção.
    (2) Implicação Dienes-Rescher.
    (3) Normas-t: mínimo.

Usando as Eqs.(2.95),(2.93),(2.62) e (2.86) obtem-se:

$$\mu_{B'}(y) = min_{l=1}^{M}\{sup_{x\in U}\min\,[\mu_{A'}(x), max\,(1 - min_{\mathrm{i}=1}^{\mathrm{n}}(\mu_{A_i^l}(x_i)), \mu_{B^l}(y))]\} \qquad (2.100)$$

Para a Eq.(2.100) é dado um conjunto fuzzy A' em U e é obtido um conjunto fuzzy B' em V.

Por apresentarem simplicidade computacional a máquina de inferência produto e a máquina de inferência mínimo são muito comuns em aplicações de sistemas e controle fuzzy, porém, se a aplicação envolve casos onde as funções de pertinência $\mu_{A_i^l}(x_i)$ para algum $x \in U$ seja pequena, são fornecidos $\mu_{B'}(y)$ pequenos resultando em problemas na implementação.

As máquinas de inferência Lukasiewicz, Zadeh e Dienes-Rescher foram propostas para resolver estas dificuldades.

2.2.12.5- Fuzificadores e Defuzificadores

O fato das máquinas de inferência fuzzy combinarem regras da base de regras fuzzy, mapeando os conjuntos fuzzy A' em U para conjuntos fuzzy B' em V, restringe o seu uso direto em problemas reais, os quais, na sua maioria, tratam-se de sistemas de numeração real, o que não poderia servir como entrada direta em um sistema de inferência fuzzy. Em função desta restrição e, para que se possa explorar a aplicação dos sistemas fuzzy nos sistemas reais, é necessária a construção de interfaces entre este ambiente externo e a máquina de inferência fuzzy, tais interfaces são denominadas fuzificadores e defuzificadores.

- **Fuzificadores**: O fuzificador é definido como um mapeamento de um ponto real $x^* \in U \subset R^n$ para um conjunto fuzzy A' em U.

Para que seja projetado um fuzificador, alguns critérios devem ser obedecidos.

(1) O fuzificador deve considerar que a entrada é um ponto crisp x^*, significando que o conjunto fuzzy A' deva ter alto valor de pertinência em x^*.

(2) O fuzificador deve ajudar a suprimir ruído caso este venha a corromper sua entrada.

(3) O fuzificador deve ajudar a simplificar os cálculos relacionados com a máquina de inferência fuzzy. Como exemplo, simplificar os cálculos envolvendo $sup_{x \in U}$.

Na seqüência apresentam-se três fuzificadores, o singleton, o Gaussiano e o triangular.

- Fuzificador singleton: Mapeia o ponto $x^* \in U \subset R^n$, de valor real, para um singleton fuzzy A' em U, representado por:

$$\mu_{A'}(x) = \begin{cases} 1 & se \ \ x = x^* \\ 0 & outros \ valores \end{cases} \tag{2.101}$$

- Fuzificador Gaussiano: Mapeia o ponto $x^* \in U \subset R^n$, de valor real, para conjunto fuzzy A' em U. Para este mapeamento é necessário uma função de pertinência Gaussiana, conforme:

$$\mu_{A'}(x) = e^{-(\frac{x_1 - x_1^*}{a_1})^2} \star \ldots \star e^{-(\frac{x_n - x_n^*}{a_n})^2} \tag{2.102}$$

onde "$\star$" é o operador norma-t, usualmente utiliza-se o produto algébrico ou o mínimo e os parâmetros a_i são positivos.

- Fuzificador Triangular: Mapeia o ponto $x^* \in U \subset R^n$, de valor real, para conjunto fuzzy A' em U. Neste caso é utilizada uma função de pertinência triangular, conforme:

$$\mu_{A'}(x) = \begin{cases} \left(1 - \dfrac{|x_1 - x_1^*|}{b_1}\right) \star \ldots \star \left(1 - \dfrac{|x_n - x_n^*|}{b_n}\right) & se \ \ |x_i - x_i^*| \leq b_i \, , i = 1,2, \ldots n \\ 0 & outros \ valores \end{cases} \tag{2.103}$$

onde "$\star$" é o operador norma-t, usualmente utiliza-se o produto algébrico ou o mínimo e os parâmetros b_i são positivos.

Algumas considerações relevantes são feitas acerca dos fuzificadores apresentados, as quais auxiliam na sua seleção.

- O fuzificador singleton promove uma simplificação significativa nos cálculos envolvidos na máquina de inferência fuzzy para quaisquer tipos de funções de pertinência que as regras fuzzy SE-ENTÃO posam assumir.

- O fuzificador Gaussiano e o triangular, também simplificam os cálculos na máquina de inferência fuzzy, se as funções de pertinência nas regras fuzy SE-ENTÃO forem Gaussianas ou triangulares respectivamente.

- O fuzificador singleton não pode suprimir ruído na entrada, já os fuzificadores Gaussiano e triangular podem [6].

Defuzificadores: O defuzificador é definido como um mapeamento de um conjunto fuzzy B' em $V \subset R$, tomado a partir da saída de uma máquina de inferência fuzzy, para um ponto crisp $y^* \in V$. É responsável por determinar um ponto em V que melhor represente o conjunto fuzzy B'. Isso é similar à média de uma variável aleatória. Conforme foi mostrado na discussão sobre a máquina de inferência, B' é construído através de algumas técnicas específicas, portanto há uma quantidade de opções para se determinar o ponto que represente B'. Por isso, a escolha do defuzificador deve ser orientada com base nos três critérios seguintes:

(1) Plausibilidade: O valor de saída crisp, ou seja, o ponto $y^* \in V$ deve representar B' em V sob um ponto de vista intuitivo, como por exemplo, pode estar próximo ao meio do suporte, ou ainda ter um alto grau de pertinência em B'.

(2) Simplicidade computacional: É um critério de grande importância para a área de controle fuzzy, uma vez qua o custo computacional está diretamente associado ao desempenho dos sistemas.

(3) Continuidade: Relacionada com mudanças abruptas de valores. Uma pequena variação em B' não pode provocar uma grande mudança em y^*.

Relacionam-se os três tipos de defuzificadores mais comumente utilizados: o defuzificador de Centro de Gravidade (COG), o defuzificador de Centro Ponderado (CAv) e o defuzificador de Máximo Valor.

Defuzificador de Centro de Gravidade (COG): O defuzificador COG especifica o y^* como o centro da área coberta pela função de pertinência B', conforme definido a seguir:

$$y^* = \frac{\int_V y\, \mu_{B'}(y) dy}{\int_V \mu_{B'}(y) dy} \tag{2.104}$$

onde $\int_v$ é a integral convencional. Esta operação é mostrada através da Fig.16.

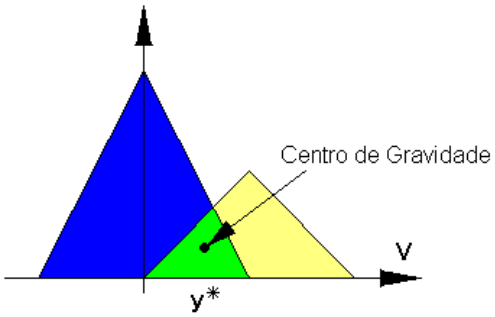


Figura 16- Representação do defuzificador de Centro de Gravidade.

Defuzificador de Centro Ponderado (CAv): Sendo o conjunto fuzzy B' a união ou interseção de M conjuntos fuzzy, a ponderação dos centros desses M conjuntos fuzzy produz uma boa aproximação para o centro de gravidade Eq.(8.15). Usam-se como pesos as alturas dos conjuntos fuzzy correspondentes. Calcula-se y* através de:

$$y^* = \frac{\sum_{l=1}^{M} \bar{y}^l w_l}{\sum_{l=1}^{M} w_l} \tag{2.105}$$

onde $\bar{y}^l$ é o centro do $l-$ésimo conjunto fuzzy e w_l é a respectiva altura.
Esta operação é ilustrada graficamente pela Fig.17.

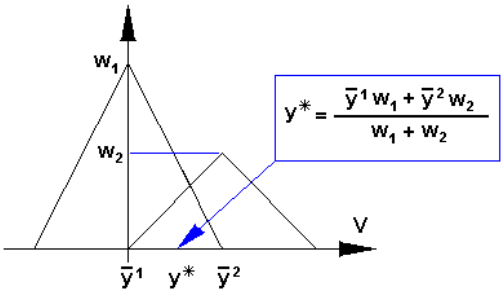


Figura 17- Representação do defuzificador de Centro Ponderado.

Defuzificador de Máximo Valor: A idéia sob a qual o defuzificador de máximo valor é concebido consiste em escolher o y^*, que é um ponto em V, no qual a função de pertinência $\mu_{B'}(y)$ atinge o seu valor máximo. Quando definido o conjunto:

$$hgt(B') = \{y \in V | \mu_{B'}(y) = sup_{y \in V} \mu_{B'}(y)\} \tag{2.106}$$

onde $hgt(B')$ é o conjunto de todos os pontos em V nos quais $\mu_{B'}(y)$ alcança seu valor máximo. Então o defuzificador de máximo valor define o y^* de forma arbitrária dentre os elementos em $hgt(B')$, significando que:

$$y^* = qualquer\ ponto\ em\ hgt(B') \tag{2.107}$$

Se houver apenas um ponto em $hgt(B')$, somente este ponto definirá y^*, caso $hgt(B')$ contenha mais de um ponto, pode-se definir y^* através de um dos seguintes critérios:

(1) Menor dos máximos:

$$y^* = inf\{y \in hgt(B')\} \tag{2.108}$$

(2) Maior dos máximos:

$$y^* = sup\{y \in hgt(B')\} \tag{2.109}$$

(3) Média dos máximos:

$$y^* = \frac{\int_{hgt(B')} y\, dy}{\int_{hgt(B')} dy} \tag{2.110}$$

Os três defuzificadores apresentados, possuem características que os tornam diferentes entre si. Estas características, confrontadas com os critérios apresentados para a escolha do defuzificador se resumem da seguinte forma:

- O defuzificador de Centro de Gravidade é plausível, e apresenta uma carga computacional elevada.

- O defuzificador de centro Ponderado é o mais comum em aplicações de sistemas fuzzy e controle, pois ele atende aos três critérios de escolha de um bom defuzificador.

- O defuzificador de Máximo valor é plausível, apresenta simplicidade computacional, porém pequenas variações em B' podem render grandes variações em y^*.

2.2.13- Conclusão

Este capítulo aborda os conceitos relacionados à lógica fuzzy e aos sistemas fuzzy.

Verificou-se que, a partir da lógica clássica e através da generalização dos seus conceitos, é possível definir os conceitos empregados na lógica fuzzy. Foram definidos os conjuntos fuzzy e as operações básicas a eles relacionadas: complemento, união, interseção e também outras operações adicionais como normas-s e normas-t aumentando-se assim seus recursos teóricos e conseqüentemente proporcionando uma melhor abrangência desta teoria no seu campo de atuação. Foram definidos os conceitos de variável lingüística e de base de regras fuzzy e também foram introduzidos os principais elementos componentes dos sistemas fuzzy: a máquina de inferência fuzzy, os fuzificadores e os defuzificadores.

3 – A PROGRAMAÇÃO ORIENTADA A OBJETOS APLICADA À CONSTRUÇÃO DE CONTROLADORES FUZZY

3.1 – INTRODUÇÃO

Através da análise orientada a objetos (AOO) é possível o desenvolvimento de classes que representem, por intermédio dos seus objetos, cada um dos blocos usados na construção de um controlador fuzzy. A idéia do uso de tal técnica tem como vantagem, em primeiro lugar, permitir a criação de uma biblioteca de classes genéricas que facilitem o projeto destes controladores, reduzindo o seu código, facilitando também a sua manutenção e, em segundo lugar, a possibilidade da implementação simultânea (em um único programa) de diversos controladores especializados para controlarem um determinado sistema.

Serão apresentadas, na seqüência, as definições das classes que criam objetos para os conjuntos fuzzy, variáveis lingüísticas e fuzificadores, defuzificadores, máquina de inferência, bem como a classe dos objetos controladores como sendo uma integração dos objetos anteriores.

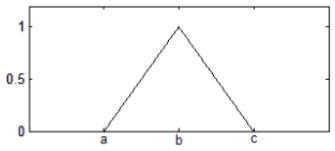
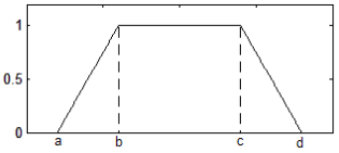
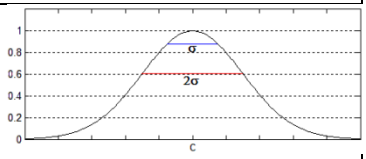
3.2 – REPRESENTAÇÕES DE CONJUNTOS FUZZY, VARIÁVEIS LINGUÍSTICAS E FUZIFICADORES

Nesta abordagem, objetivando definir uma classe que crie os objetos que irão atuar como variáveis lingüísticas e fuzificadores é preciso que se parta de uma classe que servirá de base a esta: a classe dos conjuntos fuzzy, a qual define o tipo de conjunto fuzzy, bem como os seus parâmetros, para cada valor lingüístico que a variável lingüística possa assumir. Na Tabela 11, apresentam-se alguns tipos de conjuntos fuzzy, comumente utilizados em sistemas fuzzy para a definição das funções de pertinência [8].

A especificação da classe dos conjuntos fuzzy, denominada CjFuzzy, é apresentada na Fig.18, onde são apresentados os seus atributos e métodos.

Os objetos da classe CjFuzzy são utilizados como atributos da classe das variáveis lingüísticas, ou seja, neste caso é aplicado o conceito de herança[14], onde a classe das variáveis lingüísticas é classe derivada de CjFuzzy.

Os objetos da classe CjFuzzy armazenam os parâmetros que definem cada conjunto fuzzy que está definido dentro do universo de discurso da variável lingüística associada, em outras palavras, representam os valores lingüísticos que a referida variável lingüística possa assumir.

Tabela 11 – Alguns tipos de conjuntos fuzzy freqüentemente utilizados			
Tipo	Parâmetros	Definição de μ_A	Gráfico
Triangular	[a b c]	$\mu_A(x: a,b,c) = \begin{cases} 0 & ,para\ x < a \\ \frac{x-a}{b-a} & ,para\ a \le x < b \\ \frac{c-x}{c-b} & ,para\ b \le x \le c \\ 0 & ,para\ x > c \end{cases}$	
Trapezoidal	[a b c d]	$\mu_A(x: a,b,c,d) = \begin{cases} 0 & ,para\ x < a \\ \frac{x-a}{b-a} & ,para\ a \le x < b \\ 1 & ,para\ b \le x < c \\ \frac{d-x}{d-c} & ,para\ c \le x < d \\ 0 & ,para\ x > d \end{cases}$	
Gaussiana	[σ c]	$\mu_A(x: \sigma, c) = e^{-\frac{(x-c)^2}{\sigma}}$	

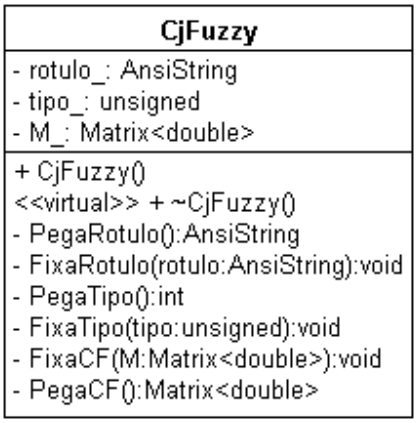


Figura 18 – Classe CjFuzzy

Cada um dos valores lingüísticos é identificado no atributo rotulo_.

O atributo tipo_ armazena um número inteiro, o qual define se a função de pertinência é triangular (tipo_ = 3), trapezoidal (tipo_ = 4) ou gaussiana (tipo_ = 2). Na verdade este valor representa o número de elementos contidos no vetor de parâmetros que define o conjunto fuzzy, conforme está mostrado na Tabela 11.

O atributo M_ deve conter o próprio vetor de parâmetros passado pelo criador da classe CjFuzzy.

O objetivo de se definir a classe CjFuzzy desta maneira é subsidiar o fuzificador, que faz uso das definições de μ_A da Tabela 11 e em breve será discutido.

Além do construtor da classe e do seu destrutor (CjFuzzy( ) e ~CjFuzzy( )) outros métodos são implementados para agir como interface: PegaRotulo( ), que retorna o valor liguístico e FixaRotulo(rotulo), que registra no atributo rotulo_ o valor lingüístico especificado pelo usuário, ocorrendo o mesmo com os métodos que fazem referência ao atributo tipo_, bem como ao atributo M_.

Para que sejam criados objetos que representem as variáveis linguísticas de um sistema fuzzy, é mostrado na Fig.19 a especificação da classe, que é denominada de VarLing.

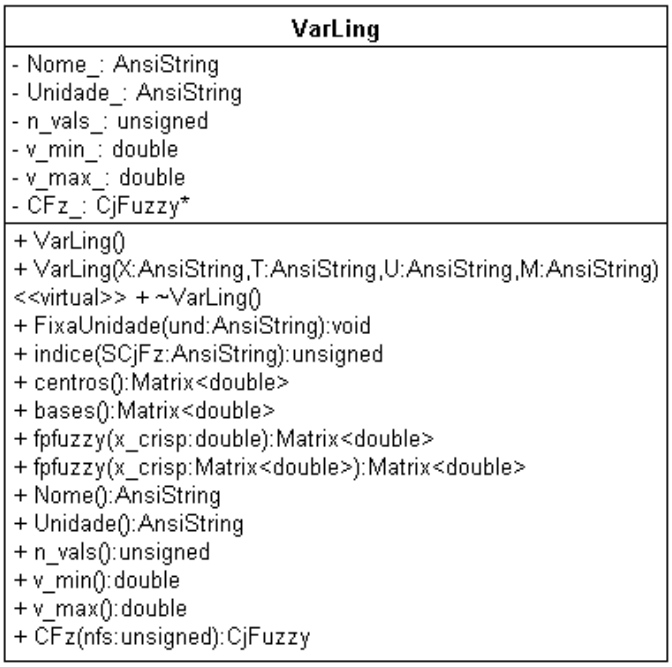


Figura 19 – Classe VarLing

O objetivo principal desta classe é tornar mais simples a programação das variáveis lingüísticas, no momento das suas definições e a partir das suas especificações, atendendo as condições exigidas pelo controlador fuzzy que está sendo implementado. Esta simplificação se resume em criar objetos que sejam variáveis lingüísticas com linhas de comando com a seguinte sintaxe:

VarLing NomeDaVarLing(**X** , **T**, **U** , **M**) (3.1)

onde **X** é uma variável do tipo "string" contendo o nome da variável lingüística, **T** é um vetor contendo as "strings" dos valores lingüísticos (rótulos) que a variável lingüística pode assumir, **U** é um vetor, que representa o domínio onde a variável X possui seus valores crisp, e deve conter dois elementos que são o limite inferior e o limite superior respectivamente

deste domínio. O parâmetro **M** constitui uma regra semântica que relaciona cada valor lingüístico de **T** com cada conjunto fuzzy em **U** [3].

A classe VarLing é derivada da classe CjFuzzy e possui uma propriedade denominada *CFz_ que é um ponteiro para um "array" de objetos da classe CjFuzzy. Cada objeto deste "array" representa um dos conjuntos fuzzy que compõem as funções de pertinência da variável lingüística associada, sendo atribuído um rótulo a cada um desses conjuntos fuzzy, portanto, um objeto da classe VarLing representa uma determinada variável lingüística que contém todos os conjuntos fuzzy que lhe foram definidos.

Uma vez declaradas as classes CjFuzzy e VarLing, é possível a criação dos objetos que representam as variáveis lingüísticas do sistema de controle fuzzy em projeto. Como exemplo, considere as especificações da variável lingüística E (Erro), mostrada na Fig.20.

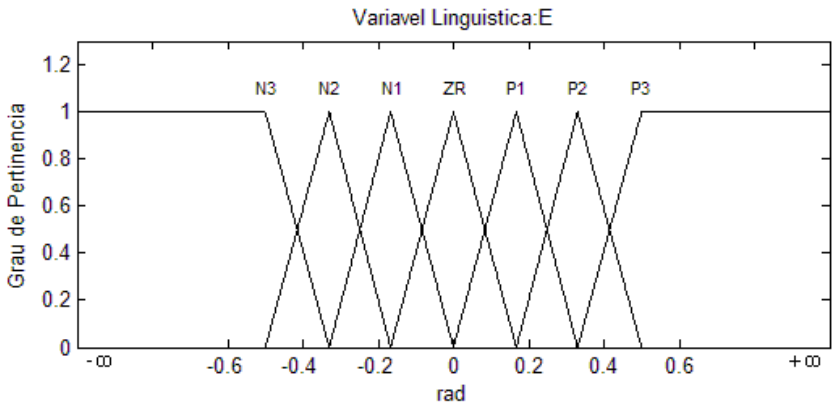


Figura 20 – Exemplo de especificação de Variável Linguística (Erro)

Para que se possa fazer uso da classe VarLing através de comando cuja sintaxe seja a da forma em (3.1) é necessário que se defina, com base no que é especificado na Fig.20, os parâmetros **X**, **T**, **U** e **M**. A Tabela 12 apresenta estas definições .

Os valores de **X**, **T**, **U** e **M** fornecem as informações necessárias para os atributos dos objetos que representam cada variável lingüística, servindo de parâmetros para o funcionamento do fuzificador que, para o modelo de programação que é apresentado neste trabalho, é implementado como um método da classe VarLing e não como um objeto isolado pertencente a uma classe específica. Sua denominação como método é fpfuzzy (.), o qual recebe como parâmetro o valor crisp, oriundo do sinal de entrada da variável lingüística associada. Este método retorna um vetor com a dimensão de **T**, onde cada um dos seus elementos apresenta o grau de pertinência do valor crisp de entrada em relação a cada um dos

conjuntos fuzzy que estão definidos no universo de discurso da referida variável lingüística. Este vetor, que é fornecido pelo método fpfuzzy, é o que torna possível a operação da principal etapa da máquina de inferência: o "loop" de programa que testa todas as regras de base, para determinar aquelas que são ativadas.

Tabela 12 – Definição dos parâmetros para o criador da classe VarLing

Parâmetro	Valor
X	"E"
T	{'N3','N2','N1','ZR','P1','P2','P3'}
U	[-10, 10]
M	{[-10.1, -10, -0.5, -0.33], [-0.5, -0.33, -0.166], [-0.33, -0.166, 0], [-0.166, 0, 0.166], [0, 0.166, 0.33], [0.166, 0.33, 0.5], [0.33, 0.5, 10, 10.1]}

Outros métodos importantes da classe VarLing e também importantes ao processo de programação e ao processo de operação dos controladores fuzzy desenvolvidos sob esta metodologia, são o gerador de índices, chamado por **índice(.)**, que recebe a string de um rótulo e fornece um valor inteiro que vai de 0 até (**n_vals_**)-1 e serve para gerar os valores que compõem a tabela de índices que, de fato, é a forma na qual este tipo de programação usa para representar a Base de Regras do sistema. Os métodos **centros(.)** e **bases(.)**, que calculam a posição dos centros e o tamanho das bases, respectivamente, de cada conjunto fuzzy definido para a variável lingüística e retornam um vetor de centros e um vetor de bases, cada qual com dimensão de **T**.

3.3 – REPRESENTAÇÃO DA MÁQUINA DE INFERÊNCIA

Para representar a máquina de inferência, é criada uma estrutura, chamada de **MInferencia**, que é vista na Fig.21. A principal função com a criação de um objeto da classe **MInferencia** é a criação de duas tabelas correlacionadas, a **indxEntr** que reúne os índices das variáveis lingüísticas de entrada e a **indxSai**, que reúne os índices para as variável lingüística de saída. A técnica de construção destas tabelas é mostrada na Seção 3.4.

As tabelas de índices são construídas a partir da base de regras fuzzy, que é fornecida pelo programador em formato matricial. São as tabelas de índices que servem de referência para a coordenação do sistema de inferência fuzzy. Em outras palavras, essas tabelas de índices constituem um mapa em memória do qual o sistema de inferência tira as conclusões com base nas regras ativadas. Essas tabelas são a síntese de todas as combinações possíveis dos valores lingüísticos entre as variáveis lingüísticas de entrada associadas aos valores que a saída fuzzy deve apresentar em cada uma das combinações. Seus índices são utilizados para referenciar os elementos dos vetores, que saem dos defuzificadores portando os graus de pertinência para cada valor lingüístico de cada variável lingüística correspondente. Esses graus de pertinência serão analizados pelo sistema de inferência, que é parte de um objeto controlador, que será em breve discutido. A análise dos graus de pertinência, para determinar o grau de ativação das regras de base de regras fuzzy, é feita dentro de um "loop" de programa, o qual é executado desde a primeira regra até a última, onde os índices são utilizados. Esta é uma das vantagens desta técnica de programação, pois o programador não precisa escrever linhas de comando contendo "Ifs" e "Thens" tantas vezes quanto sejam o número de regras existentes em uma base de regras fuzzy.

Com a criação de um objeto da classe **MInferencia**, também são calculados os centros e as bases dos conjuntos fuzzy que são definidos para a variável lingüística da saída. Os centros e as bases são usados no processo de defuzificação.

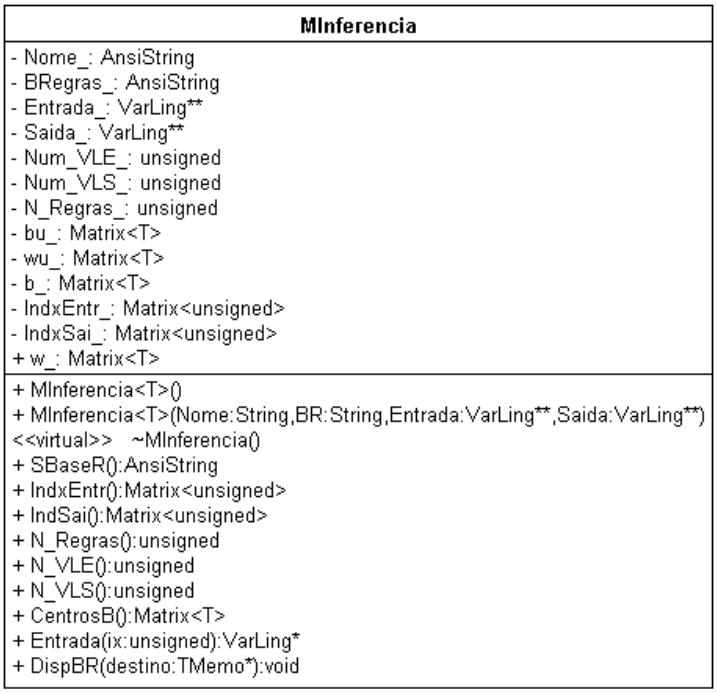


Figura 21 – Classe MInferencia

3.4 – TÉCNICA DE INDEXAÇÃO PARA A BASE DE REGRAS

Para poder explicar a técnica utilizada pela classe **MInferencia** para gerar a indexação da base de regras fuzzy, considere um sistema de inferência fuzzy com duas variáveis lingüísticas de entrada x_1 , x_2 e uma de saída u, onde cada uma das três variáveis pode assumir um dos três valores lingüísticos {“N”,”Z”,”P”}, ou seja, combinando as duas variáveis de entrada obtem-se nove regras. Com base neste fato, uma base de regras fuzzy plausível seria, conforme a Tabela 13.

Tabela 13 – Exemplo de Base de Regras fuzzy.

Regra	
1	$SE\ x_1$ é $N\ \ e\ x_2$ é $N, ENTÃO\ u$ é P
2	$SE\ x_1$ é $Z\ \ e\ x_2$ é $N, ENTÃO\ u$ é P
3	$SE\ x_1$ é $P\ \ e\ x_2$ é $N, ENTÃO\ u$ é Z
4	$SE\ x_1$ é $N\ \ e\ x_2$ é $Z, ENTÃO\ u$ é P
5	$SE\ x_1$ é $Z\ \ e\ x_2$ é $Z, ENTÃO\ u$ é Z
6	$SE\ x_1$ é $P\ \ e\ x_2$ é $Z, ENTÃO\ u$ é N
7	$SE\ x_1$ é $N\ \ e\ x_2$ é $P, ENTÃO\ u$ é Z
8	$SE\ x_1$ é $Z\ \ e\ x_2$ é $P, ENTÃO\ u$ é N
9	$SE\ x_1$ é $P\ \ e\ x_2$ é $P, ENTÃO\ u$ é N

Para a programação, é necessário que se forneça a Tabela 13 conforme é mostrado na Tabela 14.

Tabela 14 – Outra representação de base de regras fuzzy para a Tabela 13.

		x_1		
		N	Z	P
x_2	N	P	P	Z
	Z	P	Z	N
	P	Z	N	N

A Tabela 14 tem os seus dados convertidos para valores numéricos que serão considerados índices, conforme mostrado na Tabela 15.

Tabela 15 – Conversão de índices para a Tabela 14.

		x_1		
		1	2	3
x_2	1	3	3	2
	2	3	2	1
	3	2	1	1

Todas as combinações possíveis entre as variáveis de entrada são geradas e as informações da Tabela 15 são organizadas conforme a Tabela 16. As colunas x_1 e x_2 são atribuídas à tabela **indxEntr** e a coluna u é atribuída à tabela **indxSai**, citadas na Seção 3.3.

Tabela 16- Geração da tabela de índices.

Regra	x_1	x_2	u
1	1	1	3
2	2	1	3
3	3	1	2
4	1	2	3
5	2	2	2
6	3	2	1
7	1	3	2
8	2	3	1
9	3	3	1

Desta forma, as tabelas **indxEntr** e **indxSai** estão prontas para serem usadas com os métodos **InferSysCAv** ou **inferSysCOG** da classe **Controlador**, apresentado na Seção 3.6.

3.5 – REPRESENTAÇÕES DOS DEFUZIFICADORES

Para a criação de objetos que representem defuzificadores em um determinado sistema fuzzy, é necessário a definição de classes especializadas, de acordo com o método de defuzificação que é implementado no sistema, conforme apresentado nas Fig. 22 e 23. Tem-se então as classes **COGravity** e **CAverage** que permitem a criação de objetos para esta finalidade. **COGravity** é a classe da qual se constroem objetos defuzificadores que implementam a defuzificação pelo centro de gravidade dos conjuntos fuzzy, e **CAverage** pelo centro ponderado dos conjuntos fuzzy.

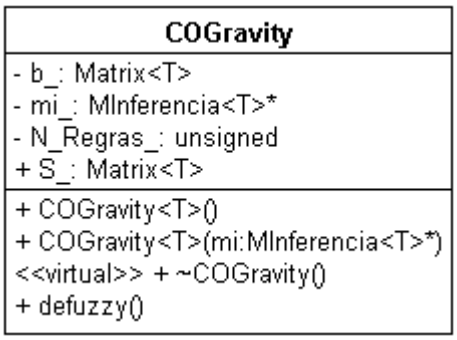


Figura 22 – Classe para defuzificador do tipo Centro de Gravidade.

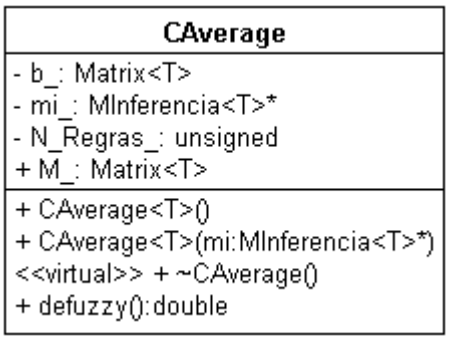


Figura 23 – Classe para defuzificador do tipo Centro Ponderado.

Apesar das duas classes apresentarem o mesmo número de propriedades, há algumas diferenças entre as duas: a propriedade M_ e a propriedade S_, onde M_ é um vetor com dimensão do número de regras, cujos elementos servem de repositório para o grau de ativação de cada regra e, S_ é um vetor, também com dimensão do número de regras, servindo de repositório para a área de cada conjunto fuzzy em cujas regras foram ativadas. As propriedades M_ e S_ são atualizadas pelos métodos **InferSysCAv** e **InferSysCOG**,

respectivamente, que são pertencentes à classe **Controlador** que será vista neste capítulo. A propriedade b_, que é um vetor com a dimensão de **T** da variável lingüística da saída, contendo as posições dos seus centros. As outras diferenças são referentes aos seus métodos, que são específicos a cada uma dessas classes: os dois métodos construtores, o método destrutor e o método principal **defuzzy(.)**, que é o defuzificador propriamente dito.

Para que se possa criar um objeto defuzificador, é necessário que se tenha um objeto de máquina de inferência criado, pois os defuzificadores, aqui desenvolvidos, dependem de parâmetros da máquina de inferência. A classe **MInferência** permite com que sejam disponibilizados os dois tipos de defuzificadores simultaneamente de forma que, seja possível de se optar pelo uso de um dos dois defizificadores, dependendo da necessidade.

Para obter um valor crisp na saída de um desses defuzificadores, primeiro é preciso acionar o sistema de inferência da máquina de inferência para depois chamar o método **defuzzy**( ).

3.6 – REPRESENTAÇÃO DO CONTROLADOR FUZZY

Pela programação orientada a objetos é possível representar controladores fuzzy por intermédio dos objetos criados a partir da classe **Controlador**, apresentada na Fig. 24.

Analisando os seus métodos, pode-se concluir que, para se criar um controlador fuzzy é preciso que se tenha um objeto máquina de inferência e um ou dois objetos defuzificadores (neste caso, um CAv e outro COG).

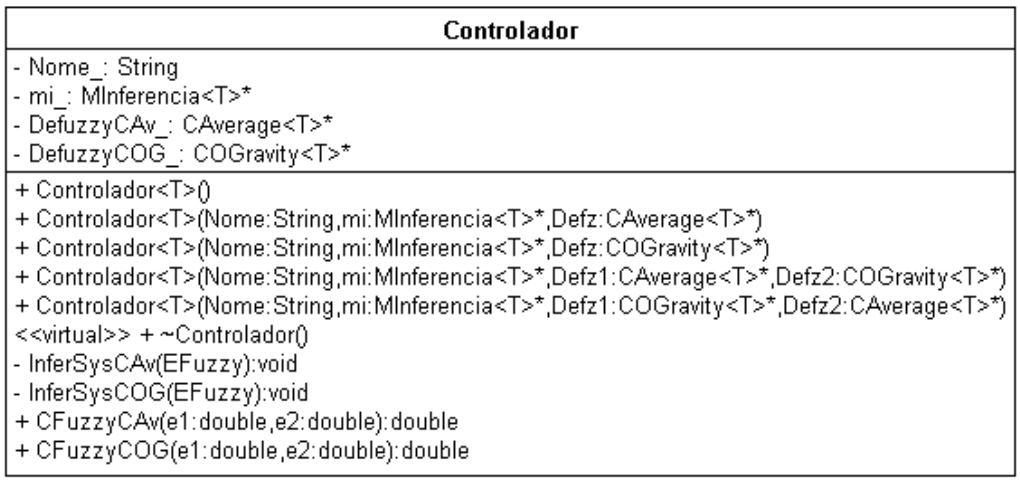


Figura 24 – Classe Controlador

Aqui o principal é o método controlador, que pode ser ou **CFuzzyCAv** ou **CFuzzyCOG**, dependendo do critério de defuzificação. Esses dois métodos implementam o controle fuzzy a partir da análise de duas variáveis crisp de entrada (e1 e e2). O método controlador usa internamente um dos dois métodos de inferência (**InferSysCAv** ou **InferSysCOG**) para a tomada de decisão.

O diagrama de classes, apresentado na Fig.25, mostra as relações entre a classe **Controlador** e as demais classes apresentadas. A próxima seção é apresentado o diagrama de seqüência.

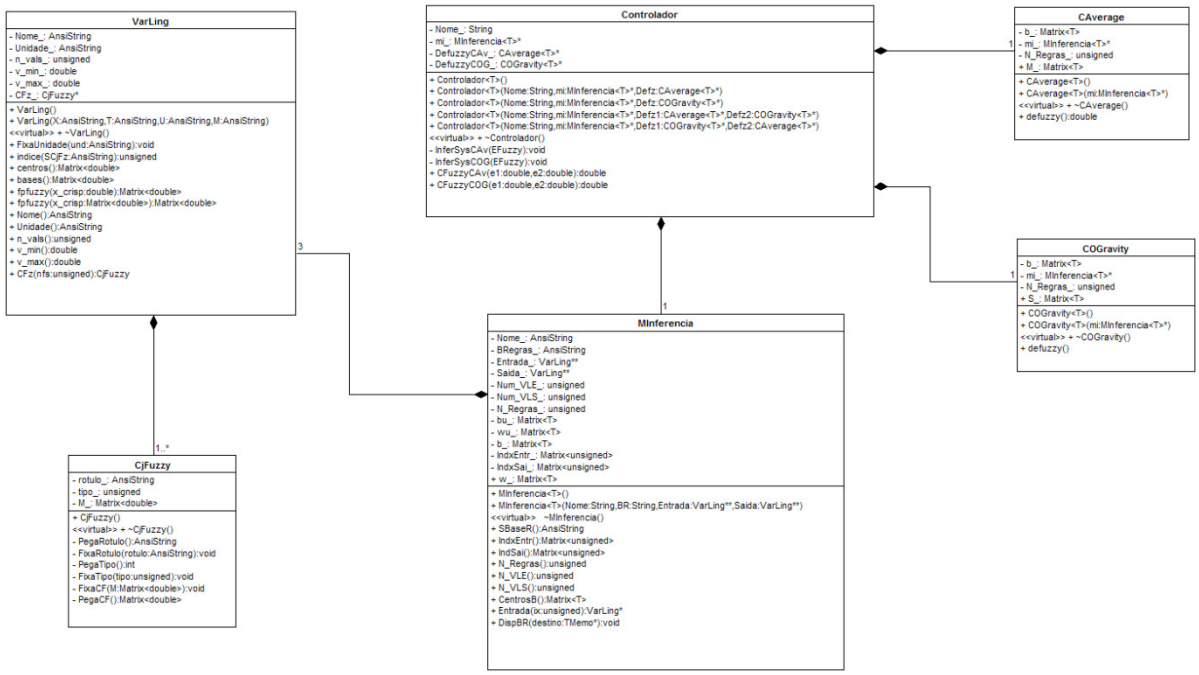


Figura 25 – Diagrama de Classes para um Controlador Fuzzy.

3.7 – CONCLUSÃO

No presente capítulo descreveu-se a utilização das classes propostas para que se façam projetos de controladores fuzzy de uma forma mais sistematizada via programação em linguagem orientada a objetos. É claro que esta não é a única concepção de projeto e, mesmo através do paradigma da orientação a objetos, outras idéias podem surgir.

Os objetivos foram o de reduzir o código, poder fazer o reuso de código e criar uma sistemática plausível para a implementação dos controladores fuzzy.

4 – SIMULADOR PARA SISTEMAS DINÂMICOS CONTÍNUOS

4.1 – INTRODUÇÃO

O uso de simuladores para sistemas dinâmicos é bastante conveniente, ainda mais quando há diversificada quantidade de recursos computacionais disponíveis para fazê-lo. Para que se ponham à prova todas as classes que até agora foram desenvolvidas, far-se-á uso de um simulador de sistemas dinâmicos contínuos, especialmente desenvolvido para este trabalho, aplicando-se método de integração numérica bastante conhecido.

Este simulador é implementado através de um objeto construído a partir de uma classe, denominada SDinamico. Os objetos desta classe têm como parâmetros a ordem do sistema, as variáveis de tempo para simulação (tempo inicial, passo, tempo final) e, como métodos, as interfaces de ajuste dos seus parâmetros, o(s) algoritmo(s) de integração numérica, os métodos de simulação, os quais contêm os algoritmos de integração numérica, e um método reservado para o usuário poder inscrever, em tempo de programação, as equações em espaço de estados do modelo do sistema pretendido para a simulação. Esta metodologia de implementação apresenta vantagens como a de inscrever sistemas, cujo modelo esteja em espaço de estados, através das suas matrizes, caso o modelo seja linear, fato que simplifica a programação do método do usuário ou, caso o modelo seja não-linear, quando se faz o uso das suas equações, lançando-se mão da biblioteca de funções matemáticas disponível na linguagem em que se está programando. Para os casos dos modelos que sejam variantes no tempo, também é reservada ao usuário uma variável de tempo, que pode ser utilizada quando as equações do sistema estiverem em função do tempo.

4.2 – SOLUÇÕES DE EQUAÇÕES DIFERENCIAIS ORDINÁRIAS POR MÉTODOS NUMÉRICOS

Um sistema físico pode ser matematicamente representado por um sistema de Equações Diferenciais Ordinárias (EDO), o qual é denominado de modelo em espaço de estado, conforme mostrado na Eq.4.1.

$$\dot{x} = f(t, x, u) \qquad (4.1)$$
$$x(t_0) = x_0$$

onde t é a variável de tempo, x é o vetor de estados, u é o vetor de entrada.

O modelo em espaço de estado se caracteriza por representar um sistema de ordem n através de um sistema de n EDOs de primeira ordem e, o uso destas equações em primeira ordem, é o que viabiliza a aplicação de diversos métodos de integração numérica para o cálculo da trajetória dos estados de um sistema, ao longo do tempo, a partir de um dado estado inicial (problema do valor inicial – PVI).

Em geral, os métodos de integração numérica, para a resolução de EDOs, têm a forma apresentada conforme:

$$x_{i+1} = x_i + m.h \tag{4.2}$$

onde x_{i+1} é o novo valor para x, x_i é o valor atual de x, m é chamado de declive e h é o passo da intergração.

O principal fator que torna diferente os diversos métodos que se apresentam como na Eq.(4.2), é o declive m, o qual é objeto de pesquisa no campo da Matemática, onde a questão é como estabelecer a sua melhor estimativa, pois a precisão do método depende dele.

Para que estes métodos possam ser aplicados é necessário que se tenha um ponto de partida (valor inicial), ou seja, um valor de x para certo t.

Como métodos de integração numérica, são citados o método de Euler e o método de Runge-Kutta de 4ª ordem, os quais são implementados neste trabalho para a realização do simulador de sistemas dinâmicos contínuos. Esses métodos são ditos de passo simples que utilizam estimativas através de derivadas dentro do intervalo do passo para realizarem suas aproximações [9] [11].

4.3 – O MÉTODO DE EULER

No método de Euler, a estimativa para o declive é fornecida diretamente pela primeira derivada, ou seja:

$$m = f(t_i, x_i, u_i) \tag{4.3}$$

onde $f(t_i, x_i, u_i)$ é a equação diferencial Eq.(4.1) calculada para (t_i, x_i, u_i). Então a Eq.(4.3) substituída em (4.2) fica conforme:

$$x_{i+1} = x_i + f(t_i, x_i, u_i).h \tag{4.4}$$

que é uma extrapolação linear sobre o passo h.

Tomando-se como valor inicial (t_i, x_i, u_i) e expandindo-se x em série de Taylor:

$$x_{i+1} = x_i + f(t_i, x_i, u_i).h + \frac{f'(t_i, x_i, u_i).h^2}{2!} + \frac{f''(t_i, x_i, u_i).h^3}{3!} + \cdots \tag{4.5}$$

onde $h = t_{i+1} - t_i$.

É possível de se ver que o método de Euler Eq.(4.4) corresponde à expansão em série de Taylor Eq.(4.5) até o segundo termo.

4.4 – OS MÉTODOS DE RUNGE-KUTTA

Os métodos de Runge-Kutta são métodos numéricos que se destinam a resolver EDOs de primeira ordem sem a necessidade do cálculo de derivadas superiores, apresentando três propriedades gerais [9]:

1- São métodos de passo único onde, para que se encontre x_{i+1}, necessita-se apenas da informação do ponto anterior (t_i, x_i, u_i).
2- Concordam com a expansão em série de Taylor até o termo h^p, onde h é o passo e p é a ordem do método.
3- Não é necessário o cálculo das derivadas de $f(t, x, u)$ pois se faz o uso apenas da própria função.

Na sua forma mais geral, os métodos de Runge-Kutta se apresentam conforme o seguinte:

$$x_{i+1} = x_i + m(t_i, x_i, u_i, h)h \tag{4.6}$$
$$m = a_1 k_1 + a_2 k_2 + \cdots + a_n k_n \tag{4.7}$$

Estágio 1: $k_1 = f(t_i, x_i, u_i)$ (4.8)
Estágio 2: $k_2 = f(t_i + p_1 h, x_i + q_{11} k_1 h, u_i)$
Estágio 3: $k_3 = f(t_i + p_2 h, x_i + q_{21} k_1 h + q_{22} k_2 h, u_i)$
$\vdots$ $\vdots$
Estágio n: $k_n = f(t_i + p_{n-1} h, x_i + q_{n-1,1} k_1 h + q_{n-1,2} k_2 h + \cdots + q_{n-1,n-1} k_{n-1} h, u_i)$

onde m é uma função de incremento, também denominada de declive associado ao intervalo e escrita na sua forma geral conforme a Eq.(4.7), onde $a_1, a_2, \cdots, a_n$ são constantes e $k_1, k_2, \cdots, k_n$ são as derivadas dentro de cada passo e são usadas como relações de

recorrência, ou seja, um valor de k é usado para o cálculo de outro, podendo ser determinados conforme as Eqs.(4.8). É importante observar que, entre um passo e outro de integração, cada k é calculado em um estágio da iteração.

Para que se defina um método específico, primeiramente o valor de n, ou seja, o número de estágios do método deve ser especificado, para que posteriormente se possam calcular as constantes $p_1, p_2, \dots, p_{n-1}$, $q_{1,1}, \dots, q_{n-1,n-1}$ e $a_1, a_2, \dots, a_n$ [9][10][11].

O valor de n para os métodos de Runge-Kutta, apresenta particularidades que se destacam em cada caso. Para $n = 1$, define-se um algoritmo de primeira ordem, com denominação usualmente abreviada na literatura como RK1, que implementa o método de Euler, visto na Seção 4.3.

Para $n = 2$, define-se o algoritmo RK2 que, dependendo das considerações utilizadas no cálculo das constantes p, q e a, tem-se três variações para o algoritmo conforme a Tabela 17.

Tabela 17 – Métodos de Runge-Kutta de 2ª ordem.

Método	Heun	Polígono melhorado	Ralston
x_{i+1}	$x_i + \frac{1}{2}(k_1 + k_2)h$	$x_i + k_2 h$	$x_i + \frac{1}{3}(k_1 + 2k_2)h$
k_1	$f(t_i, x_i, u_i)$	$f(t_i, x_i, u_i)$	$f(t_i, x_i, u_i)$
k_2	$f(t_i + h, x_i + hk_1, u_i)$	$f(t_i + \frac{h}{2}, x_i + \frac{hk_1}{2}, u_i)$	$f(t_i + \frac{3h}{4}, x_i + \frac{3hk_1}{4}, u_i)$

Para $n = 3$, define-se o algoritmo RK3, de terceira ordem, do qual uma das suas hipóteses de implementação é apresentada conforme:

$$x_{i+1} = x_i + \frac{1}{6}(k_1 + 4k_2 + k_3)h \tag{4.9}$$

Onde: $k_1 = f(t_i, x_i, u_i)$

$$k_2 = f(t_i + \frac{h}{2}, x_i + \frac{k_1 h}{2}, u_i)$$

$$k_3 = f(t_i + h, x_i + k_1 h + 2k_2 h, u_i)$$

Para $n = 4$ tem-se o algoritmo que implementa o método de Runge-Kutta de 4ª ordem clássico, ou RK4, conforme o seguinte:

$$x_{i+1} = x_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{4.10}$$

Onde: $k_1 = f(t_i, x_i, u_i)$

$$k_2 = f(t_i + \frac{h}{2}, x_i + \frac{h}{2}k_1, u_i)$$

$$k_3 = f(t_i + \frac{h}{2}, x_i + \frac{h}{2}k_2, u_i)$$

$$k_4 = f(t_i + h, x_i + hk_3, u_i)$$

Este é o algoritmo utilizado dentre os que são apresentados neste trabalho.

É importante observar que, para os parâmetros k do algoritmo descrito pela Eqs.(4.9) e (4.10), $f(t_i, x_i, u_i)$ é própria função que deve ter inscrito o modelo (equações de estado) do sistema que está sendo simulado. Esta função é definida como método da classe que cria os objetos que realizam a simulação e fica disponível para que o usuário insira, em tempo de programação, as equações que definem o modelo em espaço de estados do sistema em questão.

Uma característica de grande valor neste método, é que o seu algoritmo pode ser escrito na forma vetorial, o que permite que sistemas cujas ordens sejam maiores que 1 possam ser simulados.

4.5 – CLASSE PARA IMPLEMENTAR UM SIMULADOR DE SISTEMAS DINÂMICOS CONTÍNUOS

Através da Programação Orientada a Objetos pode-se desenvolver uma classe especializada para gerar objetos que façam a simulação de sistemas dinâmicos contínuos [12] [13]. Uma proposta de classe é a **SDinamico**, que é apresentada na Fig.26.

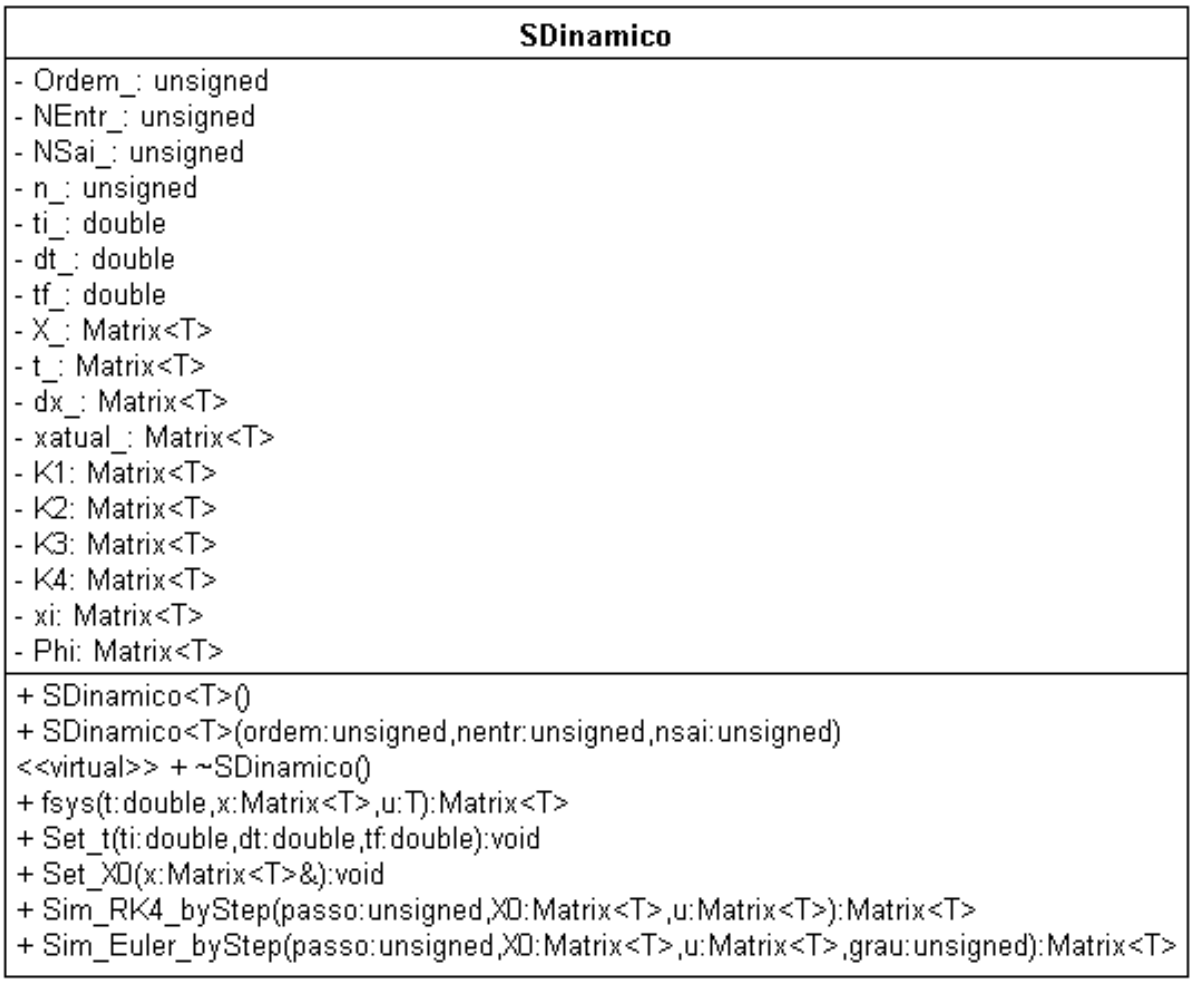


Figura 26 – Classe SDinamico

Os objetos pertencentes à classe **SDinamico**, quando criados, dispõem de diversos recursos que os tornam capazes de simular um determinado sistema que venha a ter, em tempo de programação, suas equações de estado inscritas como método da classe **SDinamico**. Este método é o **fsys (t, x, u)**, o qual fica disponível para o programador inserir as equações de estado no corpo da sua função e tem um formato padrão que foi definido para este trabalho.

Abaixo é mostrado um exemplo típico do padrão de programação para **fsys** onde se inscreve as n equações de estado ($f_1, f_2, \dots, f_n$) de um sistema de ordem n.

```
Matrix<double>SDinamico::fsys(double t, Matrix<double>& x,double u)
    {
        // dx_ é declarada no template da classe SDinamico.
        // dx_( i , 0 ) corresponde à derivada de cada estado, ou seja ẋ_i.
        dx_(0,0) = f_1(t, x, u);
        dx_(1,0) = f_2(t, x, u);
              ⋮
        dx_(n-1,0) = f_n(t, x, u);

        return dx_;
    }
```

Para criar os objetos da classe **SDinamico**, é preciso que já se tenha definido **fsys(t, x, u)**, conforme descrito acima, para então fazer uso do criador da classe **SDinamico** chamado **SDinamico(ordem, nentr, nsai)**, onde se deve informar a ordem **n** do sistema que foi inscrito em **fsys**, o número de entradas do sistema **nentr** e o número de saídas do sistemas **nsai**. Como exemplo de utilização desta classe na criação de um objeto, considere que o nosso sistema seja um pêndulo rotativo invertido (sistema de 4ª ordem). Logo se deve pegar as equações de estado do modelo deste pêndulo e criar a sua **fsys**. Em seguida, a criação de um objeto que seja o simulador do pêndulo, deve ser feita como segue:

```
SDinamico PRot(4,1,1);
```

A partir do momento em que o objeto simulador da classe **SDinamico** é criado, pode-se lançar mão dos seus recursos. O objeto **PRot** será a referência para esses recursos, onde todos os métodos criados para a classe **SDinamico** também estarão relacionados à ele.

Para que se possa fazer o uso adequado desses objetos, alguns parâmetros referentes à simulação propriamente dita precisam ser inicializados, tais como o tempo inicial (ti), o intervalo do passo (dt) e o tempo final (tf). Para tal, o método **Set_t(ti, dt, tf)** está disponível e, depois da criação do objeto do simulador, sua utilização inicializará estes parâmetros. O método **Set_t(ti, dt, tf)** calcula o valor de **n_** (número de passos da simulação) e calcula **t_**

(tempo em cada passo da simulação) através do método **gern(ti,dt,tf)** da classe **Matrix**, pois **t_** é objeto desta classe dentro da classe **SDinamico**(conceito de herança)[5][14].

A classe **SDinamico** também inclui métodos de consulta aos seus atributos, uma vez que tais atributos são declarados como "private", não se pode acessá-los de forma direta(conceito de encapsulamento)[5][14], nem para escrita e nem para leitura, então esses métodos são utilizados para que seja possível a obtenção dos dados que são importantes na programação da utilização dos objetos da classe **SDinamico**. Esses métodos são facilmente identificados no "template" da classe, pois são métodos que não apresentam parâmetros, ou seja, têm "void" nos parênteses da função e apresentam apenas uma instrução no corpo da função: "return..." [14].

Para que se possa realizar a simulação, ou seja, obter de um sistema, a partir de um estado num instante inicial, todos os valores sucessivos dos estados em instantes de tempo pré-definidos até um tempo final, faz-se o uso do método que possui o algoritmo de integração numérica específico. Para **SDinâmico** tem-se dois métodos disponíveis: **Sim_RK4_byStep(passo, X0, u)** e o **Sim_Euler_byStep(passo, X0, u, grau)**, os quais implementam os métodos de Runge-Kutta de 4ª ordem e o método de Euler, respectivamente. Estes dois métodos executam a simulação passo-a-passo controlada pelo programador. Esta sistemática tem grande importância, pois dá a chance de se analisar (medir) o estado do sistema (observabilidade) antes do passo seguinte e, como se dispõe de uma variável **u** de controle, esta poderá então ser modificada de forma a influenciar o estado seguinte (controlabilidade), ou seja, para o próximo passo. O parâmetro **passo** nos métodos simuladores é um número inteiro que vai de **0** até **n_-1**, e serve de índice para **t_** e **X_**, para a sua consulta e atualização durante a execução de uma dada simulação. Como **passo** é a variável de uma iteração, isso faz com que que os métodos **Sim_RK4_byStep(passo, X0, u)** e o **Sim_Euler_byStep(passo, X0, u, grau)**, devam ser inseridos dentro de um "loop" de programa controlado por **passo** até que a última iteração se conclua e então é obtida uma matriz **X_** de dimensão **Ordem_**×**n_** com todos os estados e **t_** com valor de cada instante de tempo correspondente aos valores de **X_**. Por **t_** e **X_** serem declarados como "private", não podem ser obtidos diretamente através da referência ao seu objeto e sim através dos métodos **X()** e **t()** que foram programados para permitir essa leitura. Os dados que **X()** e **t()** retornam podem ser atribuídos às variáveis do mesmo tipo de **X()** e de **t()**, respectivamente, no programa de aplicação.

4.6 – CONCLUSÃO

Neste capítulo foi mostrada a implementação de um simulador de sistemas dinâmicos contínuos.

Baseando-se em métodos para a integração numérica de EDOs, mais especificamente nos métodos de Runge-Kutta de 4ª ordem e Euler, desenvolveu-se uma classe, denominada **SDinamico** que, a partir da qual, criam-se objetos capazes de simular sistemas dinâmicos contínuos, através destes métodos, em ambiente computacional.

Como resultados desta simulação são obtidos, para um determinado período e a partir de um estado inicial, os estados de um determinado sistema físico, o qual deve ser inscrito como método da classe **SDinamico.**

Este simulador teve o seu desenvolvimento motivado pela necessidade de se fazer a implementação dos controladores desenvolvidos neste trabalho e verificar o funcionamento dos mesmos.

Com a apresentação da classe **SDinamico**, é possível que seja apresentado então o diagrama de seqüência Fig.27, referido na Seção 3.6. Na Fig.27 é mostrada a interação entre um objeto da classe **SDinamico** e um objeto da classe **Controlador**.

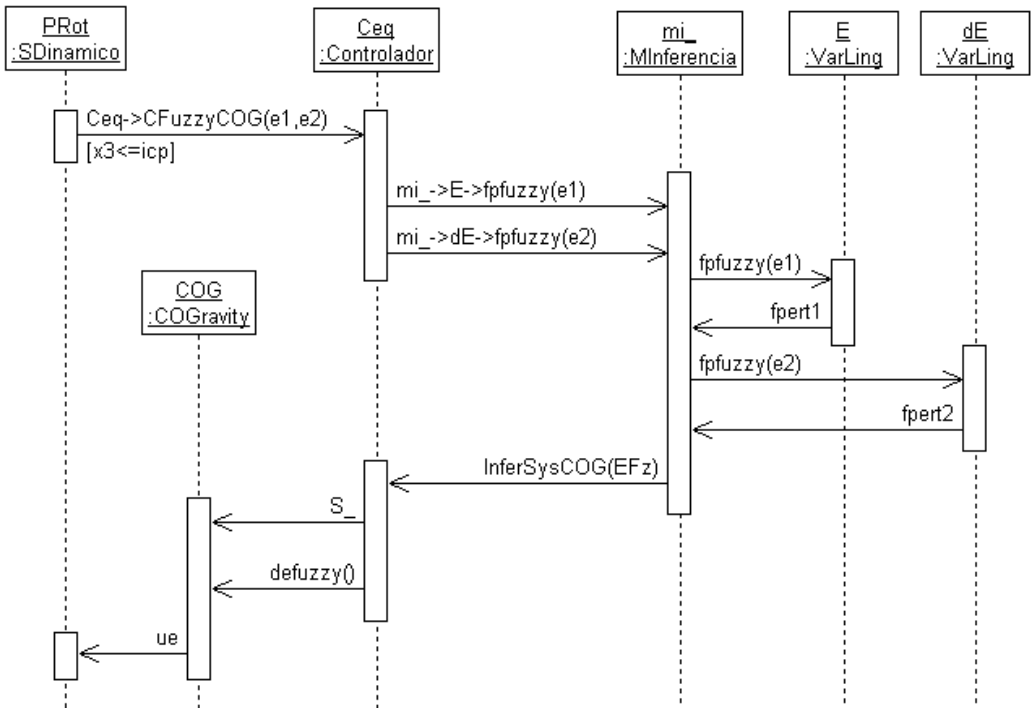


Figura 27 – Diagrama de Seqüência.

5- UM SISTEMA FUZZY DE CONTROLE PARA UM PÊNDULO ROTACIONAL INVERTIDO (PRI).

5.1- INTRODUÇÃO.

Como exemplo de aplicação do objeto de estudo do presente trabalho [4], escolheu-se o Pêndulo Rotacional Invertido (PRI) ou pêndulo de Furuta [16], que é um sistema tipicamente não linear, além de ser um sistema cujo modelo matemático é muito conhecido e bastante freqüente na literatura acadêmica, apresenta um grau razoável de dificuldade para ser controlado, isto é, tornar-se estável o equilíbrio do pêndulo no seu ponto de equilíbrio instável (posição apontando para cima).

Para esta demonstração, é necessário um trabalho prévio de análise do modelo do PRI, produzir o seu modelo linearizado em torno do ponto de equilíbrio e, em seguida, projetar o seu sistema de controle fuzzy.

Partindo das especificações definidas para o projeto dos elementos que compõem o sistema de controle fuzzy do PRI, e Com base no Capítulo 3, é feita a sua programação computacional e também a programação para o simulador de sistemas dinâmicos contínuos (Capítulo 4), o qual permite inscrever seu modelo não-linear e, com um algoritmo específico, pode-se acoplar ao PRI o sistema de controle fuzzy, implementando-se um esquema de controle por realimentação da saída, o qual usa três controladores fuzzy diferentes: o primeiro para implementar o balanço do pêndulo (“swing up”) , o segundo para equilibrar o pêndulo na posição instável e o terceiro para posicionar a base na origem. Com esse esquema, é possível o controle do equilíbrio do pêndulo e o controle da posição da base a partir de única entrada.

A programação é implementada em microcomputador PC através de duas plataformas de software. A primeira em MatLab®/Simulink®, com o objetivo de validar a técnica. A segunda plataforma é em linguagem C++, pois o C++ pode permitir um controle maior sobre o hardware do computador, possibilitando ainda o envio dos sinais, gerados em um programa, para o exterior da máquina através das suas interfaces, bem como receber sinais externos e, com isso implementar-se aplicações em controle de sistemas físicos reais.

A programação em MatLab®/Simulink®, feita sob o paradigma da orientação a objetos, é obtida com a criação das classes específicas em funções do MatLab®, o modelo do PRI é introduzido no Simulink® através de uma “s-function” e o sistema de controle através

dos blocos que fazem “chamadas” às funções do MatLab®. Para completar a sua funcionalidade deve ser programada em MatLab® uma função de inicialização que é fundamental para a criação das variáveis e dos objetos que serão tratados pelos blocos do Simulink®, esta função de inicialização segue o modelo de programação apresentada no Capítulo 3. Após a inicialização (que deve portar um estado inicial para o PRI) o sistema pode ser simulado apresentando os seus resultados em gráficos.

Na programação em C++, tudo é implementado no seu ambiente de programação, ou seja, todo o código necessário à aplicação deve ser escrito pelo programador, exceto o código que implementa a interface de usuário (janelas, formulários, objetos visuais e objetos não visuais)[14], portanto, todas as classes apresentadas nos Capítulos 3 e 4, além de uma biblioteca de funções para atender às necessidades gráficas do programa e outra biblioteca especializada em cálculo matricial devem ser programadas. Neste caso tudo se consolida em uma rotina de execução que é acionada a partir da interface de usuário. A interface de usuário apresenta campos para entrada de diversos parâmetros como ganhos dos controladores, estado inicial do PRI, seleção de algoritmo (RK4 ou Euler), seleção de defuzificador opções de exibição de gráficos, janela de exibição, e botões para o acionamento de inicialização e execução do sistema.

5.2 – O MODELO MATEMÁTICO DO PÊNDULO ROTACIONAL INVERTIDO E A SUA ANÁLISE

O pêndulo rotacional invertido (PRI) consiste de um eixo vertical rotativo, que é movimentado por um motor DC, que compõe a base do PRI, de um eixo horizontal com uma das extremidades conectadas à parte superior do eixo vertical e a outra extremidade conectada a um pêndulo conforme a Fig.28.

O PRI possui apenas uma entrada de controle que é composta pelos terminais de acionamento do motor DC. É um sistema não linear de quarta ordem e possui dois pontos de equilíbrio, um estável, quando o pêndulo está voltado para baixo e um instável, quando o pêndulo está voltado para cima.

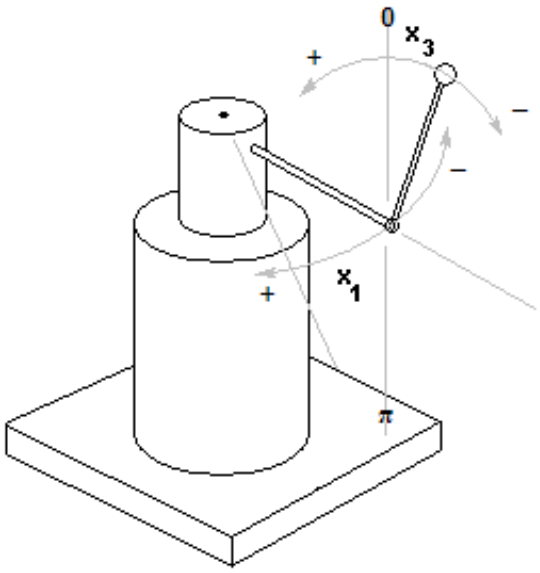


Figura 28 – Pêndulo Rotacional Invertido (PRI).

O seu modelo matemático apresenta as equações de estados descritas conforme o seguinte conjunto de equações:

$$\dot{x}_1 = x_2 \tag{5.1}$$

$$\dot{x}_2 = -a_p x_2 + K_p u$$

$$\dot{x}_3 = x_4$$

$$\dot{x}_4 = -\frac{K_1 a_p}{J_1} x_2 + \frac{m_1 g l_1}{J_1} sen(x_3) - \frac{C_1}{J_1} x_4 + \frac{K_1 K_p}{J_1} u$$

$$\bar{y} = (x_1, x_2, x_3, x_4)^T \tag{5.2}$$

e os seus parâmetros são definidos conforme Tabela 18 [7].

O vetor (x_1, x_2, x_3, x_4) é definido como o estado do sistema, sendo x_1, x_2, x_3 e x_4 suas variáveis de estado conforme suas definições na Tabela 18.

Para que o modelo do PRI seja linearizado em torno do ponto de equilíbrio instável, convenciona-se este ponto como sendo o estado $x = (0,0,0,0)^T$. É importante observar que o valor de K_1, deve ser de -1.9412×10^{-3} para este estado, conforme a Tabela 18.

Tabela 18 - Definição dos parâmetros para o modelo do PRI.

Parâmetro	Valor	Unidade	Significado
K_1	$\begin{cases} 1.9412 \times 10^{-3} & , \frac{\pi}{2} < x_3 < \frac{3\pi}{2} \\ -1.9412 \times 10^{-3} & , outros\ valores \end{cases}$	Kg.m/rad	Constante de torque do motor DC
K_p	74.8903	rad/V.s^2	Parâmetro do motor DC
a_p	33.0408	1/s^2	Parâmetro do motor DC
g	9.8066	m/s^2	Aceleração da gravidade
m_1	0.086184	Kg	Massa do pêndulo
l_1	0.113	m	Comprimento da haste do
C_1	2.9794x10^{-3}	N.m.s/rad	Constante de fricção
J_1	1.3011x10^{-3}	N.m.s^2	Momento de inércia
x_1	variável	rad	Posição angular da base
x_2	variável	rad/s	Velocidade angular da
x_3	variável	rad	Posição angular do
x_4	variável	rad/s	Velocidade angular do
u	variável	V	Entrada de controle

O modelo linearizado é apresentado conforme:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -33.0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 49.3 & 73.4 & -2.3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 74.9 \\ 0 \\ -111.7 \end{bmatrix} u \qquad (5.3)$$

$$\bar{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \qquad (5.4)$$

O modelo linearizado do PRI é útil para o cálculo de um controlador PD convencional que auxilia no projeto dos controladores fuzzy os quais são apresentados na Seção 5.3 onde os seus ganhos são utilizados como parâmetros para o ajuste do controlador fuzzy de equilíbrio e do controlador fuzzy da base do PRI.

Para se fazer a análise do modelo matemático do PRI, inicia-se com o cálculo dos pólos do sistema, que podem ser obtidos a partir dos autovalores da matriz **A**, apresentada na

Eq.(5.3). Estes pólos são 7.4988, 0, -9.7887 e –33.0408. Em análise conclui-se que se trata de um **sistema instável** por possuir um pólo positivo, ou seja, no lado direito do plano s.

A controlabilidade do sistema é analisada com base no posto (ou "rank") da matriz de controlabilidade, que é definida conforme:

$$\mathcal{M}_C = [B \quad AB \quad \ldots \quad A^{n-1}B] \tag{5.5}$$

Para o modelo do PRI apresentado e através da Eq.(5.5), obtém-se a seguinte matriz de controlabilidade:

$$\mathcal{M}_C = \begin{bmatrix} 0 & 100 & -2500 & 81800 \\ 100 & -2500 & 81800 & -2701300 \\ 0 & -100 & 3900 & -139200 \\ -100 & 3900 & -139200 & 4638900 \end{bmatrix} \tag{5.6}$$

A matriz $\mathcal{M}_C$ tem o valor do seu posto igual a 4 e a ordem do sistema também é 4, portanto o sistema é **controlável**.

A observabilidade é analisada com base no posto da matriz de observabilidade, que é definida conforme:

$$\mathcal{M}_O = [C \quad CA \quad \ldots \quad CA^{n-1}]^T \tag{5.7}$$

Portanto, para o modelo do PRI temos:

$$\mathcal{M}_O = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & -33 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 49 & 73 & -2 \\ 0 & -33 & 0 & 0 \\ 0 & 1092 & 0 & 0 \\ 0 & 49 & 73 & -2 \\ 0 & -1742 & -168 & 79 \\ 0 & 1092 & 0 & 0 \\ 0 & -36070 & 0 & 0 \\ 0 & -1742 & -168 & 79 \\ 0 & 61423 & 5773 & -348 \end{bmatrix} \tag{5.8}$$

Para a matriz $\mathcal{M}_O$, o valor do posto é 4 e, sendo o sistema de quarta ordem, conclui-se que o sistema é **observável**.

Esta análise é importante, pois algumas condições precisam ser satisfeitas afim de que se possa usar a técnica da realimentação completa de estados para controlar o sistema estudado e, como o PRI apresenta as seguintes características, conforme a análise feita:

- É um sistema instável
- Possui apenas uma entrada de controle
- É controlável
- Possui quatro saídas
- Todos os seus estados são observáveis

Com estas características, é possível a aplicação da fórmula de Ackermann, conforme:

$$K_{AK} = [0 \quad \dots \quad 0 \quad 1]\, \mathcal{M}_C^{-1} \Phi_{\mathrm{d}}(\mathrm{s}) \tag{5.9}$$

onde:

K_{AK} é o vetor de ganhos para a realimentação dos estados do sistema.

$\mathcal{M}_C$ é a matriz de controlabilidade Eq.(5.5).

$\Phi_{\mathrm{d}}(\mathrm{s})$ é a equação característica do sistema em malha fechada com os pólos desejados e avaliada para quando $s = A$.

O vetor de ganhos K_{AK} é utilizado para implementar a realimentação completa de estados e, com isso, ser obtida uma alocação dos pólos do sistema para posições, dentro do plano s, que o tornem estável.

Para esta alocação, é necessário que as novas posições desses pólos sejam definidas de forma a atender algum critério de especificação de desempenho.

O PRI é um sistema de quarta ordem sendo a sua análise complexa por não se ter uma formulação cuja resposta no tempo possa ser analisada como é feito para os sistemas de segunda ordem, onde podem ser especificados alguns parâmetros como: tempo de subida,

sobre-sinal, tempo de acomodação e coeficiente de amortecimento. Aqui a técnica é usar o conceito de pólos dominantes, onde se definem dois dos pólos deste sistema sob as especificações de um sistema de segunda ordem, alocando-os no lado esquerdo do plano s, mais próximos da origem, fazendo com que os outros dois pólos restantes sejam alocados à esquerda desses pólos. O objetivo deste método é a obtenção de uma resposta no tempo próxima ao que foi especificado para o sistema de segunda ordem.

Como especificações para um sistema de segunda ordem cujos pólos serão os pólos dominantes para o PRI, são estabelecidos:

- Sobre-sinal: $M_p = 4\%$
- Tempo de acomodação: $T_s = 0.8s$
- Cálculo do coeficiente de amortecimento:

$$\xi = \frac{1}{\sqrt{\frac{\pi^2}{\left[\ln\left(\frac{M_p}{100}\right)\right]^2} + 1}} = 0.7156 \tag{5.10}$$

- Cálculo da freqüência natural (para 2% de erro):

$$w_n = \frac{4}{\xi T_s} = 6.9867 \quad \text{rad/s}$$

Com esses parâmetros, o sistema de segunda ordem pode ser definido através de uma função de transferência da seguinte forma:

$$G(s) = \frac{{w_n}^2}{s^2 + 2\xi w_n s + {w_n}^2} \tag{5.11}$$

Logo:

$$G(s) = \frac{48.81}{s^2 + 10s + 48.81} \tag{5.12}$$

e os pólos do sistema descrito pela Eq.(5.12) são:

$$s_1 = -5 + 4.880\, i$$
$$s_2 = -5 - 4.880\, i$$

Conforme a análise do PRI existe um pólo instável (7.4988) e um pólo na origem (0). Como critério escolhe-se mudar a alocação destes dois pólos, mantendo-se os outros dois pólos estáveis (-9.7887 e –33.0408) nas suas posições, por já se encontrarem à esquerda de s_1 e s_2, portanto, os pólos para o PRI devem ser alocados conforme:

$$[-33.0408 \quad -9.7887 \quad -5 - 4.880\,i \quad -5 + 4.880\,i] \tag{5.13}$$

A tarefa de calcular o vetor de ganhos K_{AK}, conforme a Eq.(5.9), é automatizada com o uso do programa MatLab®, através da função:

$$K_{AK} = acker(A, B, [pólos]) \tag{5.14}$$

onde, A é a matriz de estado e B é a matriz de entrada conforme o modelo linearizado do PRI apresentado na Eq.(5.3) e "pólos" é definido conforme (5.13). Obtém-se, portanto:

$$K_{AK} = [-2.8726 \quad -1.4997 \quad -11.3728 \quad -1.1618] \tag{5.15}$$

Colocando o sistema do PRI em malha fechada, ou seja, $A_{MF} = A - BK_{AK}$, tem-se como resultado o seguinte sistema:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 215.1 & 79.27 & 851.7 & 87.01 \\ 0 & 0 & 0 & 1 \\ -321 & -118.3 & -1197 & -132.1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 74.9 \\ 0 \\ -111.7 \end{bmatrix} u \tag{5.16}$$

$$\bar{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{5.17}$$

cujos pólos são aqueles definidos em (5.13).

A simulação no MatLab® do sistema em malha fechada, descrito pelas Eqs.(5.16) e (5.17), com estado inicial em $x = (0, 0, -0.3, 0)^T$, tem os seus gráficos apresentados na Fig.29.

Outra simulação, feita no Simulink®, com o uso de K_{AK}, para o modelo não-linear mostrado na Fig.30, auxilia este projeto em dois pontos importantes: o primeiro ponto é o de atestar que o modelo linearizado do PRI é compatível com o seu modelo não-linear, para pontos próximos ao ponto de equilíbrio (região linear), através da comparação dos gráficos da Fig.29 com os gráficos da Fig.31 e o segundo ponto é o de permitir o levantamento dos planos de fase para os sinais de entrada, cujos gráficos, apresentados na Fig.32, fornecem os dados que servem para o escalonamento das entradas e o cálculo dos ganhos de saída dos controladores fuzzy que são projetados para o PRI no presente trabalho.

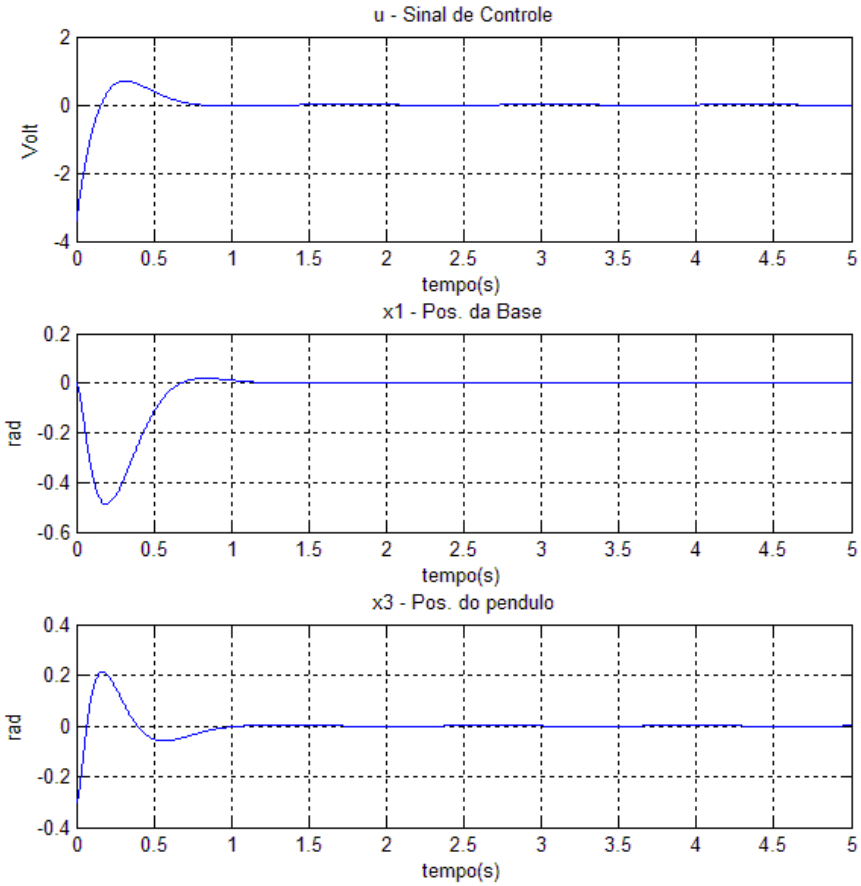


Figura 29–Simulação via MatLab® do sistema do PRI linearizado em malha fechada

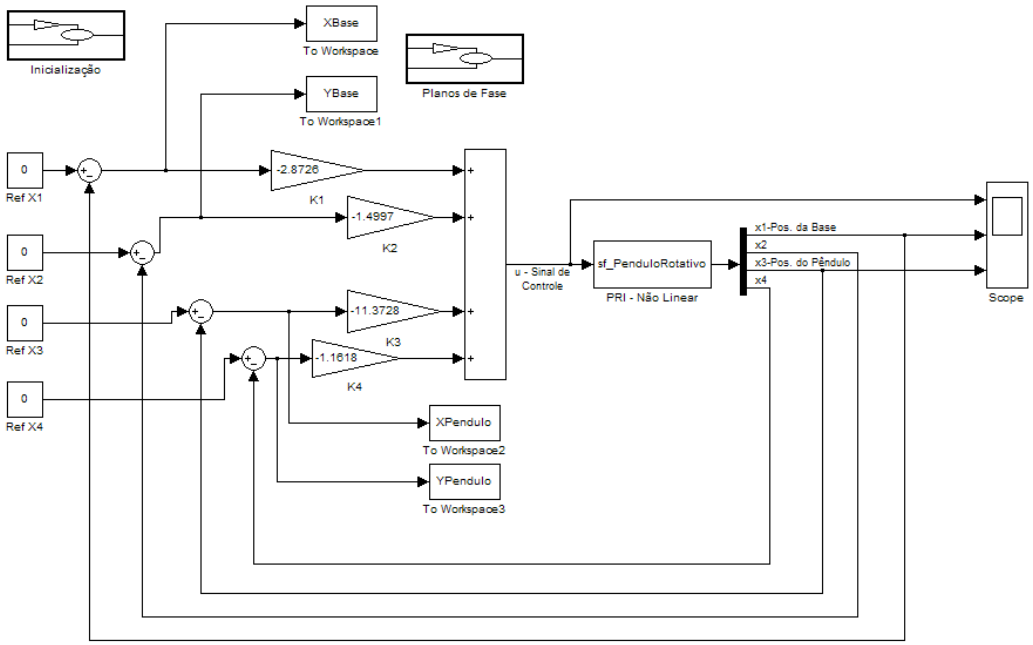


Figura 30 – Simulação via Simulink® do sist. do PRI não-linear

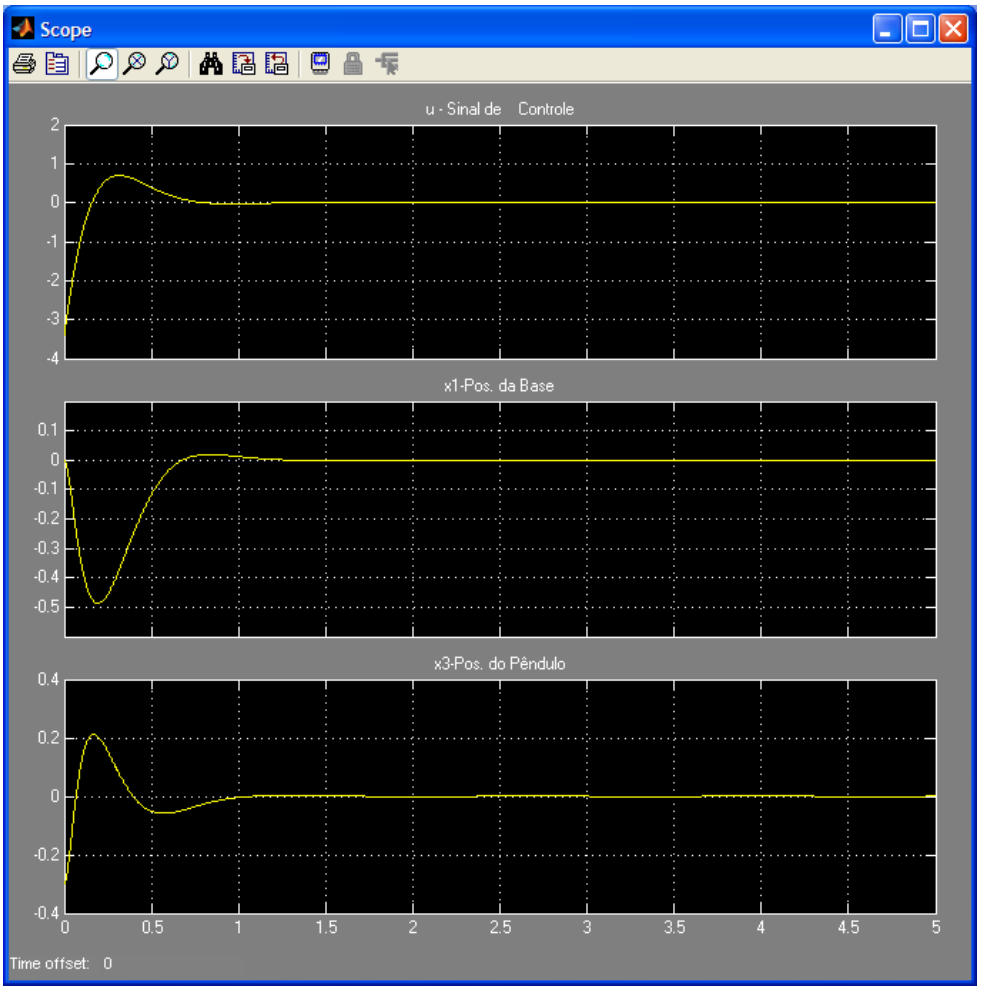


Figura 31- Simulação via Simulink® para o modelo não-linear do PRI.

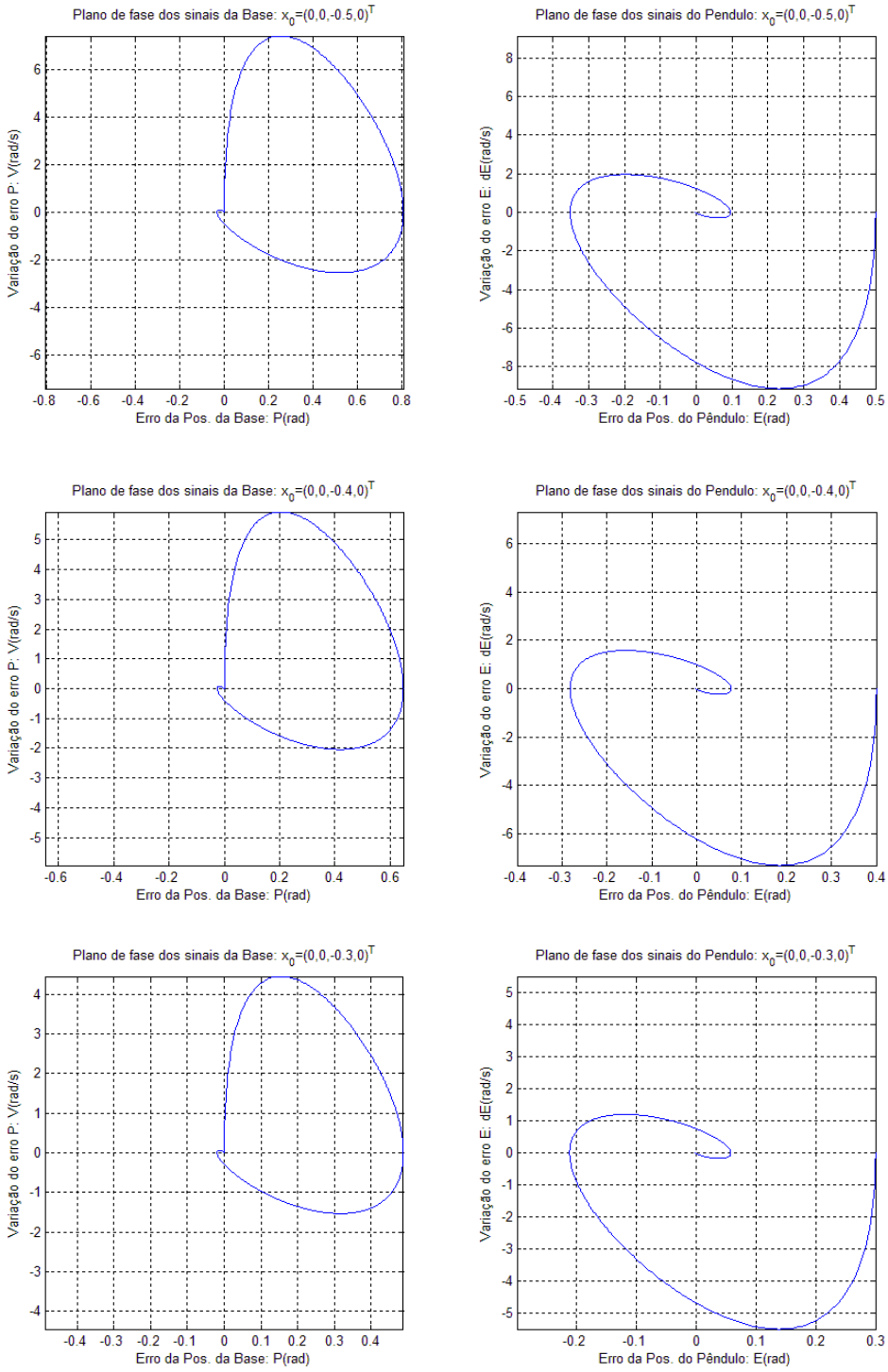


Figura 32- Planos de fase: Condição inicial para x_3 em -0.5, -0.4 e -0.3 rad.

5.3 – O SISTEMA DE CONTROLE PARA O PÊNDULO ROTACIONAL INVERTIDO.

O sistema de controle para o PRI é apresentado, em blocos, na Fig.33.

Conforme se pode observar, este sistema apresenta três controladores fuzzy, cada um com uma função diferente. Cada controlador possui duas entradas e uma saída. Suas entradas e saídas possuem ganhos associados ($G_1, G_2, G_3, G_4, G_{sw}, G_e$ *e* G_b), os quais servem para o ajuste do funcionamento dos respectivos controladores [7] [8] [15].

Para que a estratégia de controle tenha sucesso, cada controlador deve ser ativado no seu tempo. O estado em que o pêndulo e a base se encontram a cada instante é o fator que determina este tempo.

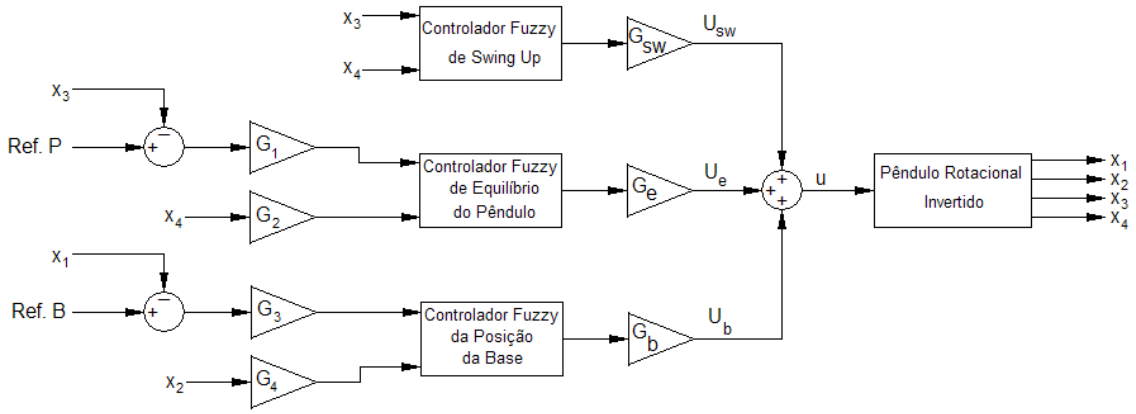


Figura 33- Diagrama em blocos do Sistema de Controle para o PRI.

Para que o controlador de "swing up" entre em operação, é estabelecida na sua programação a condição de que $|x_3| > 1\ rad$. Durante a sua ação de controle x_3 e x_4 são avaliados para que seja fornecido, na saída, um sinal que recebe um ganho G_{sw}, gerando o sinal de controle U_{sw} fazendo com que o motor da base seja acionado em um movimento de avanço e retrocesso, balançando o pêndulo, que ganha cada vez mais energia até que seja lançado próximo ao ponto de equilíbrio ($|x_3| \leq 0.5$ rad). Neste momento o controlador de "swing up" é desligado e um segundo controlador, denominado controlador de equilíbrio, passa a operar equilibrando o PRI e, fornecendo um sinal que recebe um ganho G_e, produzindo o sinal de controle U_e para manter o pêndulo estável. Com o PRI equilibrado ($|x_3| < 0.3$ rad e $|x_4| \leq 1$ rad/s) o terceiro controlador é acionado que, denominado de controlador da posição da base, gera um sinal que recebe um ganho G_b gerando por sua vez o sinal de controle U_b, responsável por fazer a base do PRI ser posicionada na sua referência.

O três sinais de controle:U_{sw}, U_e e U_b, são somados de forma que, na saída do somador, se tenha apenas um sinal de controle para a entrada do PRI.

5.4– O CONTROLADOR FUZZY DE "SWING UP".

O dispositivo que inicia o movimento do PRI, através do seu balançar, é o controlador de "swing up". Sua atuação é necessária para que o pêndulo possa ser levado da posição inicial "para baixo" e "em repouso" ($x_3 = \pi\, rad;\ x_4 = 0\, rad/s$) até próximo a posição para cima, onde o controlador fuzzy de "swing up" se desliga.

Para implementá-lo através de um controlador fuzzy, duas variáveis lingüísticas de entrada foram definidas: P, que é a posição do pêndulo e V que é a velocidade do pêndulo e uma variável de saída U_{SW} que é o sinal de saída do "swing up". P é alimentada pelo sinal de x_3 e V pelo de x_4. As respectivas variáveis e os seus conjuntos fuzzy normalizados apresentam-se na Figs.34-36.

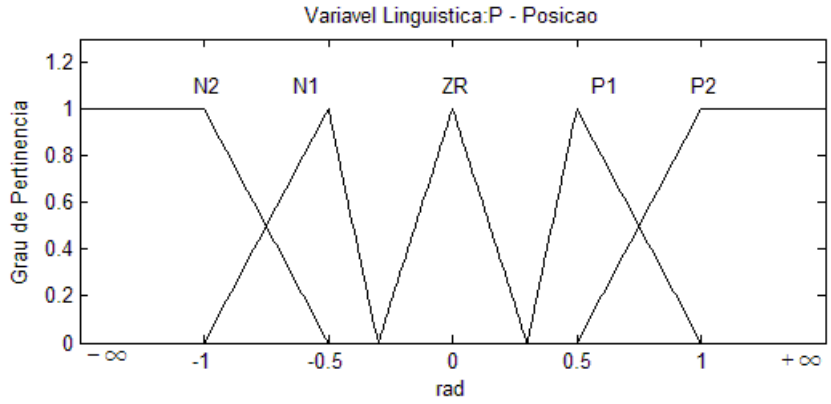


Figura 34 - Variável lingüística P (Posição do Pêndulo) – Controlador de "Swing Up".

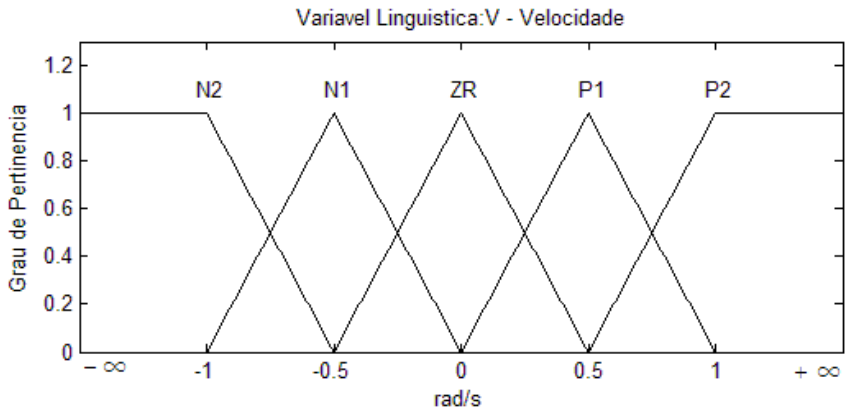


Figura 35 - Variável lingüística V (Velocidade do Pêndulo) – Controlador de "Swing Up".

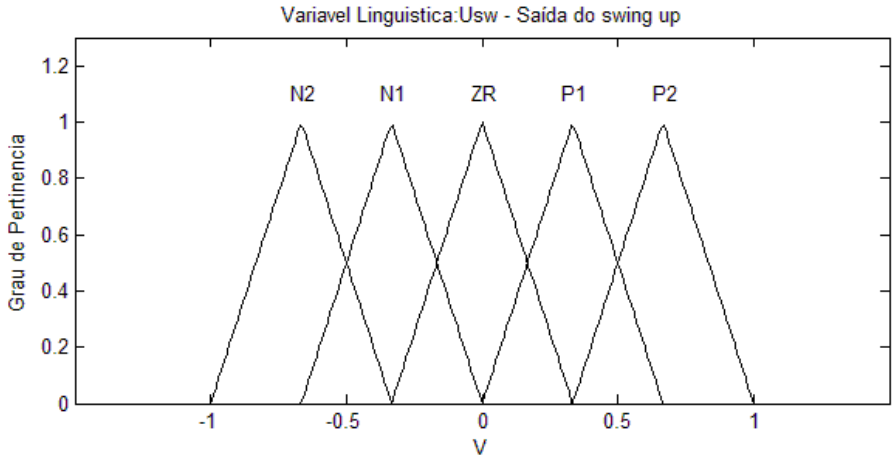


Figura 36 - Variável lingüística U_{SW} (Saída) – Controlador de "Swing Up".

São definidos os rótulos para os conjuntos fuzzy como: N2 – muito negativo, N1- pouco negativo, ZR- zero, P1-pouco positivo e P2-muito positivo.

Como as suas variáveis lingüísticas de entrada possuem cinco conjuntos fuzzy cada uma, então a base de regras fuzzy terá vinte e cinco regras. Uma proposta de base de regras fuzzy para este controlador é mostrada na Tabela 19.

Tabela 19 - Base de Regras do Swing up

		Posição P				
		N2	N1	ZR	P1	P2
Velocidade V	N2	N2	N2	ZR	N2	N2
	N1	N2	N2	ZR	N2	N2
	ZR	N2	N2	ZR	P2	P2
	P1	P2	P2	ZR	P2	P2
	P2	P2	P2	ZR	P2	P2

Este controlador é do tipo Mamdani e seu defuzificador é por centro de gravidade. O seu ajuste é feito pelo método da tentativa e erro até que se encontre um valor de ganho G_{SW} suficiente para balançar o pêndulo de forma a projetá-lo para cima fazendo-o ultrapassar, em baixa velocidade, o ponto de equilíbrio instável ($x_3 = 0\ rad$). Para essas sucessivas etapas de simulação, recomenda-se que os outros controladores (equilíbrio e base) sejam desligados, fazendo-se $G_e = 0$ e $G_b = 0$. O ajuste deve começar no estado $x = (0,0,\pi,0)^T$ e $G_{SW} = 1$ e, incrementando-se sucessivamente o seu valor até que o pêndulo consiga ultrapassar o ponto de equilíbrio. O valor obtido para G_{SW} é de 4.38.

5.5– O CONTROLADOR FUZZY DE EQUILÍBRIO DO PÊNDULO.

Para o equilíbrio do pêndulo é projetado um controlador fuzzy para o qual são definidas duas variáveis lingüísticas de entrada: E e dE e uma variável lingüística de saída: U_e. Seus conjuntos fuzzy normalizados são mostrados nas Figs.37-39 respectivamente.

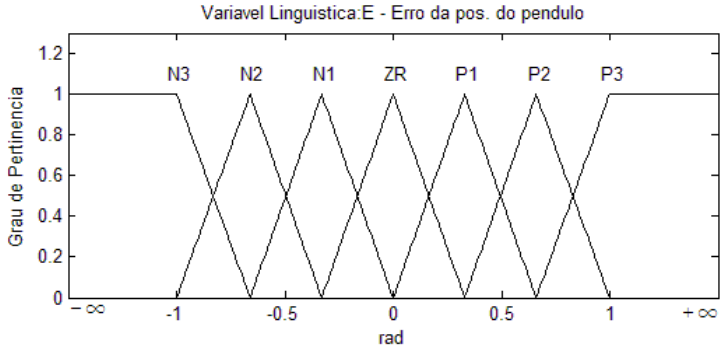


Figura 37 - Variável lingüística E (Erro da Posição do Pêndulo) – Controlador de Equilíbrio.

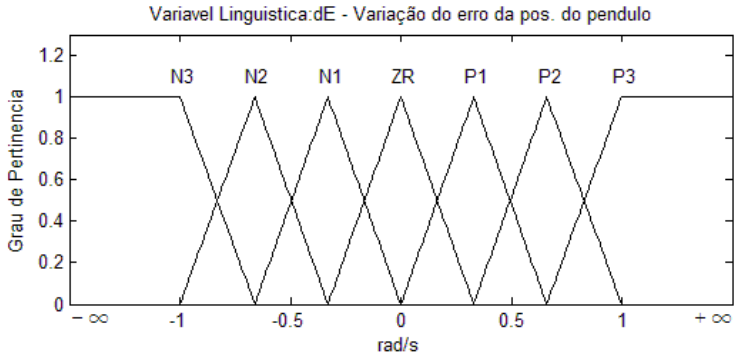


Figura 38 - Variável lingüística dE (Variação do Erro da Posição do Pêndulo) – Controlador de Equilíbrio.

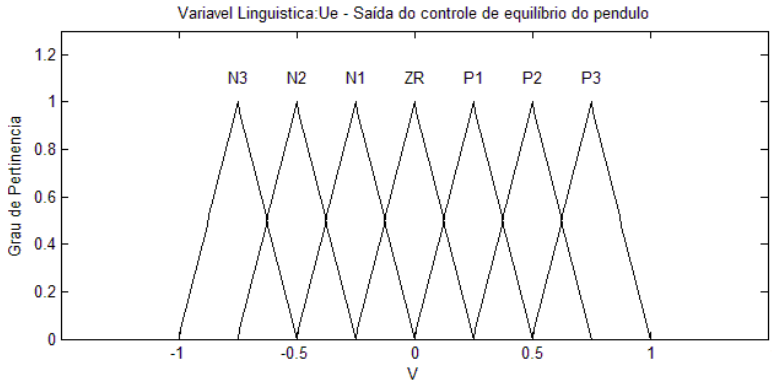


Figura 39 - Variável lingüística U_e (Saída) – Controlador de Equilíbrio.

Seus valores lingüísticos têm como rótulo: N3-muito negativo, N2-médio negativo, N1-pouco negativo, ZR-zero, P1-pouco positivo, P2-médio positivo, P3-muito positivo. Este controlador também é um sistema do tipo Mamdani com defuzificador por centro de gravidade e possui uma base de regras com quarenta e nove regras conforme mostrado na Tabela 20.

Tabela 20-Base de Regras para o Pêndulo e para a Base

		Erro E / Erro B						
		N3	N2	N1	ZR	P1	P2	P3
Derivada do Erro dE/ Derivada do Erro dB	N3	P3	P3	P3	P3	P2	P1	ZR
	N2	P3	P3	P3	P2	P1	ZR	P1
	N1	P3	P3	P2	P1	ZR	P1	P2
	ZR	P3	P2	P1	ZR	P1	P2	P3
	P1	P2	P1	ZR	P1	P2	P3	P3
	P2	P1	ZR	P1	P2	P3	P3	P3
	P3	ZR	P1	P2	P3	P3	P3	P3

Os ganhos G_1, G_2 e G_e são calculados fazendo-se o uso do vetor de ganhos K_{AK}, vide Eq.(5.15) e do valor máximo absoluto do erro da posição do pêndulo ($|epmax|$), obtido a partir dos gráficos da Fig.32. Considerando o acionamento do controlador de equilíbrio (início do controle do pêndulo – icp) em $x_3 \leq 0.5\, rad$ e o acionamento do controlador da base (início do controle da base – icb) em $x_3 \leq 0.3\, rad$ e, a partir do gráfico correspondente da Fig.32, tem-se: $|epmax| = x_3 = 0.5$.

1- Cálculo de G_1:

$$G_1 = \frac{1}{|epmax|} = \frac{1}{0.5} = 2 \tag{5.18}$$

2- Cálculo de G_e:

$$|K_{AK}(3)| = G_1.G_e \Rightarrow G_e = \frac{|K_{AK}(3)|}{G_1} = \frac{11.3728}{2} = 5.6864 \tag{5.19}$$

3- Cálculo de G_2:

$$|K_{AK}(4)| = G_2.G_e \Rightarrow G_2 = \frac{|K_{AK}(4)|}{G_e} = \frac{1.1618}{5.6864} = 0.2043 \tag{5.20}$$

5.6– O CONTROLADOR FUZZY DA POSIÇÃO DA BASE.

Para o controlador fuzzy da base, são definidas duas variáveis de entrada: erro da posição da base B e variação do erro da posição da base dB e a variável lingüística da saída U_b, conforme mostradas nas Figs.40-42. Seus rótulos são iguais aos rótulos que foram definidos para o controlador fuzzy de equilíbrio e a sua base de regras fuzzy também tem quarenta e nove regras, podendo ser usada a mesma da Tabela 20. O ajuste dos seus ganhos, G_3, G_4 e G_b são feitos de forma similar ao dos ganhos do controlador fuzzy de equilíbrio.

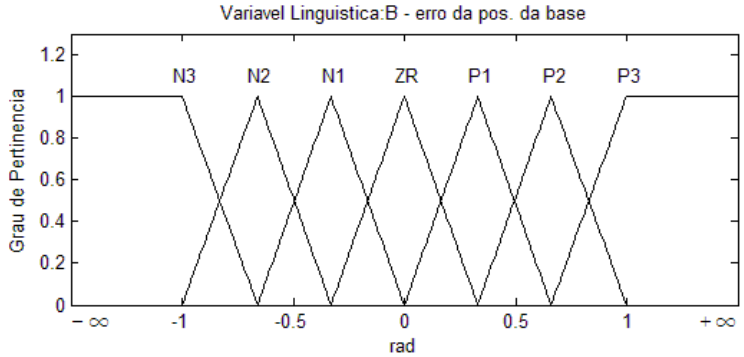


Figura 40 - Variável lingüística B (Erro da Posição da Base) – Controlador da Posição da Base.

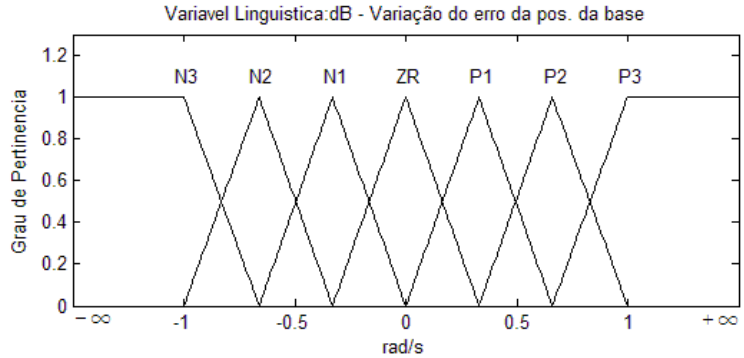


Figura 41 - Variável lingüística dB (Variação do Erro da Pos.da Base) – Controlador da Pos. da Base.

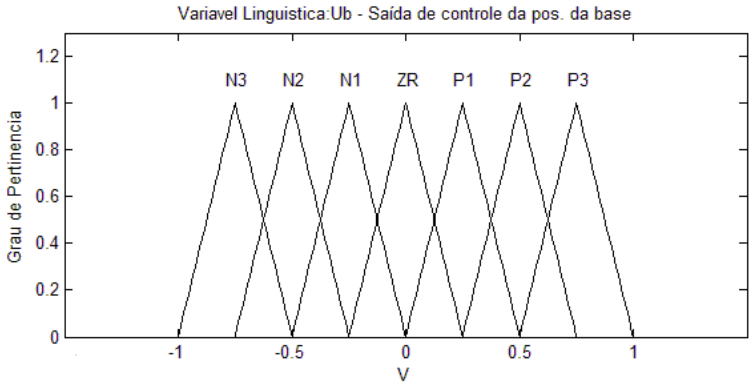


Figura 42 - Variável lingüística U_b (Saída) – Controlador da Posição da Base.

Utilizando o vetor de ganhos K_{AK} Eq.(5.15) e o valor máximo absoluto do erro da posição da base ($|ebmax|$), obtido a partir dos gráficos da Fig.32, onde se considera o acionamento do controlador de equilíbrio (icp) em $x_3 \leq 0.5\,rad$ e o acionamento do controlador da base (icb) em $x_3 \leq 0.3\,rad$, tem-se: $|ebmax| = 0.82\,rad$.

1- Cálculo de G_3:

$$G_3 = \frac{1}{|ebmax|} = \frac{1}{0.82} = 1.2195 \tag{5.21}$$

2- Cálculo de G_b:

$$|K_{AK}(1)| = G_3.G_b \Rightarrow G_b = \frac{|K_{AK}(1)|}{G_3} = \frac{2.8726}{1.2195} = 2.3555 \tag{5.22}$$

3- Cálculo de G_4:

$$|K_{AK}(2)| = G_4.G_b \Rightarrow G_4 = \frac{|K_{AK}(2)|}{G_b} = \frac{1.4977}{2.3555} = 0.6367 \tag{5.23}$$

Com os valores dos ganhos definidos para cada controlador do sistema de controle do PRI, é possível que se implemente a sua simulação, as quais são apresentadas a seguir. É importante observar que para cada condição apresentada pelos planos de fase da Fig.32 e usando-se as Eqs.(5.18)-(5.23), pode-se obter diferentes valores de ajuste para $G_1, G_2, G_3, G_4, G_e\ e\ G_b$, conforme mostrado na Tabela 21.

Tabela 21 – Ajustes dos Ganhos dos Controladores Fuzzy para o PRI

Condição de Operação		Parâmetros obtidos no Plano de Fase		Valores dos Ganhos					
				Pêndulo			Base		
icp	icb	$\|epmax\|$	$\|ebmax\|$	G_1	G_2	G_e	G_3	G_4	G_b
0.5	0.3	0.5	0.82	2.0000	0.2043	5.6864	1.2195	0.6367	2.3555
0.4	0.2	0.4	0.63	2.5000	0.2554	4.5491	1.5873	0.8287	1.8097
0.3	0.1	0.3	0.5	3.3333	0.3405	3.4118	2.0000	1.0441	1.4363

5.7- IMPLEMENTAÇÃO NO MATLAB®/SIMULINK®

Para a simulação no MatLab®/Simulink® é necessário um programa, em MatLab®, conforme apresentado no Anexo I, e também a criação de um modelo no Simulink® confrome mostrado na Fig.43. O PRI que aparece no modelo é representado por uma s-function, do MatLab®, onde as Eqs.(5.1) devem ser inscritas para a simulação. Deve-se ajustar o solucionador do Simulink® para ode4(Runge-Kutta) e um passo fixo de 0.001s. Euler pode ser usado, porém, devido a problemas de precisão e convergência deste algoritmo,

o passo deve ser diminuído para 0.0001s para se obter algum resultado, o que produzirá uma simulação muito demorada.

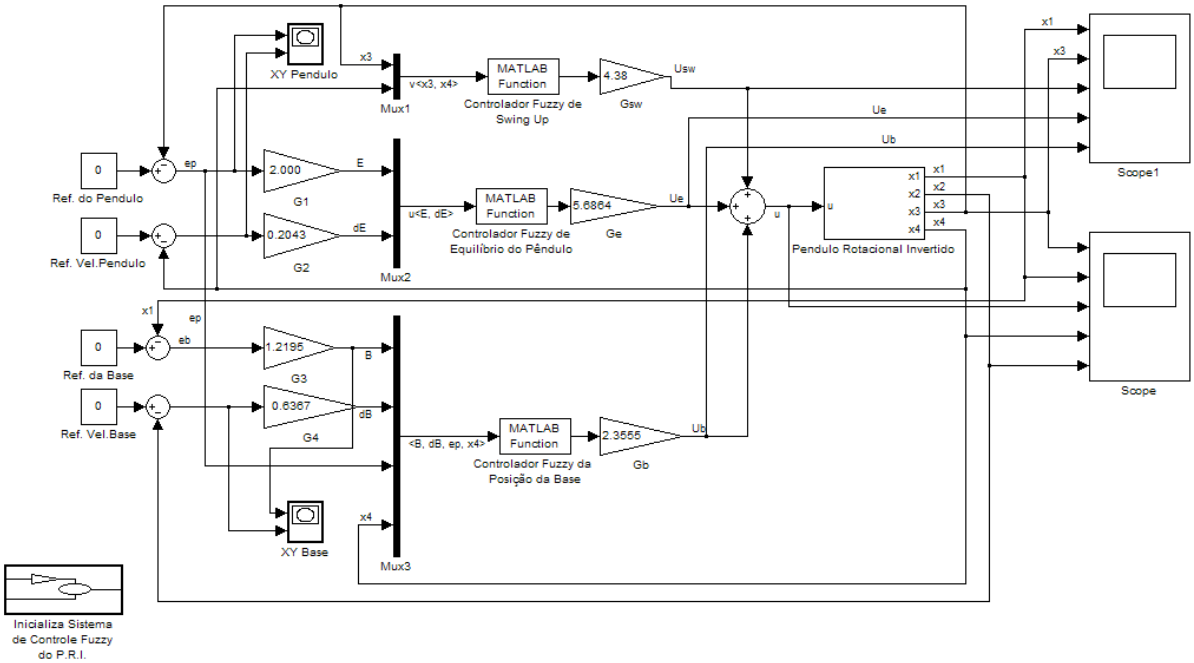


Figura 43 – Modelo do sistema de controle para o PRI no Simulink®.

Para os blocos MatLab Function que desempenham o papel dos controladores, é preciso programar três funções do MatLab® para estes blocos, das quais os controladores, criados pelo programa de inicialização, vão chamar os métodos controladores CFuzzyCAv ou CFuzzyCOG, dos objetos controladores criados, condicionando as suas operações, ou seja, não se pode acionar CFuzzyCAv ou CFuzzyCOG de forma direta pois sem as condições impostas para o funcionamento de cada controlador não será possível controlar o PRI. A Fig.42 apresenta os resultados para uma simulação de 10s.

Analisando a Fig.44 são vistos os gráficos dos seguintes sinais: x_3- posição do pêndulo, x_1- posição da base, u- sinal de controle, x_4- velocidade do pêndulo e x_2 - velocidade da base. Inicialmente o PRI encontra-se em repouso na posição inferior. Após a atuação do controlador de "swing up" o pêndulo fica próximo do equilíbrio e, em torno de 2.4s finalmente é estabilizado. Para manter esse equilíbrio, a base se movimenta da sua referência e, para retornar, o controlador da base atua neste sentido. Esta operação é comprovada pelo sinal da posição da base que fica oscilando em torno da refrência.

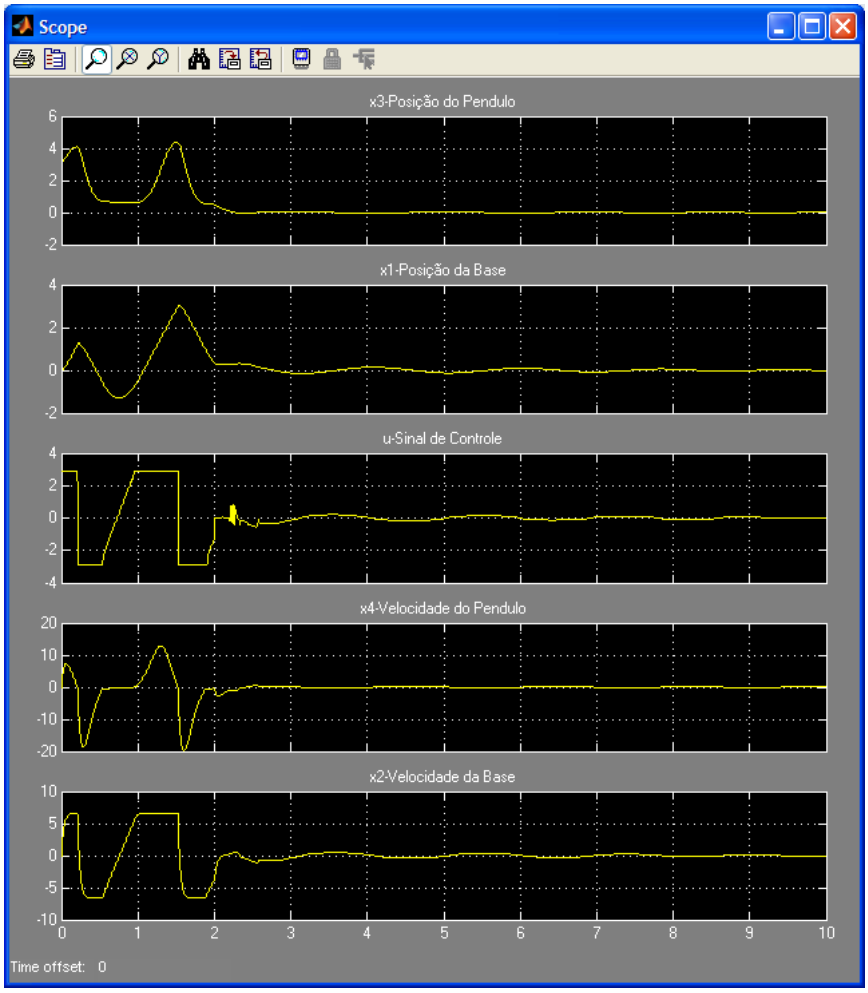


Figura 44 – Resultados da simulação do eqilíbrio do PRI.

São apresentados, na Fig.45, os gráficos dos sinais x_1-posição da base, x_3-posição do pendulo e as três componentes do sinal de controle u, ou seja, U_{sw}- sinal de controle do "swing up", U_e- sinal de controle de equilíbrio do pêndulo e U_b- sinal de controle da base.

5.8- IMPLEMENTAÇÃO EM C++

A implementação em C++ tem o seu código apresentado no Anexo I. O seu principal objetivo é testar o projeto apresentado em uma plataforma diferente do MatLab®.

Esta etapa trata do desenvolvimento de um aplicativo de software de demonstração que na verdade acaba se tornando uma plataforma experimental para o sistema do PRI em função da grande quantidade de ajustes de parâmetros que esta oferece.

Programado em Borland® C++ Builder™, o programa em execução apresenta um painel conforme o da Fig.46, onde se podem ver os parâmetros ajustáveis dos controladores,

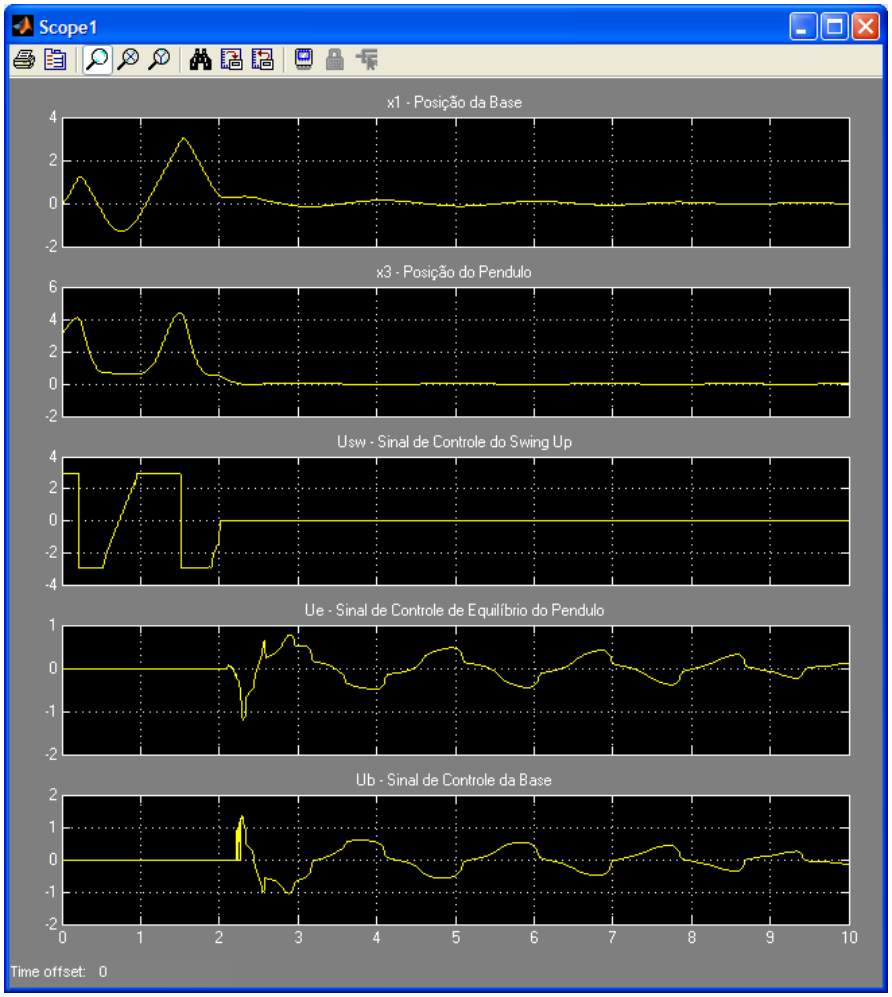


Figura 45 – Composição do sinal de controle $u = U_{sw} + U_e + U_b$.

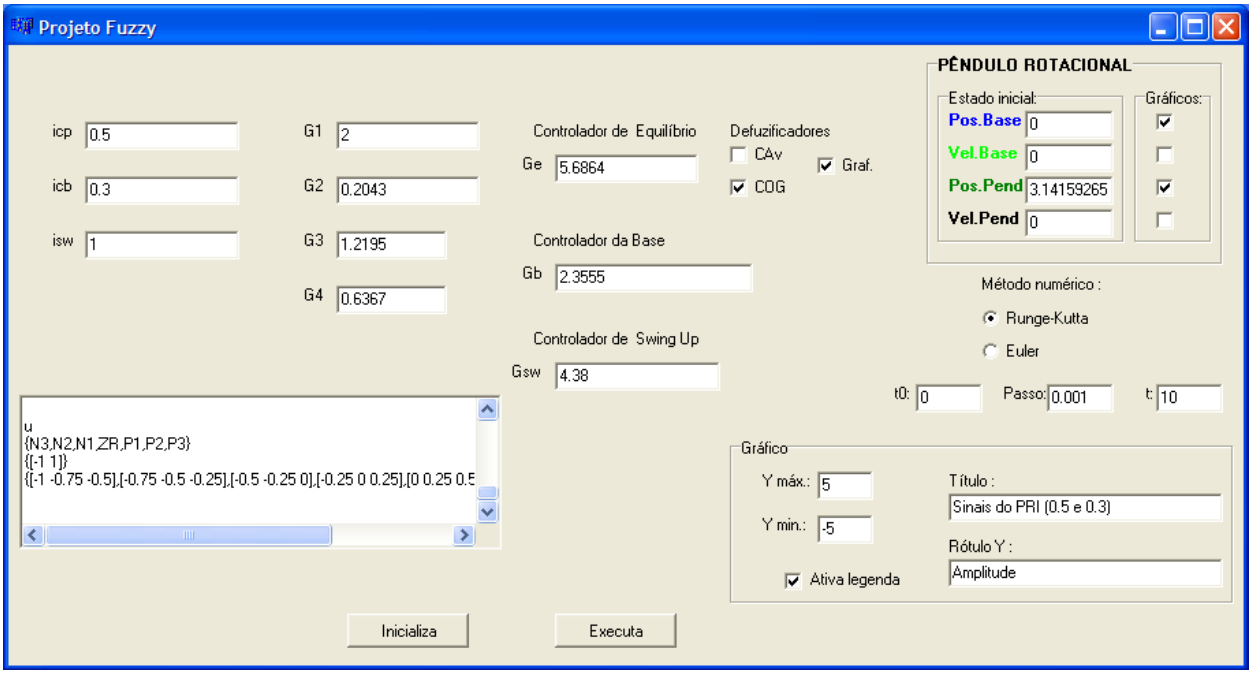


Figura 46 - Implementação em C++ para a simulação do controle do PRI.

opções de gráficos, duas opções de algoritmos numéricos de solução de equações diferenciais ordinárias e as suas variáveis de tempo e um campo de estado inicial do PRI, pois o seu modelo já está vinculado a este programa.

Depois de inicializado e executado são produzidos gráficos apresentados na Fig.47 para os sinais do PRI.

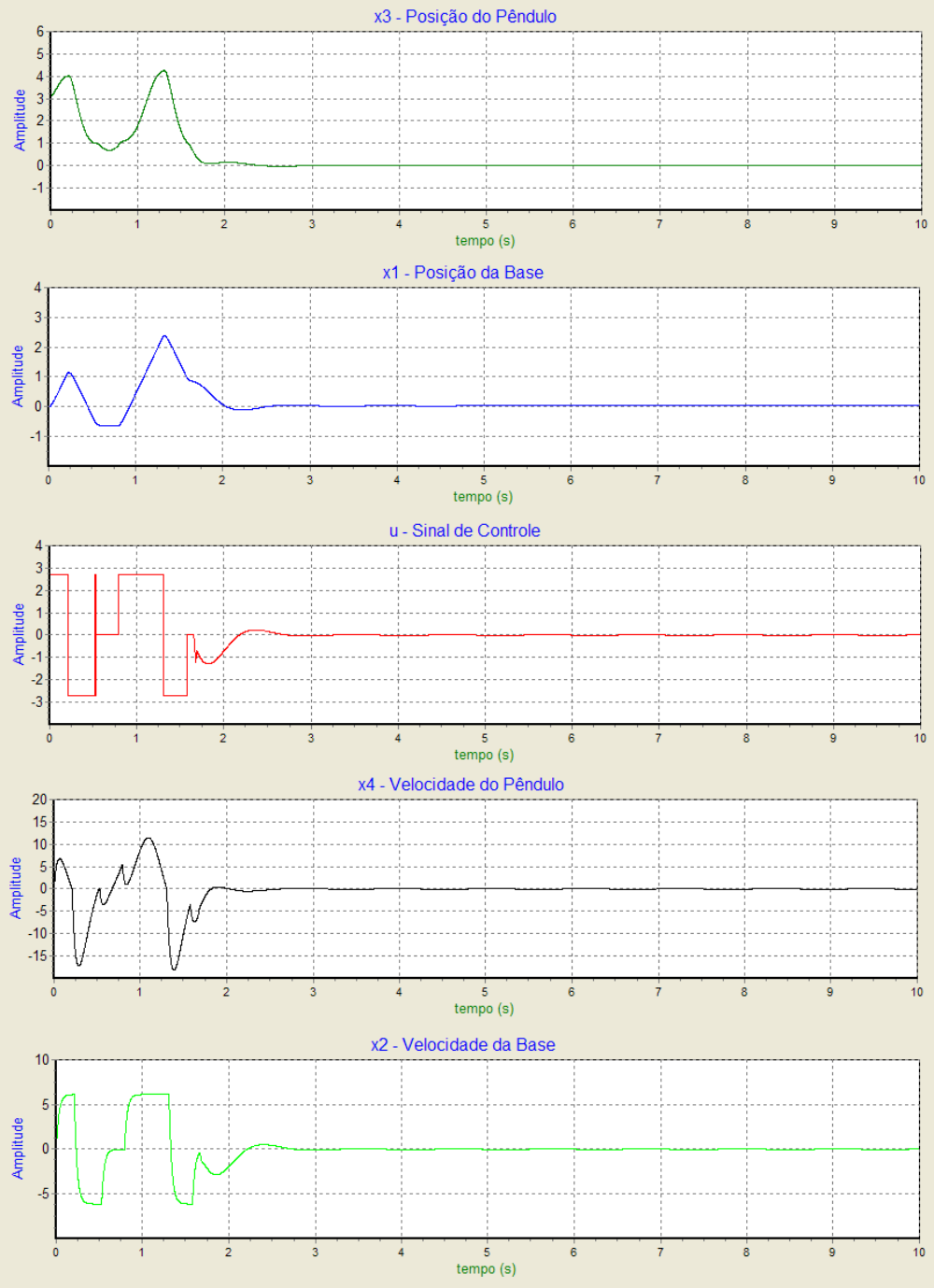


Figura 47 – Gráficos da Implementação em C++.

5.9– CONCLUSÃO

No presente capítulo reúnem-se os conceitos apresentados nos Capítulos 2, 3 e 4 para demonstração, via simulação, da implementação de Controladores Fuzzy, usando-se como exemplo o modelo de um Pêndulo Rotacional Invertido, ou pêndulo de Furuta[16], como é denominado. Após sua análise, sob o ponto de vista da estabilidade, controlabilidade e observabilidade, é realizado um projeto de um sistema de controle convencional, sob especificação de desempenho de resposta no tempo e com base na realimentação de estados, cujos ganhos são calculados pela fórmula de Ackermann para o modelo linear do PRI.

Realizada a aplicação do controle convencional no modelo não-linear do PRI e constatada a sua validação via comparação gráfica, são levantados os Planos de Fase a partir das entradas deste controlador via simulação no MatLab®/Simulink®.

Projetados os Controladores Fuzzy normalizados, cujos ganhos foram calculados (escalonamento das entradas e das saídas) com base nas informações dos Planos de Fase e no vetor de ganhos calculado pela fórmula de Ackermann, foi construído um modelo de simulação no Simulink® onde, aplicando-se os valores dos ganhos calculados, verificou-se o correto funcionamento do sistema de controle, atendendo as suas especificações.

Com os dados válidos constrói-se a aplicação em C++, cujos gráficos apresentados (Fig.47), comparados aos resultados gráficos do Simulink®(Fig.44), se compatibilizam, onde o sistema apresenta uma resposta aproximada às especificações de desempenho definidas para o seu projeto, ou seja, acomodação em torno de 0.8s (Fig.47 – Gráfico: x3-Posição do Pêndulo. Considerando o início da ação de controle de equilíbrio após a etapa de “swing up”).

6 - CONSIDERAÇÕES FINAIS

O emprego de Programação Orientada a Objetos mostrou-se adequado para a implementação de controladores Fuzzy em ambiente computacional, uma vez que proporcionou a redução de código simplificando sua programação. Esta redução se deve primeiro à técnica de indexação dos elementos da Base de Regras e dos Conjuntos Fuzzy de cada variável lingüística do sistema. Esta técnica elimina a necessidade de se programar a base de regras linha a linha, fato que reduz consideravelmente a possibilidade de se cometer erros durante a programação. E em segundo que a definição das classes permite uma criação de objetos de forma bastante sistemática, inclusive com a visão dos blocos de cada controlador empregado. É possível de se avaliar a quantidade de linhas necessária em cada controlador:

Considerando um controlador de duas entradas e uma saída tem-se:

1-Cada variável lingüística é definida com 5 linhas de comando, portanto, para três variáveis (duas de saída e uma de entrada) são necessárias 15 linhas de comando.

2-A Base de regras Fuzzy é definida com apenas uma linha de comando, pois a mesma é introduzida no sistema em formato matricial, ou seja, se o sistema tiver 49 regras, por exemplo, as 49 linhas de programa correspondentes serão substituídas por apenas uma linha.

3-A Máquina de Inferência Fuzzy é definida com apenas 3 linhas de comando.

4-Cada tipo de Defuzificador é definido com uma linha de comando.

5-O Controlador é definido com uma linha de comando.

Conclui-se então que para cada Controlador Fuzzy, conforme descrito são necessárias 31 linhas de comando.

A reutilização de código se destaca na forma de como se podem introduzir mais Controladores Fuzzy em um determinado sistema, uma vez que a programação para cada controlador, mesmo que desempenhem funções diferentes, significando uma Base de Regras adaptada para cada caso, é feita com a mesma seqüência de passos.

A sistematização dos projetos de Controladores Fuzzy apresentada simplifica a tarefa de construí-los, permitindo com que o seu projetista dê maior atenção ao projeto do controlador propriamente dito que para a programação.

6.1 –PRICIPAIS CONTRIBUIÇÕES

As contribuições deste trabalho são as seguintes:

- A apresentação e uma técnica sistemática de se programar Controladores Fuzzy.
- O código gerado para este trabalho, tanto em MatLab®/Simulink® quanto em C++, fica disponibilizado em "Compact Disc" anexo, para futuras consultas, utilização, reprodução, distribuição ou alteração.
- Um exemplo de aplicação apresentando a análise do modelo matemático do Pêndulo Rotacional Invertido e a definição de uma estratégia de controle para estabilizá-lo (via controle convencional e via controle Fuzzy).
- A apresentação de uma metodologia de ajuste de Controladores Fuzzy, através do escalonamento das suas entradas e das suas saídas, uma vez que ajustar controladores fuzzy pode não ser uma tarefa tão simples.

6.2 – SUGESTÕES PARA TRABALHOS FUTUROS

O presente trabalho pode se estender através do desenvolvimento de um "hardware" que sirva para controlar plantas físicas reais através das portas de saída do microcomputador, para tanto é necessário a criação de uma classe específica de comunicação que auxilie tal finalidade.

Outra aplicação bastante interessante é o desenvolvimento de controladores Fuzzy configuráveis em tempo real e sob determinadas condições.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ZADEH, L. A. “Fuzzy Sets”. Berkeley, California: Information and Control 8, 1965.

[2] ZADEH, L. A. “Outline of a new aproach to the analisis of complex systems and decision process”. IEEE Transactions on Systems, Man, and Cybernetics, 3, 1, Janeiro 1973.

[3] ZADEH, L. A. “The concept of a linguistic variable and its application to approximate reasoning I, II, III, “Information Sciences,” 8, 1975.

[4] LEITÃO, Alvaro L. R.; COSTA Jr, Carlos T.; BARRA Jr, Walter; SILVA, Orlando F. “Sistematização de projetos de controladores fuzzy através de programação orientada a objetos”. Salvador: VI SNCA – AINST, 2009.

[5] CESTA, André; RUBIRA, Cecília. “Tutorial: C++ como uma linguagem de programação orientada a objetos”. São Paulo: IC-UNICAMP, 1997.

[6] WANG, Li-Xin. “A course in fuzzy systems and control”. Prentice-Hall International. Upper Saddle River, NJ, USA, 1997.

[7] PASSINO, Kevin M; YURKOVICH, Stephen. “Fuzzy control”. Menlo Park: Addison-Wesley Longman Inc, 1998.

[8] KOVACIC, Zdenko; BOGDAN, Stjepan. “Fuzzy controller design : Theory and applications”. Boca Raton: CRC Press, 2006.

[9] BUTCHER, J. C. “Numerical methods for ordinary differential equations”. John Willey & Sons, Chichester, England, 2003.

[10] CHAPRA, S.; CANALE, R. P. “Numerical methods for Engineers”. McGrow-Hill International Editions, Applied Mathematics Series, 1990.

[11] PRESS, W. H.; TEUKOLSKY, S. A.; VITTERLING, W. T; FLANNERY, B. P. "Numerical receipes in C: The art of scientifical computing" – Second Edition. Cambridge. Cambridge University Press. 1992.

[12] CELLIER, François; KOFMAN, Ernesto. "Continuous system simulation". New York: Springer Science+Business Media Inc, 2006.

[13] ZAMBONI, Lincoln C; PAMBOUKIAN, Sergio V. D.; BARROS, Edson A. R. "Algoritimo de Runge-Kutta: Desenvolvendo classes para o reuso na engenharia". São Paulo: ICECE, 2007.

[14]SCHILDT, H.; GUNTLE, G. "Borland® C++ Builder™ – Referência Completa". Editora Campus, Rio de Janeiro, 2001.

[15] REZNIK, Leonid. "Fuzzy controllers". Oxford: Newnes, 1997.

[16] ASTROM, K.J.; FURUTA, K. "Swinging Up a Pendulum by Energy Control". Paper from 13th IFAC World Congress, San Francisco, CA, 1996.

APÊNDICE A

A PROGRAMAÇÃO ORIENTADA A OBJETOS NO MATLAB®.

COMO IMPLEMENTAR A PROGRAMAÇÃO ORIENTADA A OBJETOS NO MATLAB®.

Para que se possam implementar programas orientados a objetos no MatLab® e até mesmo utilizar estes recursos nas aplicações em Simulink®, é necessário que sua programação seja feita de acordo com uma determinada seqüência de passos. Antes, porém, alguns conceitos, particulares ao MatLab®, precisam ser conhecidos.

(1) – O MatLab® reconhece como uma classe, a partir da qual se criam os objetos, os subdiretorios (pastas) que sejam precedidos pelo caractere @ (“at” ou “arroba”), que estejam dentro da pasta de trabalho(work), ou subpastas, ou qualquer outra pasta que possa ser mapeada pelo sistema do MatLab® como diretório corrente de trabalho.

Por exemplo: a pasta C:/MatLab/work/@ClasseExemplo

(2)- Todas as funções do MatLab® (“M-functions”) que forem colocadas na pasta que define a classe, são consideradas métodos desta classe e podem ser acessadas por intermédio da referência ao objeto criado com a chamada do método(função) desejado(a).

(3)- Por definição do paradígma da orientação a objetos, toda classe deve ter um “criador”, que é um método que tem o mesmo nome da classe dentro da própria classe. Logo, para o caso da classe “@ClasseExemplo”, citada em (1), deve-se ter uma M-function chamada ClasseExemplo.m presente na pasta: C:/MatLab/work/@ClasseExemplo.

(4)- Os atributos da classe (e consequentemente de cada objeto criado para esta classe) são definidos dentro do criador da classe.

(5)- Como, no MatLab®, a definição do tipo da variável ocorre no ato da sua criação, ou seja, não se declara o tipo da variável primeiro para depois atribuir-lhe valores, como ocorre em outras linguagens, a exemplo C. No MatLab®, atribui-se à variável, o valor já formatado, como por exemplo, um vetor, uma matriz, uma string de caracteres ou até mesmo um objeto. O criador da classe é responsável por fazer este papel (devido ao que está definido em (4)) e, se este criador portar parâmetros, pode-se inicializar os atributos do objeto criado.