IDENTIFICAÇÃO E CONTROLE DE SISTEMA PENDULAR INVERSO SOBRE CARRO UTILIZANDO REDES NEURAIS ARTIFICIAIS

ADRIANA DE CARVALHO DRUMMOND

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

IDENTIFICAÇÃO E CONTROLE DE SISTEMA PENDULAR INVERSO SOBRE CARRO UTILIZANDO REDES NEURAIS ARTIFICIAIS

ADRIANA DE CARVALHO DRUMMOND

ORIENTADOR: ADOLFO BAUCHSPIESS

PUBLICAÇÃO: 113/99

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

BRASÍLIA/DF: DEZEMBRO/1999

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

IDENTIFICAÇÃO E CONTROLE DE SISTEMA PENDULAR INVERSO SOBRE CARRO UTILIZANDO REDES NEURAIS ARTIFICIAIS

ADRIANA DE CARVALHO DRUMMOND

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA.

APROVADA POR:

ADOLFO BAUCHSPIESS, Dr. - Ing., UnB
(ORIENTADOR)

CAIRO LÚCIO NASCIMENTO JÚNIOR, PhD, ITA/SP
(EXAMINADOR EXTERNO)

FRANCISCO ASSIS DE OLIVEIRA NASCIMENTO, DOUTOR, UnB
(EXAMINADOR INTERNO)

BRASÍLIA, 23 DE DEZEMBRO DE 1999.

FICHA CATALOGRÁFICA

DRUMMOND, ADRIANA DE CARVALHO

Identificação e Controle de Sistema Pendular Inverso sobre Carro utilizando Redes Neurais Artificiais [Distrito Federal] 1999.

xii, 152 p., 297 mm (ENE/FT/UnB, Mestre, Automação e Controle, 1999).

Dissertação de Mestrado – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Pêndulo Inverso sobre Carro 2. Identificação

3. Neurocontrole

I. ENE/FT/UnB II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

DRUMMOND, A.C. (1999). Identificação e Controle de Sistema Pendular Inverso sobre Carro utilizando Redes Neurais Artificiais. Dissertação de Mestrado, Publicação 113/99, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 152 p.

CESSÃO DE DIREITOS:

NOME DO AUTOR: Adriana de Carvalho Drummond

TÍTULO DA DISSERTAÇÃO DE MESTRADO: Identificação e Controle de Sistema Pendular Inverso sobre Carro utilizando Redes Neurais Artificiais.

GRAU/ANO: Mestre/1999.

Adriana de Carvalho Drummond

QRSW 08, Bl B5, apt 104, Setor Sudoeste

CEP: 70.675-825 – Brasília/DF - Brasil

DEDICATÓRIA

Para minha mãe, Maria da Paz e meu afilhado , Bernard Luis

AGRADECIMENTOS

À minha mãe, Maria da Paz, por todo o carinho, amor, dedicação, apoio, confiança, compreensão e principalmente por ter me oferecido a chance de ter uma família, de me tornar uma profissional melhor e de ser uma pessoa realizada;

À minha avó, Dalva, por todos os cuidados e dedicação;

À minha irmã, Raquel, pelo seus comentários e críticas, e

À minha irmã, Chris, por sempre estar comigo em espírito;

Ao meu padrinho, Antonio Norival, por seu entusiasmo e vibração;

Ao meu orientador Adolfo Bauchspiess pela paciência, dedicação, disponibilidade, compreensão e amizade que foram dispensados durante todo curso de mestrado, e por me ajudar a ter uma vida profissional cheia de triunfos;

Aos meus amigos antigos e aos meus novos amigos pela sua amizade que foi muito importante durante todo este período ;

À minha amiga Janaína pelo apoio, compreensão e carinho;

Aos professores e aos funcionários do Departamento de Engenharia Elétrica;

e a Deus e a Nossa Senhora da Glória por terem me dado a proteção e a força necessárias para a conclusão desta dissertação

RESUMO

Este trabalho foi desenvolvido com as finalidades de identificar e controlar um sistema de pêndulo inverso sobre carro, através da utilização de redes neurais artificiais. Para a identificação foi utilizado o método da identificação não-paramétrica e uma rede neural *Perceptron* Multicamadas com treinamento do tipo *Backpropagation*. Para o controle do sistema pendular inverso, foi desenvolvido um controlador neural, onde este é uma rede neural do mesmo tipo anterior e com o mesmo treinamento. O objetivo deste controlador é permitir que o sistema siga uma trajetória arbitrada, mantendo o equilíbrio da haste em torno da posição de referência, ou seja, deve impedir que a haste do pêndulo caia para a sua posição de mínima energia. Para tanto, foram descritas, de forma sucinta, a teoria de redes neurais e suas utilizações em controle e em identificação, além das equações dinâmicas do pêndulo inverso sobre carro.

Os resultados da identificação e do neurocontrolador foram realmente satisfatórios. O neurocontrolador foi capaz de estabilizar o sistema, além de permitir que o carro seguisse uma trajetória definida. A rede neural de identificação conseguiu simular o comportamento do sistema pendular dentro de um pequeno intervalo de tempo. Vários testes foram realizados para corroborar e validar as redes neurais obtidas durante este projeto. Como resultados paralelos foram desenvolvidas várias rotinas (no ambiente MATLAB) para a determinação das matrizes de observabilidade e controlabilidade, determinação da estabilidade e identificação de sistemas.

ABSTRACT

This work was developed to identify and control an inverted pendulum system with cart using artificial neural networks. For identification a non parametric method and a Perceptron Multilayer neural network with *Backpropagtion* training rule were applied. For control, a neural controller was developed, using the same network architecture and training rule. The neurocontroller makes the system follow an arbitrated trajectory keeping the pole in equilibrium close to the reference position. For a best understanding, the neural network theory and its utilization in control and in identification were described. Dynamics equations were described too.

The identification's results and neurocontroller's results were very satisfactory. The neurocontroller can stabilize the systems and make the cart follow a defined trajectory. The identification's neural network succeeded to simulate the behavior of the pendulous systems in a small time interval. Many tests were realized to corroborate and to validate the neural networks obtained in this project. Furthermore, many routines (in MATLAB) were developed to determination of the observability and controllability matrices, stability and system's identification.

ÍNDICE

LISTA DE FIGURAS

Introdução

1 – INTRODUÇÃO

Quando se olha ao redor utilizando-se um caráter mais científico, nota-se que o ser humano está cercado por inúmeros sistemas. Esses sistemas podem ser representados por satélites altamente avançados utilizados para telecomunicações, ou até mesmo pelo pêndulo do relógio que está pendurado na parede de sua casa. Mas não apenas artefatos construídos pelo homem podem ser caracterizados como sistemas. O movimento das ondas do mar e o bater de asas de uma abelha são também sistemas. Estes podem ser estudados, tendo seus comportamentos modelados e até mesmo podendo ter sua operação simulada através de computadores.

Porém, em geral, não há problemas de funcionamento no bater de asas de um inseto ou no movimento das ondas que precise ser corrigido para que este possa apresentar um desempenho mais satisfatório. Mas quando se trata de máquinas construídas pelo homem, estas podem apresentar particularidades no seu comportamento que não são desejáveis para a função para o qual foram concebidas. Se este tipo de problema existe, tem-se, então, um problema de controle para ser resolvido.

Normalmente, problemas de controle consistem no projeto de sistemas denominados ***controladores***, que são acoplados aos sistemas em estudo, para que estes melhorem seu desempenho durante a operação. O tipo de controlador mais utilizado nas indústrias é o denominado PID. Este controlador é o exemplo típico da teoria de controle clássico. Esta teoria é baseada no ***Teorema da Superposição***, no qual supõe que o modelo do sistema que se deseja controlar é linear. Porém, nem sempre esta suposição é válida, fazendo com que o controlador não tenha um desempenho razoável durante todo o intervalo de operação do sistema.

Introdução

Os sistemas reais, na sua maioria, não são sistemas lineares, o que limita a ação da teoria de controle clássica. Mas, então, como projetar um controlador que tenha um desempenho satisfatório em todo o seu intervalo de operação?

Nas últimas décadas, os cientistas se voltaram novamente para um campo da inteligência artificial que estava um pouco esquecido, as ***Redes Neurais Artificiais***. As redes neurais são um método matemático para identificação de sistemas, classificação de padrões, aproximação de funções baseado no aprendizado de determinadas características, que são apresentadas a rede durante uma fase do seu projeto denominada treinamento.

As redes neurais apresentam diversas arquiteturas que são definidas principalmente pelo tipo de algoritmo utilizado no treinamento e no tipo de funções de ativação utilizada por suas unidades básicas, denominadas *neurônios*. A arquitetura de redes neurais mais utilizada é a denominada ***Perceptron Multicamadas***..

Devido a esta propriedade de 'aprender' as características do comportamento de um determinado sistema durante todo o seu intervalo de operação, por que não utilizar as redes neurais para substituir o controlador clássico, já que este não apresenta o desempenho desejado em todo este intervalo?

Os cientistas ao responderem esta pergunta desenvolveram uma nova área dentro da teoria de controle denominada ***Neurocontrole***. São controladores que utilizam redes neurais para determinar os seus parâmetros (por exemplo, a constante proporcional de um controlador PID) ou, até mesmo, para os substituírem na malha de controle.

Neste trabalho, o sistema estudado não possui um comportamento linear e apresenta características muito particulares. O sistema é o ***Pêndulo Inverso sobre Carro*** e sua característica mais marcante é a instabilidade que apresenta durante o seu funcionamento. Portanto, caso um controlador clássico fosse projetado e acoplado a este sistema, ele não apresentaria um aquém do desejado, pois além do pêndulo inverso sobre carro não ser linear, ele também não é estável.

A solução proposta por este trabalho é o projeto de um neurocontrolador para este sistema pendular. Para tanto, se tornou necessário também a identificação do sistema, pois o seu modelo devido a sua característica instável não pôde ser utilizado diretamente para a determinação do neurocontrolador. Dessa forma, foram utilizados duas redes neurais para o projeto de controle, uma para identificação e outra para o controlador, propriamente dito

Portanto, a finalidade deste trabalho é a identificação e o controle do pêndulo inverso sobre carro baseados na utilização de redes neurais artificiais. Para que todos os passos do projeto ficassem claros, este relatório foi dividido em capítulos que discutem os seguintes assuntos:

- ***Capítulo 2*** – é apresentado o pêndulo inverso sobre carro, seus modelos linear e não-linear são obtidos e são analisadas algumas de suas características como: estabilidade, observabilidade e controlabilidade;
- ***Capítulo 3*** - são discutidas as características e propriedades das redes neurais artificiais. É feita uma comparação entre redes neurais artificiais e biológicas, e são mostrados alguns algoritmos de treinamento para a *Perceptron* Multicamadas, além da rede RBF, que foram os tipos de redes utilizados neste trabalho;
- ***Capítulo 4*** – são mostrados alguns tipos de identificação e de controladores utilizando redes neurais artificiais;
- ***Capítulo 5*** – são descritos os procedimentos para a realização da identificação e para o projeto do neurocontrolador do pêndulo inverso sobre carro.
- ***Capítulo 6*** – são mostrados os resultados obtidos com a identificação e com o neurocontrolador durante as simulações;
- ***Conclusão***

Pêndulo Inverso sobre Carro

2.1- INTRODUÇÃO:

O sistema de ***Pêndulo Inverso sobre Carro*** é muito utilizado academicamente, pois suas características são muito particulares e interessantes para o estudo da viabilidade e robustez de um determinado aparato de controle que seja a ele acoplado.

Por suas propriedades serem muito particulares, muitos ambientes de trabalho e de programação o utilizam como exemplo. O *MATLAB* ® e o LabVIEW® são alguns destes ambientes, que apresentam o pêndulo inverso com o seu sistema de controle em seus arquivos de demonstração.

O pêndulo inverso com carro é um sistema SIMO (single input – multiple output), ou seja, possui apenas uma entrada, mas múltiplas saídas que serão definidas no próximo item. É um sistema tipicamente instável; será mostrado que este não é estável em nenhuma das quatro definições de estabilidade. Porém, é um sistema controlável e observável.

OBS: Todas as definições e teoremas necessários para estas análises serão dados oportunamente durante o capítulo.

2.2- SISTEMAS LINEARES X SISTEMAS NÃO-LINEARES:

Para que os modelos linear e não-linear do pêndulo inverso com carro sejam entendidos perfeitamente, será feita uma pequena explanação sobre a definição de um sistema linear e de um sistema não-linear. Além disso, serão citadas as principais diferenças entre a utilização de um ou de outro.

A definição clássica de um sistema linear é quando a este se aplica o ***Princípio da Superposição***. Este princípio estabelece que a resposta de um sistema linear a duas ou mais entradas simultaneamente é igual a soma das respostas individuais [1]. Essa é uma maneira muito mais simples para se tratar sistemas complexos, pois é possível montar soluções complicadas através de soluções de equações diferenciais simples.

O Princípio da Superposição pode ser representado pela seguinte expressão matemática [2]:

$$H(\alpha_1 \underline{u}_1 + \alpha_2 \underline{u}_2) = \alpha_1 H(\underline{u}_1) + \alpha_2 H(\underline{u}_2) \tag{2.1}$$

onde: α_1 e α_2 são números reais;

u_1 e u_2 são entradas ; e

H(.) é uma função ou operador que represente o sistema em estudo.

A superposição é equivalente a

1) *Aditividade*:

$$H(\underline{u}_1 + \underline{u}_2) = H(\underline{u}_1) + H(\underline{u}_2) \tag{2.2}$$

2)*Homogeneidade*:

$$H(\alpha \underline{u}) = \alpha H(\underline{u}) \tag{2.3}$$

Abaixo está a representação matemática no espaço de estados de um sistema linear e invariante no tempo[3]:

$$\begin{aligned} \dot{\underline{x}} &= A\underline{x} + B\underline{u} \\ \underline{y} &= C\underline{x} + D\underline{u} \end{aligned} \tag{2.4}$$

onde: A- matriz do sistema;

B- matriz de entrada;

C- matriz de saída;

D- matriz de transmissão direta;

$\underline{u}$ - vetor de entrada;

$\underline{y}$ - vetor de saída;

$\underline{x}$ – vetor de variáveis de estado.

Para representar o sistema através da descrição entrada-saída, no caso SISO (função de transferência, para o caso linear) é necessário a determinação da função de transferência,

$$G(s) = C(sI - A)^{-1}B + D = \frac{b_o s^m + b_1 s^{m-1} + \ldots + b_{m-1} s + b_m}{a_o s^n + a_1 s^{n-1} + \ldots + a_{n-1} s + a_n} \tag{2.5}$$

onde, n indica o número de pólos e m indica o número de zeros do sistema.

Porém na natureza, a maioria dos sistemas não obedece o princípio da superposição, são os denominados ***sistemas não-lineares***. Se for realizada uma análise cuidadosa, até mesmo os sistemas representados por equações lineares, apenas apresentam essa "característica de linearidade" em torno de alguns pontos de operação. A desvantagem de se trabalhar com plantas não-lineares é que estas não apresentam um método simples para o projeto de controladores.

2.3- MODELOS NÃO-LINEAR E LINEAR DO PÊNDULO INVERSO COM CARRO:

Neste tópico serão obtidos e discutido os modelos não-linear e linear do pêndulo inverso com carro. Primeiramente, será descrito o modelo não-linear e então realizada uma simulação do sistema para a visualização do funcionamento do mesmo. A partir destas equações dinâmicas não-lineares será realizada uma linearização para se obter um modelo linear do sistema, que terá, dessa forma, suas características de controlabilidade, observabilidade e estabilidade analisadas.

A figura abaixo representa o pêndulo inverso com carro,

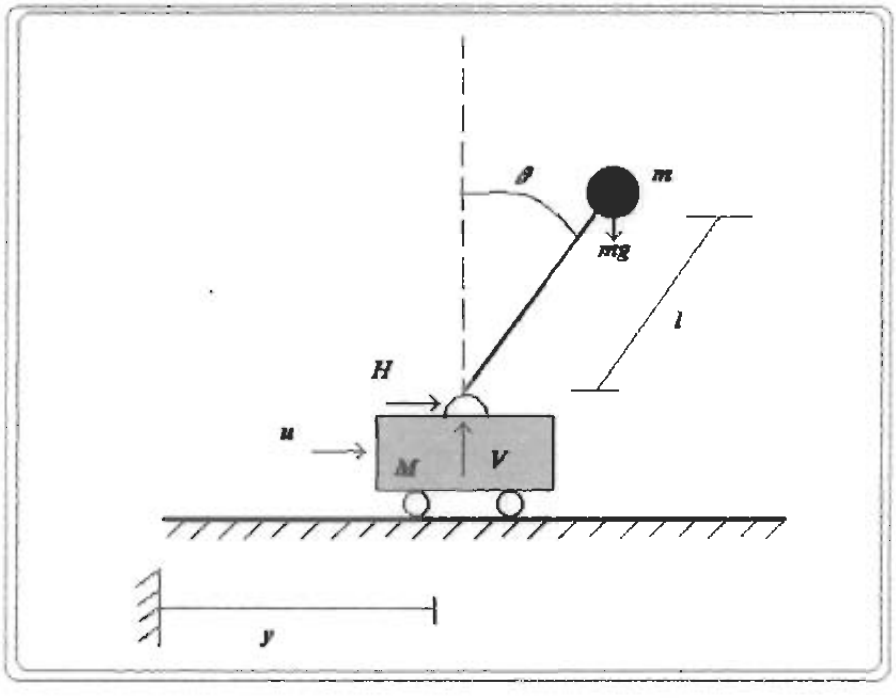


Fig 2.2- Esquema do Pêndulo Inverso com Carro (Chen, 1984)

Para a obtenção das equações dinâmicas do sistema serão aplicadas as Leis de Newton. Baseada na Lei de Newton para o movimento linear obtêm-se[4],[13]:

$$M\frac{d^2y}{dt^2} = u - H$$

$$H = m\frac{d^2}{dt^2}(y + l\,\mathrm{sen}\,\theta) = m\ddot{y} + ml\cos\theta\ddot{\theta} - ml\,\mathrm{sen}\,\theta(\dot{\theta})^2 \qquad (2.7)$$

$$mg - V = m\frac{d^2}{dt^2}(l\cos\theta) = ml\left[-\,\mathrm{sen}\,\theta\ddot{\theta} - \cos\theta(\dot{\theta})^2\right]$$

onde: M – massa do carro (Kg);

m – massa do pêndulo (Kg);

θ - ângulo da haste com relação a posição de referência (rad);

y – deslocamento horizontal do carro (m);

u – força aplicada diretamente ao carro (N), sinal de entrada do sistema;

H – força na horizontal resultante da força produzida pela massa do pêndulo e pelo próprio deslocamento do carro na horizontal (N); e

V – força na vertical resultante da força produzida pela massa do pêndulo (N).

OBS: Nestas equações o atrito foi desprezado.

Utilizando a Lei de Newton para o movimento rotacional tem-se a seguinte expressão:

$$ml^2\ddot{\theta} = mgl\,\mathrm{sen}\,\theta + Vl\,\mathrm{sen}\,\theta - Hl\cos\theta \qquad (2.8)$$

A partir destas expressões, realizando as devidas substituições, são obtidas as seguintes equações dinâmicas para o pêndulo inverso:

$$\ddot{y} = \frac{1}{(M+m)}\left[u - ml\cos\theta\ddot{\theta} + ml\,\mathrm{sen}\,\theta(\dot{\theta})^2\right]$$

$$\ddot{\theta} = \frac{1}{2l\cos^2\theta}\left[2g\,\mathrm{sen}\,\theta + l\,\mathrm{sen}\,2\theta(\dot{\theta})^2 - \cos\theta\ddot{y}\right] \qquad (2.9)$$

A não-linearidade destas expressões são representadas, por exemplo, pelos termos com senos, cossenos e quadráticos.

Para uma visualização do funcionamento do sistema pendular, foi realizada uma simulação destas equações não-lineares utilizando um diagrama de blocos implementado no SIMULINK no ambiente MATLAB®. O diagrama está mostrado a seguir juntamente com as duas variáveis de saída do sistema. Como já foi mencionado, o pêndulo inverso com carro é um sistema classificado como SIMO. As suas duas saídas são:

- Deslocamento – translação no sentido horizontal (variável y);
- Ângulo do pêndulo- movimentação rotacional em torno de um eixo na vertical (variável θ).

O sinal de entrada aplicado foi o degrau unitário, que representava a aplicação de uma força de 1N diretamente no carro do pêndulo inverso.

Para que esta simulação fosse realizada foi necessário atribuir valores de massa do carro, massa do pêndulo, comprimento da haste e valor da gravidade. Um protótipo do pêndulo inverso com carro teve sua construção iniciada e estes valores foram retirados deste sistema real.

M (massa do carro)- 0,012 Kg
m(massa do pêndulo)- 0,014 Kg
l (comprimento da haste)- 0,2m
g(valor da gravidade)- 9,8 m/s^2

Uma observação que deve ser feita é que neste modelo não-linear do pêndulo foi considerado que a haste não tem peso, sendo toda a massa da parte pendular concentrada na ponta da mesma (vide fig 2.1).

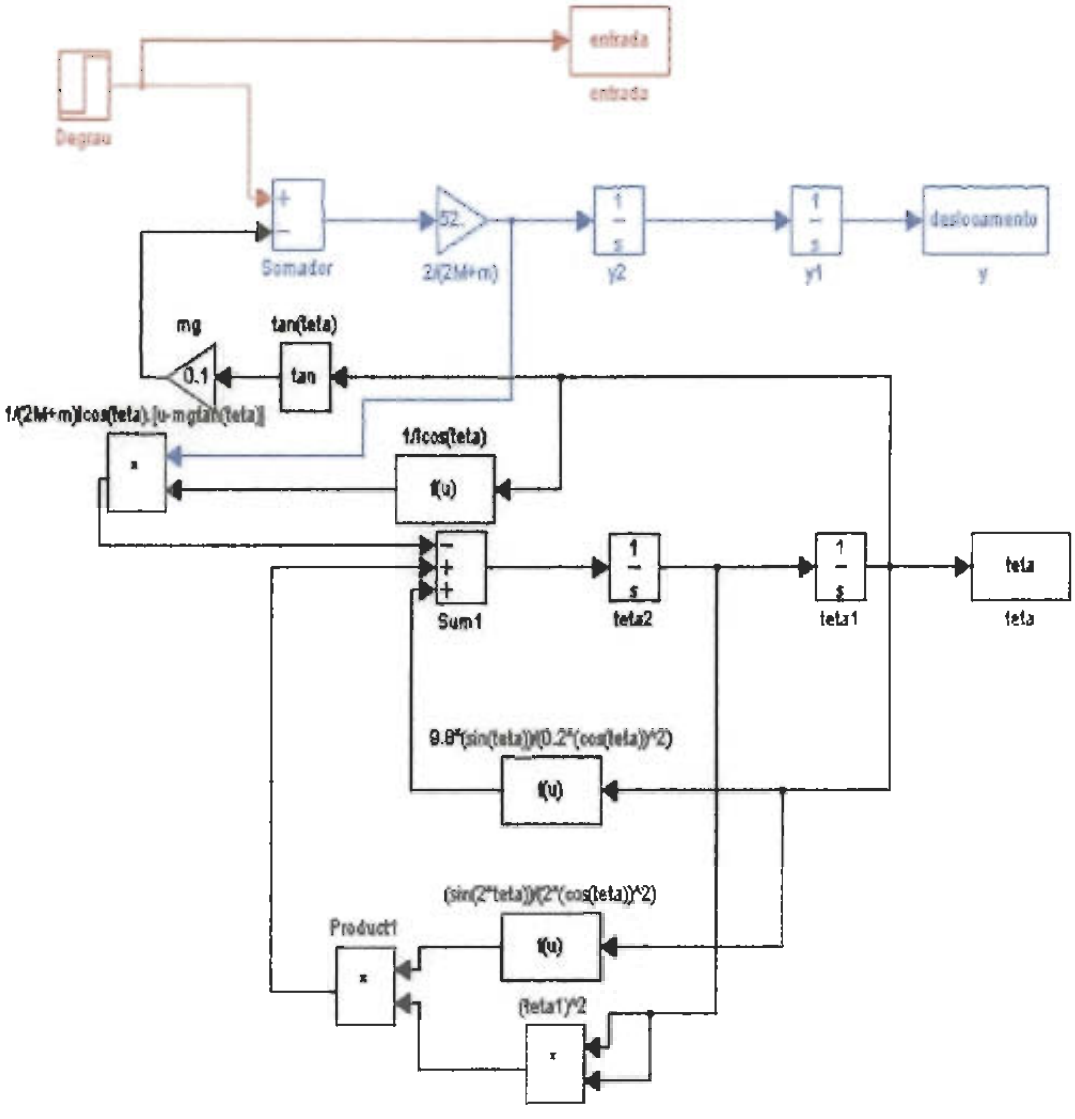


Fig 2.3- Diagrama de blocos do modelo não-linear do pêndulo inverso sobre carro

O tempo de simulação foi muito pequeno devido a existência de singularidade no modelo não-linear do sistema pendular. O método de integração utilizado foi o Dormand-Prince e o passo de integração era variável.

O comportamento das variáveis de saída está mostrado nos gráficos a seguir:

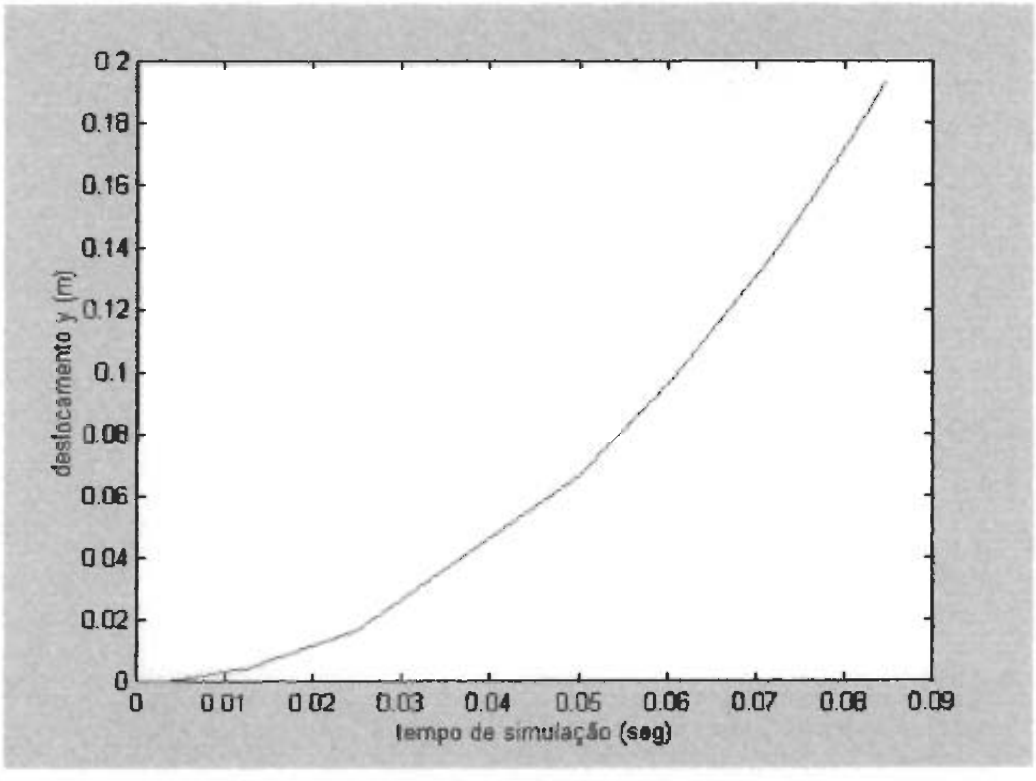


Fig 2.4 – Gráfico da resposta da variável de deslocamento a uma força de 1N aplicada ao carro do pêndulo

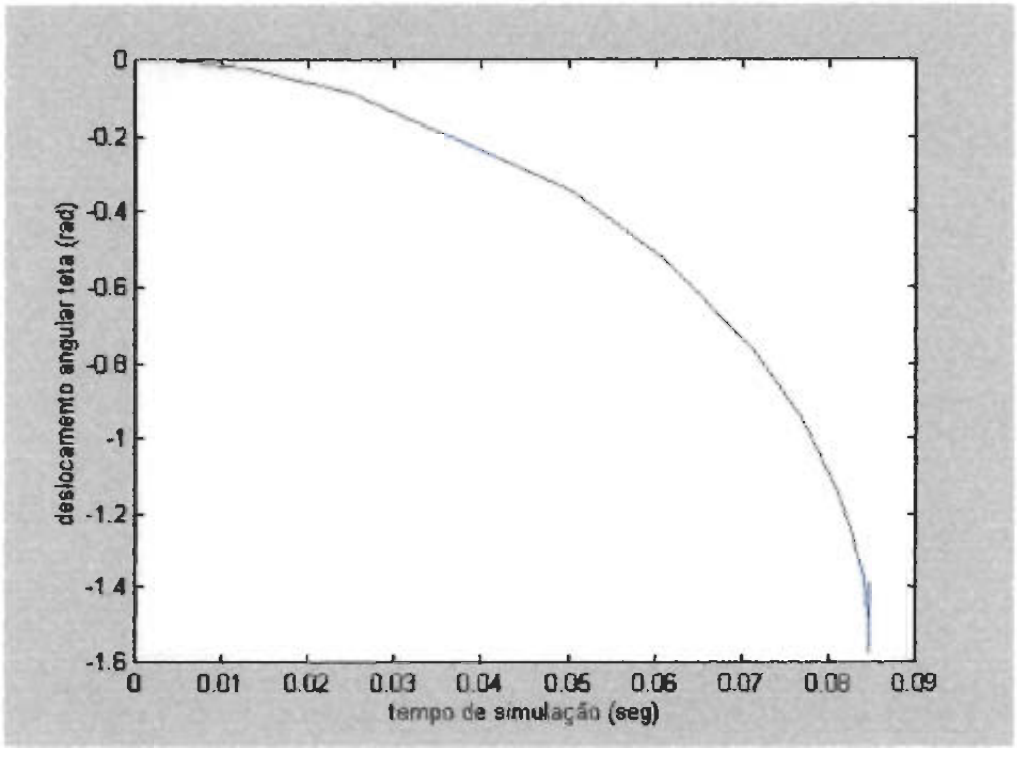


Fig 2.5- Gráfico da resposta da variável do deslocamento angular a uma força de 1N aplicada ao carro do pêndulo inverso

O que foi mostrado até este momento foi o modelo não-linear do sistema em malha aberta em estudo, sem a utilização de nenhum método de controle. Porém, para a análise do sistema, neste trabalho, foi realizada uma linearização do modelo obtido.

O problema de controle do pêndulo inverso é mantê-lo na posição vertical, portanto é razoável supor que o ângulo de deslocamento θ e a velocidade angular sejam muito pequenos, se estes são considerados em radianos. Baseado nesta suposição, têm-se os seguintes resultados [4]:

$$\text{Sen}\,\theta \cong \theta \quad \text{e} \quad \text{Cos}\,\theta \cong 1$$

E por estes termos com deslocamentos angulares θ ao quadrado serem valores muito pequenos em radianos, podem ser desconsiderados. Dessa forma, obtém-se as seguintes expressões:

$$\begin{aligned} M\ddot{y} &= u - m\ddot{y} - ml\ddot{\theta} \\ ml^2\ddot{\theta} &= mgl\theta + mgl\theta - (m\ddot{y} + ml\ddot{\theta})l \end{aligned} \tag{2.10}$$

Utilizando estas expressões, obtêm-se as seguintes equações dinâmicas para o modelo linearizado do pêndulo inverso:

$$\begin{aligned} (M+m)\ddot{y} + ml\ddot{\theta} &= u \\ 2l\ddot{\theta} - 2g\theta + \ddot{y} &= 0 \end{aligned} \tag{2.11}$$

Com as equações linearizadas é possível obter as funções de transferência das duas saídas com o sinal de entrada. Aplicando a transformada de Laplace nas expressões, obtêm-se:

$$\begin{aligned} (M+m)s^2Y(s) + mls^2\Theta(s) &= U(s) \\ (2ls^2 - 2g)\Theta(s) + s^2Y(s) &= 0 \end{aligned} \tag{2.12}$$

Portanto, as duas funções de transferência obtidas são:

$$\begin{aligned} G_{yu}(s) &= \frac{Y(s)}{U(s)} = \frac{2ls^2 - 2g}{s^2\left[(2M+m)ls^2 - 2g(M+m)\right]} \\ G_{\theta u}(s) &= \frac{\Theta(s)}{U(s)} = \frac{-1}{(2M+m)ls^2 - 2g(M+m)} \end{aligned} \tag{2.13}$$

Outra maneira de se representar este sistema é no espaço de estados. Esta será a representação utilizada durante todo este trabalho, pois através desta representação é possível ter acesso ao funcionamento de cada parte do modelo, facilitando o projeto do controlador.

Definindo as variáveis de estado como:

$$\begin{aligned} x_1 &= y \\ x_2 &= \dot{y} \\ x_3 &= \theta \\ x_4 &= \dot{\theta} \end{aligned}$$

Rearrumando as equações (2.11) têm-se, [4]:

$$\begin{aligned} \ddot{y} &= -\frac{2gm}{2M+m}\theta + \frac{2}{2M+m}u \\ \ddot{\theta} &= \frac{2g(M+m)}{(2M+m)l}\theta - \frac{1}{(2M+m)l}u \end{aligned} \tag{2.14}$$

Através destas equações e das definições das variáveis de estado é obtida a seguinte representação no espaço de estados:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{bmatrix}=\begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & \frac{-2mg}{2M+m} & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & \frac{2g(M+m)}{(2M+m)l} & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix}+\begin{bmatrix}0\\ \frac{2}{2M+m}\\ 0\\ \frac{-1}{(2M+m)l}\end{bmatrix}u \qquad (2.15)$$

$$y=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}x$$

Substituindo os valores numéricos na expressão matricial mostrada acima, obtêm-se:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{bmatrix}=\begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & -7{,}22 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 67{,}05 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix}+\begin{bmatrix}0\\ 52{,}63\\ 0\\ -131{,}58\end{bmatrix}u \qquad (2.16)$$

$$y=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}x$$

Baseado nestes valores, também foi realizada uma simulação para a visualização do comportamento do sistema linearizado e para a comparação entre o funcionamento dos dois modelos do pêndulo inverso. Abaixo está esquematizado o diagrama de blocos da planta, que também foi implementado no SIMULINK, como o anterior. O método de integração foi o Dormand-Prince, com passo de iteração variável.

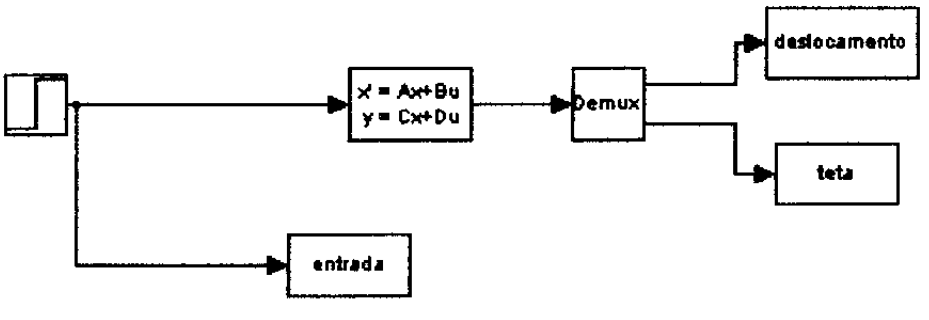


Fig 2.6- Diagrama de blocos do modelo linearizado do pêndulo inverso com carro

A resposta das duas saídas do sistema quando um sinal degrau unitário é colocado na entrada é mostrada nos gráficos a seguir.

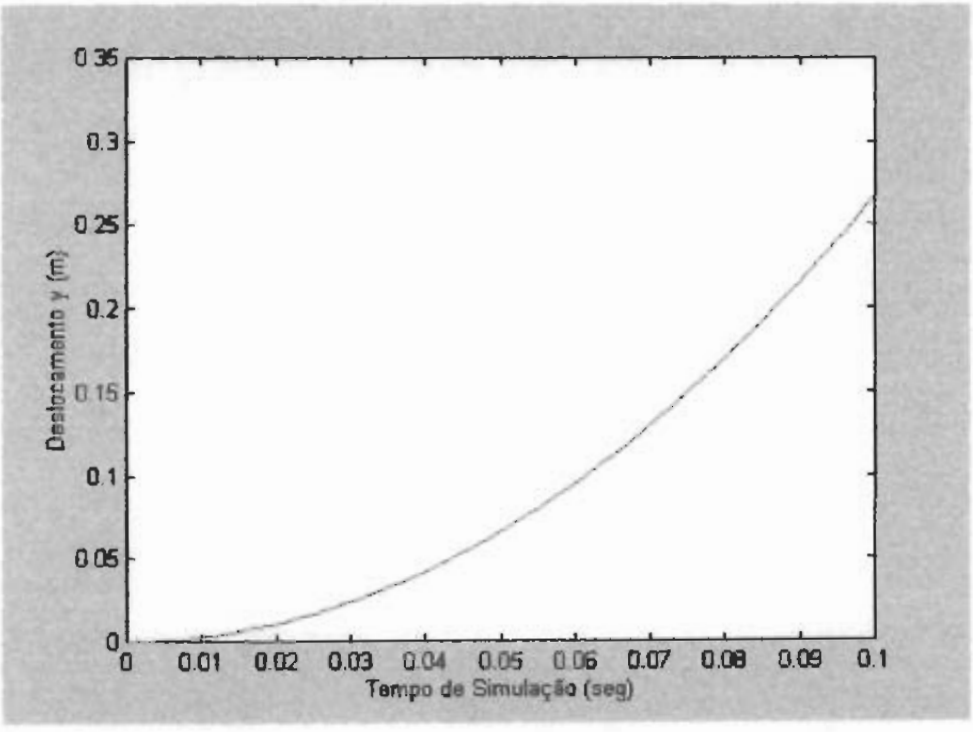


Fig 2.7- Gráfico da resposta da variável de deslocamento y a aplicação da força de 1N

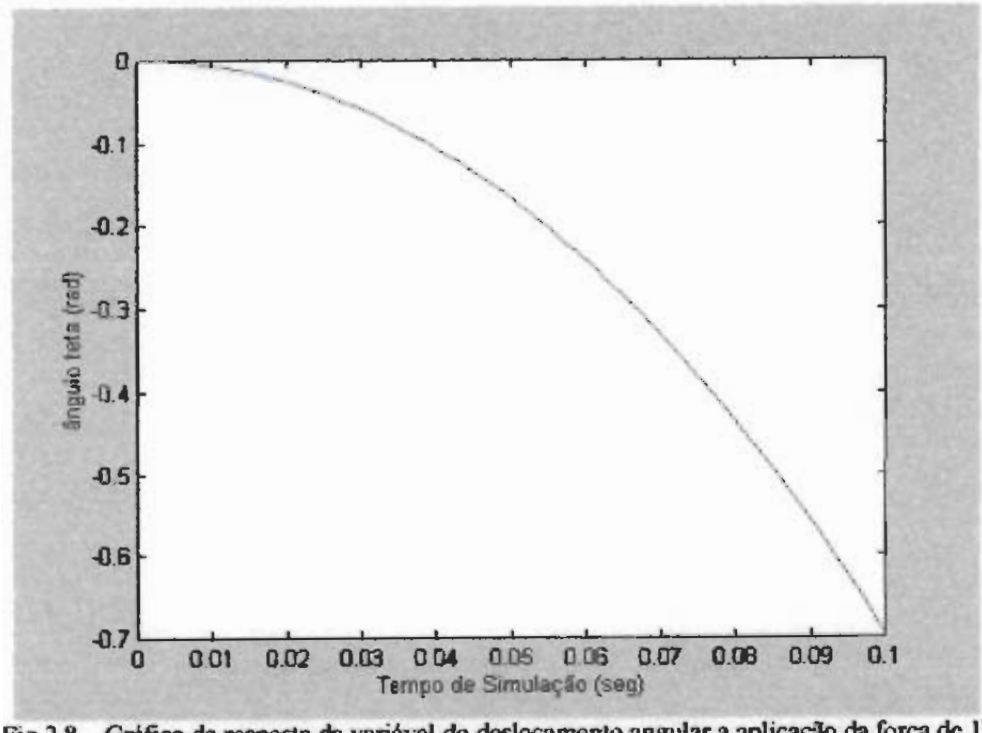


Fig 2.8 – Gráfico da resposta da variável do deslocamento angular a aplicação da força de 1N

2.4- ANÁLISE DE ESTABILIDADE:

A ***instabilidade*** do pêndulo inverso com carro é uma das suas características mais importantes, e é por sua causa que este sistema se torna um interessante objeto de estudo. Neste tópico serão introduzidas as quatro definições de estabilidade utilizadas no desenvolvimento de plantas de controle. Será utilizado o modelo linearizado do pêndulo, portanto todas as definições considerarão o sistema em questão linear e invariante no tempo.

Primeiramente, serão feitas as explicações teóricas [2] e, então o modelo do pêndulo será utilizado para provar a instabilidade do sistema.

1. ***BIBO Estabilidade:*** Um sistema em repouso é BIBO estável se e somente se para toda entrada limitada, a saída também será limitada. Equivale a verificar se os pólos da função de transferência G(s) têm parte real menor que zero (Re(pólos)<0).

2. ***Estabilidade no Sentido de Lyapunov:*** Um estado de equilíbrio é estável no sentido de Lyapunov se e somente se $\forall\ \varepsilon > 0, \exists\ \delta(\varepsilon) > 0$ tal que

$$\|\underline{x}_o - \underline{x}_e\| \le \delta \Rightarrow \left\|\phi\left(t;t_o,\underline{x}_o,0\right) - \underline{x}_e\right\| \le \varepsilon, \forall t_o, \forall t \ge t_o \qquad (2.17)$$

onde, ϕ é a matriz de transição de estado do sistema.

OBS: Estado de equilíbrio de $\dot{x} = A(t)x$, $\underline{x}_e$ é um estado de equilíbrio se e somente se

$$\phi\left(t;t_o,x_e,0\right) = x_e$$

Para o tipo de sistema adotado neste trabalho, todo estado de equilíbrio de $\dot{x} = A(t)x$ é estável no sentido de Lyapunov se e somente se todos os autovalores de **A** tem parte real negativa ou nula, e aqueles com parte real nula são zeros simples do polinômio mínimo de **A** .

3. Estabilidade Assintótica: Um estado de equilíbrio é assintoticamente estável se for estável no sentido de Lyapunov e se toda a trajetória que se inicia suficientemente próxima a $\underline{x_e}$ converge para $\underline{x}$ quando t→∞;

Para sistemas lineares e invariantes no tempo, o estado zero de $\dot{x} = A(t)x$ é assintoticamente estável se e somente se $\forall$ $\mathrm{Re}(\lambda_i) < 0$, onde λ_i são os autovalores do sistema.

4. Estabilidade Total: Um sistema dinâmico é totalmente estável (T-estável) se e somente se para todo estado inicial e para toda entrada limitada, tanto a saída quanto os estados são limitados.

$$|y(t)| \leq k < \infty$$
$$|x_i(t)| \leq k < \infty$$

A equação dinâmica linear invariante no tempo é T-estável se e somente se as três condições abaixo são satisfeitas:

1. Todos os autovalores de A: $\mathrm{Re}(\lambda_i) \leq 0$;
2. Todos os autovalores de A com $\mathrm{Re}(\lambda_i) = 0$ são zeros simples do polinômio mínimo de A
3. Os modos não amortecidos de A não estão acoplados a nenhum componente da entrada.

Aplicando estas definições ao modelo linear do pêndulo inverso com carro e substituindo os valores numéricos nas suas constantes, têm-se as seguintes funções de transferência:

$$G_{yu}(s) = \frac{131{,}58\left[0{,}4s^2 - 19{,}6\right]}{s^4 - 67{,}05s^2} \tag{2.18}$$

onde seus pólos são: s1= 0;s2 = 0;s3 = 8,1884 e s4 = -8,1884.

$$G_{y\theta}(s) = \frac{-131{,}58}{s^2 - 67{,}05} \tag{2.19}$$

onde seus pólos são: s1 = 8,1884 e s2 = -8,1884.

Com base nos valores dos pólos encontrados para as duas funções de transferência pode-se afirmar que o sistema ***não é BIBO estável*** .

Para a análise dos outros tipos de estabilidade é necessário a determinação dos autovalores do sistema, que são obtidos através da seguinte expressão, denominada polinômio mínimo da matriz A:

$$\det(A - \lambda I) = 0 \tag{2.20}$$

Substituindo a matriz A do modelo linear, tem-se a seguinte expressão matricial:

$$\det(A - \lambda I) = \begin{vmatrix} -\lambda & 1 & 0 & 0 \\ 0 & -\lambda & -7{,}22 & 0 \\ 0 & 0 & -\lambda & 1 \\ 0 & 0 & 67{,}05 & -\lambda \end{vmatrix} = 0 \tag{2.21}$$

Para o cálculo do determinante desta matriz foi utilizado o método de Laplace (cálculo dos cofatores). Será omitido todos os passos para a determinação deste valor, pois este método é muito extenso, porém fácil de ser verificado. Portanto, tem-se:

$$\det(A - \lambda I) = \lambda^4 - 67{,}05\lambda^2 \qquad (2.22)$$

onde as raízes da expressão, ou os autovalores do sistema são: $\lambda 1 = 0, \lambda 2 = 0, \lambda 3 = 8{,}1884$ e $\lambda 4 = -8{,}1884$.

Com base nestes valores e nas definições pode-se afirmar que o sistema ***não é estável no sentido de Lyapunov*** e ***não é assintoticamente estável***. Isto pode ser concluido devido a um autovalor estar no semi-plano direito e existir um autovalor duplo na origem.

Como o sistema não é estável assintoticamente e nem no sentido de Lyapunov, esta condição garante a sua ***não estabilidade total*** (vide definição).

Para que esta análise fosse feita de uma forma mais rápida e pudesse ser ampliada para qualquer outro sistema sem ser necessário a maioria do desenvolvimento matemático, foi realizado um programa executável no ambiente MATLAB® . Algumas das janelas do programa '*estabilidade*' estão mostradas abaixo e seu código está anexado ao apêndice.

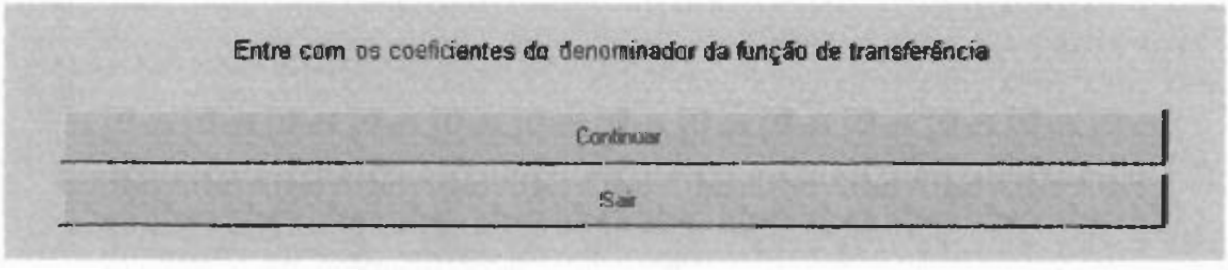


Fig 2.9 – Primeira janela do programa 'estabilidade'

Após entrar com estes coeficientes aparecerá a seguinte janela:

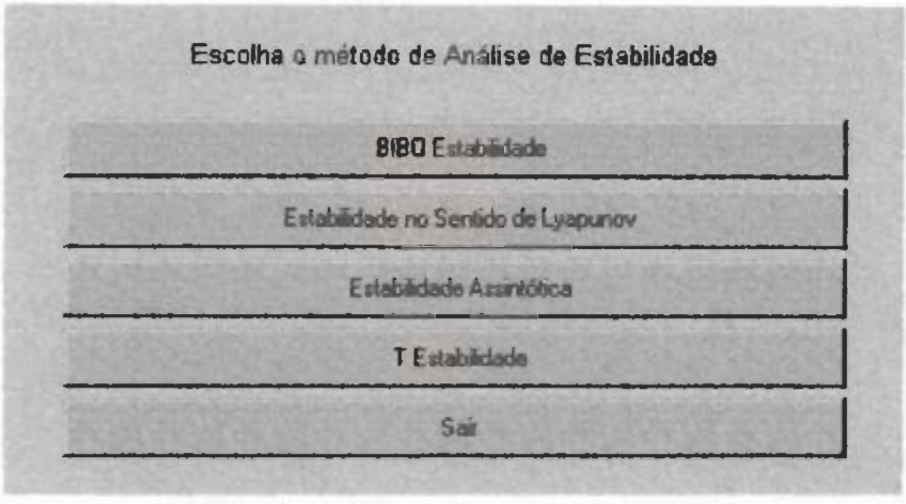


Fig 2.10- Segunda janela do programa 'estabilidade'

Caso fosse escolhido o item BIBO estabilidade, como exemplo, para o pêndulo inverso se teria como última janela do programa:

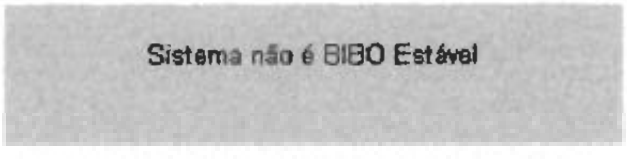


Fig 2.11- Última janela do programa 'estabilidade'

Se fosse escolhido qualquer outro item apareceria uma janela intermediária:

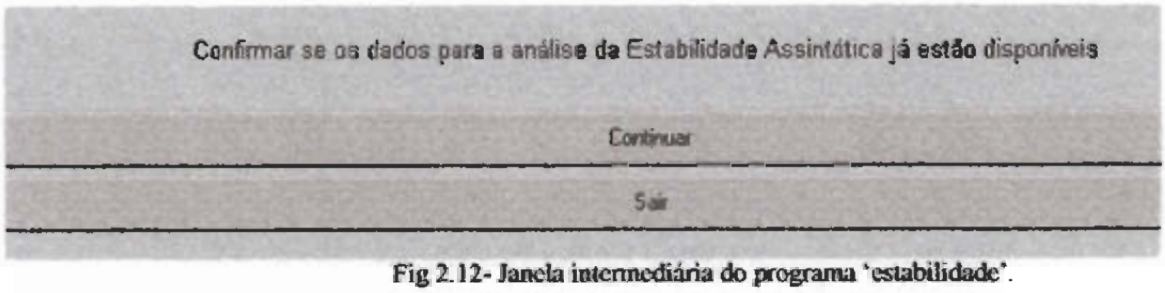


Fig 2.12- Janela intermediária do programa 'estabilidade'.

Caso o item escolhido fosse 'Continuar', seria confirmado se todos os dados necessários para a determinação da estabilidade já se encontravam disponíveis. Se não

estiverem estes valores devem ser definidos para a conclusão da análise de estabilidade Se o item 'Sair' fosse o escolhido, uma tela de finalização seria apresentada:

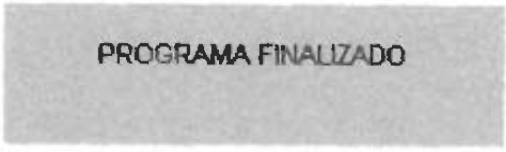


Fig 2.13- Janela de Finalização do programa 'estabilidade'

2.5- OBSERVABILIDADE E CONTROLABILIDADE:

Neste tópico será estudado duas características importantes de um sistema, quando é desejado projetar uma malha de controle, a ***controlabilidade*** e a ***observabilidade***. Primeiramente, estes dois termos serão definidos, dando mais ênfase aos sistemas lineares e invariantes no tempo, já que será utilizado o modelo linear para esta análise. Após serem feitas as definições, estas serão aplicadas ao pêndulo inverso em questão.

1. Controlabilidade: A equação de estado de um sistema é controlável no instante to se para qualquer estado $\underline{x_o} = \underline{x}(t_o)$ no espaço de estados Σ e qualquer estado $\underline{x_1} = \underline{x}(t_1) \in \Sigma$ existe um tempo finito $t_1 > to$ e uma entrada $\underline{u}[t_o, t_1]$ capaz de transferir o estado $\underline{x_o}$ para o $\underline{x_1}$ no instante t_1[2].

Para o caso linear e invariante no tempo, é equivalente afirmar que a matriz n *x* np, mostrada abaixo e denominada *matriz de controlabilidade*, tem posto *n*.

$$\begin{bmatrix} B & AB & \dots & A^{n-1}B \end{bmatrix} \qquad (2.23)$$

2. Observabilidade: A equação dinâmica de um sistema é observável em to se para qualquer estado $\underline{x_o}$ em to do espaço de estados $\exists$ t_1 finito, $t_1 > to$, tal que o conhecimento de $\underline{u}[t_o, t_1]$ e $y[t_o, t_1]$ é suficiente para se determinar $\underline{x_o}$ [2].

Para sistemas lineares e invariantes no tempo, equivale a afirmar que a matriz q x n, mostrada a seguir e denominada *matriz de observabilidade*, tem posto *n*.

$$\begin{bmatrix} C \\ CA \\ \cdots \\ CA^{n-1} \end{bmatrix} \tag{2.24}$$

Para a análise de observabilidade e controlabilidade do modelo linear do pêndulo inverso com carro foi desenvolvido um programa no ambiente MATLAB® que não apenas determina se o sistema desejado é controlável ou observável, mas calcula e disponibiliza as matrizes de observabilidade e controlabilidade na tela. Abaixo está a primeira tela do programa 'analise1', sendo que o seu código está anexado ao apêndice.

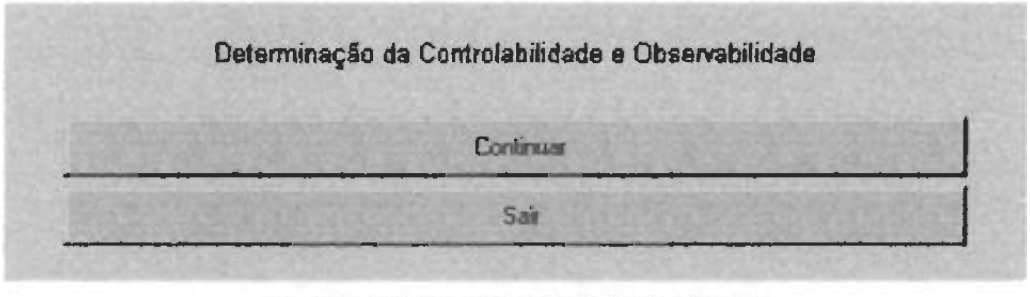


Fig 2.14- Janela inicial do programa 'análise1'

Independente se é desejado conhecer a controlabilidade ou observabilidade, é necessário definir as matrizes A, B, C e D do sistema.

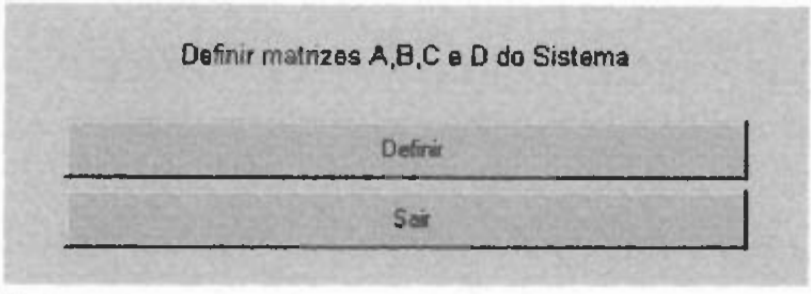


Fig 2.15- Janela onde se define as matrizes do sistema no espaço de estados

Após a entrada dos valores das matrizes, faz-se a escolha sobre qual das duas propriedades se deseja analisar.

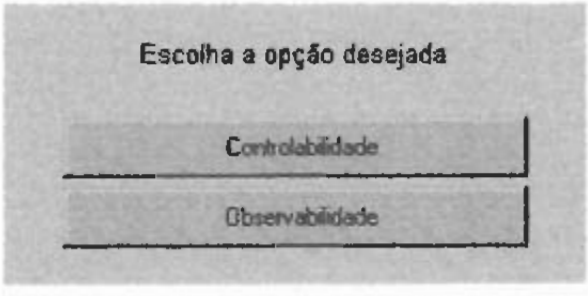


Fig 2.16- Janela onde se escolhe o tipo de análise desejada

- Primeiro Caso: Controlabilidade

É obtida a seguinte resposta para o sistema em estudo:

Matriz de Controlabilidade do sistema

CO1 =

1.0e+003 *

0	0.0526	0	0.9500
0.0526	0	0.9500	0
0	-0.1316	0	-8.8224
-0.1316	0	-8.8224	0

Sistema plenamente controlável

- Segundo Caso: Observabilidade

Para este sistema o programa retorna a resposta mostrada abaixo.

Matriz de Observabilidade do sistema

CO2 =

1.0000	0	0	0
0	0	1.0000	0
0	1.0000	0	0
0	0	0	1.0000
0	0	-7.2200	0
0	0	67.0500	0
0	0	0	-7.2200
0	0	0	67.0500

Sistema plenamente observável

Estas respostas são de fácil verificação, pois é possível observar que o posto das matrizes CO1 e CO2 é 4, o mesmo valor numérico do tamanho da matriz A .

Caso seja escolhido o item 'Sair', aparecerá a janela abaixo:

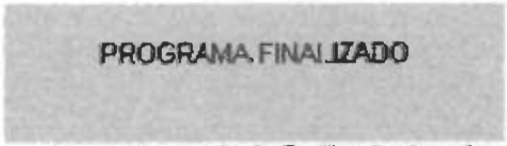


Fig 2.17 – Janela de finalização da rotina.

3 – REDES NEURAIS ARTIFICIAIS

Neste capítulo serão descritos alguns conceitos básicos sobre a técnica de inteligência artificial denominada **Rede Neural Artificial**. Também será feito um breve histórico sobre o surgimento das mesma e a utilização deste método na realização de um controlador que será implementado ao sistema do pêndulo inverso com carro. Este controlador é especificamente denominado de **neurocontrolador** e terá seu funcionamento e topologia explicados no capítulo 4. Portanto, este capítulo será dividido nos seguintes itens:

- Introdução;
- Breve Histórico;
- Neurônio Biológico;
- Redes Neurais Biológicas;
- Neurônio Artificial;
- Redes Neurais Artificiais;
- Redes '*Perceptron*' Multicamadas;
- Redes Neurais de Base Radial.

3.1 – INTRODUÇÃO:

O conceito de **inteligência artificial (IA)** foi primeiramente concebido pelos filósofos a 400 anos AC. Eles imaginavam a mente como uma máquina especial, na qual o conhecimento operava codificado em uma linguagem interna, e que como resultado (produto final) a mente indicava quais as ações corretas a serem tomadas [5].

Mesmo existindo a concepção da idéia a séculos atrás, foi preciso o desenvolvimento de outras áreas do conhecimento para que a inteligência artificial surgisse como assunto de pesquisa reconhecido pela comunidade científica. Foi necessário que matemáticos providenciassem as ferramentas necessárias para o desenvolvimento deste conceito, através do desenvolvimento da teoria da probabilidade e da incerteza, além de proporcionarem a base matemática para a realização dos algoritmos que são o fundamento da IA.

Os pesquisadores amadureceram a idéia de que seres humanos e outros animais podem ser considerados como máquinas de processamento de informação. E finalmente, com o desenvolvimento da engenharia de computação (avanços tanto no *hardware* como *software*) foi possível implementar conceitos da IA

Mesmo sendo o conceito de IA bem antigo, este termo só foi proposto pela primeira vez por John MacCarthy da Universidade de Standford, em 1956.

A IA não teve um desenvolvimento linear, este foi formado por ciclos de sucessos mesclados com ciclos de total ostracismo. Recentemente, o avanço desta área está voltado para a sua aplicação na solução de problemas apresentados por sistemas reais, como por exemplo o controle de um sistema intrinsicamente instável como o pêndulo inverso com carro, que é o objeto deste trabalho.

Dentro deste campo de pesquisa existem diversas linhas de pesquisa e técnicas que estão em estudo. Neste trabalho será tratada apenas uma técnica de inteligência artificial, as **redes neurais**. Nesta linha, é realizada uma analogia entre células nervosas vivas e o processo eletrônico [5]. Sendo que o ponto forte desta técnica é o aprendizado dos sistemas através da apresentação metódica de exemplos que possuem uma grande quantidade de informação.

3.2 – BREVE HISTÓRICO:

Neste item do capítulo será apresentado um breve histórico do desenvolvimento das redes neurais artificiais. Este histórico tem a finalidade de situar o leitor para que este possa observar que o conceito das RNA (redes neurais artificiais) não é muito novo e que seu crescimento como técnica foi lento, ou seja, entre o seu aparecimento e sua aplicação em problemas práticos houve um grande intervalo.

- **Década de 40**: O neurofisiologista McCulloch e o matemático Pitts (ambos da Universidade de Illinois) desenvolveram o conceito das RNAs. Eles fizeram uma analogia entre os neurônios biológicos e o processo eletrônico que estes neurônios realizavam durante o seu funcionamento. O trabalho consistia em um circuito com resistores e amplificadores que representavam as relações existentes entre os neurônios biológicos (estas relações são denominadas conexões sinápticas, e serão tratadas com mais profundidade nos próximos itens). O psicólogo Donald Hebb descobriu a base de aprendizado nas redes

neurais quando explicou o que ocorre a nível celular, durante o processo de aprendizagem do cérebro [6].

- **Década de 50:** Foi realizada em 1956, a 1ª Conferência Internacional de Inteligência Artificial, onde foi apresentado um modelo de RNA por Nathaniel Rochester, ~~um~~ funcionário da IBM. Este modelo era baseado na simulação de diversos neurônios interconectados.

- **Década de 60:** No ano de 1960, Frank Rosenblat [18]mostrou a rede **Perceptron**. Em 1969, foi publicado um livro entitulado ***'PERCEPTRON'*** , cujos autores eram Marvin Minsky e Seymour Papert [19]. Neste livro, eles criticaram muito duramente as pesquisas de redes neurais artificiais. Como eram cientistas renomados, praticamente destruíram todos os investimentos relacionados a esta técnica.

- **1982:** Foi o ano no qual o físico e biólogo John Hopfield (Instituto de Tecnologia da Califórnia) deu uma novo impulso às redes neurais, revisando e contestando as teses matemáticas que Minsky e Papert divulgaram na década de 60. Hopfield apresentou um trabalho no qual elementos de processamento interconectados procuram um estado de energia mínima, além disso também mostrou que a memória do sistema é feita pelas interconecções entre os neurônios [6].

Atualmente as pesquisas com redes neurais artificiais vêm sendo desenvolvidas em diversas universidades e institutos, buscando principalmente aperfeiçoar a sua aplicação em problemas extremamente sofisticados e com alto grau de complexidade.

3.3 – NEURÔNIO BIOLÓGICO:

Para uma melhor compreensão do neurônio artificial (elemento básico das redes neurais artificiais) será feita uma pequena explanação sobre a fisiologia e a morfologia do neurônio biológico, além da explicação sobre o funcionamento do fluxo da informação. A comparação entre estes dois tipos de neurônio deve ser feita posteriormente.

O **neurônio** também denominado **célula nervosa**, como em outras células é delimitada por uma membrana plasmática, que possui funções muito importantes para a transmissão e processamento da informação. Nestas células é possível identificar três partes distintas. Essas partes são listadas e definidas abaixo:

- ***Corpo Celular ou Soma :*** é a porção central do neurônio, contém o núcleo celular e o citoplasma. Como existem vários tipos de neurônio cada um para realizar uma determinada função (exemplo: células piramidais do cerebelo e neurônio motor) o soma varia muito com relação a dimensão, forma e localização da mesma na célula nervosa. Como exemplos podem ser citados neurônios que possuem um soma com diâmetro superior a 0,5 mm e outros que podem apresentar um soma menor que 2 mícrons. Com relação a sua localização na célula, pode-se utilizar os neurônios motores como exemplo. Estes apresentam o corpo celular no seu centro [7].

- ***Axônio:*** é a projeção filamentar que possui um diâmetro relativamente uniforme que pode se estender por distâncias que podem variar de algumas centenas de mícrons até alguns metros. A sua função básica é transmitir informação na forma de pulsos elétricos regenerativos, ou seja, sem acontecer atenuações para todo o sistema nervoso. Os axônios podem se juntar e formar **troncos nervosos** ou **nervos**, sendo que estes podem possuir de uns poucos filamentos à dezena de milhares. Nos vertebrados a velocidade de propagação dos pulsos é aumentada devido a presença de capas de **mielina** que são produzidas por células **neurogliais de Schwamm**, o que significou um grande desenvolvimento na evolução destes animais.

- ***Dendritos:*** são prolongamentos que se organizam em complexas **árvores dendritais**. São a parte receptiva da célula, e devido a presença de muitos filamentos oferece uma enorme área de contato para a recepção da informação.

Os contatos entre os neurônios são feitos através de estruturas denominadas **sinapses**. Estas estruturas são formadas por uma parte da membrana do axônio, denominada **membrana pré-sináptica** (transmissão do impulso nervoso), por uma parte da membrana dos dendritos, denominada **membrana pós-sináptica** (recepção do impulso nervoso) e pelo intervalo existente entre as duas membranas que é chamado de **fenda sináptica**. Essa fenda é vencida pelo pulso elétrico através de mediadores químicos, **neurotransmissores**, que são liberados pela membrana do axônio. Estes mediadores alcançam a membrana dos dendritos provocando uma alteração na sua polarização elétrica. Dessa forma, a informação é propagada em uma só direção [7].

Uma observação que deve ser feita é que dependendo do tipo de neurotransmissor, esta alteração eletroquímica da membrana dendrital pode ser no sentido

de facilitar ou inibir a formação de um potencial de ação, ou seja, permite ou inibe a transmissão da informação para o próximo neurônio

A seguir é mostrada uma figura esquemática da célula nervosa (Fig 3.1) que evidencia cada parte da mesma que foi discutida anteriormente.

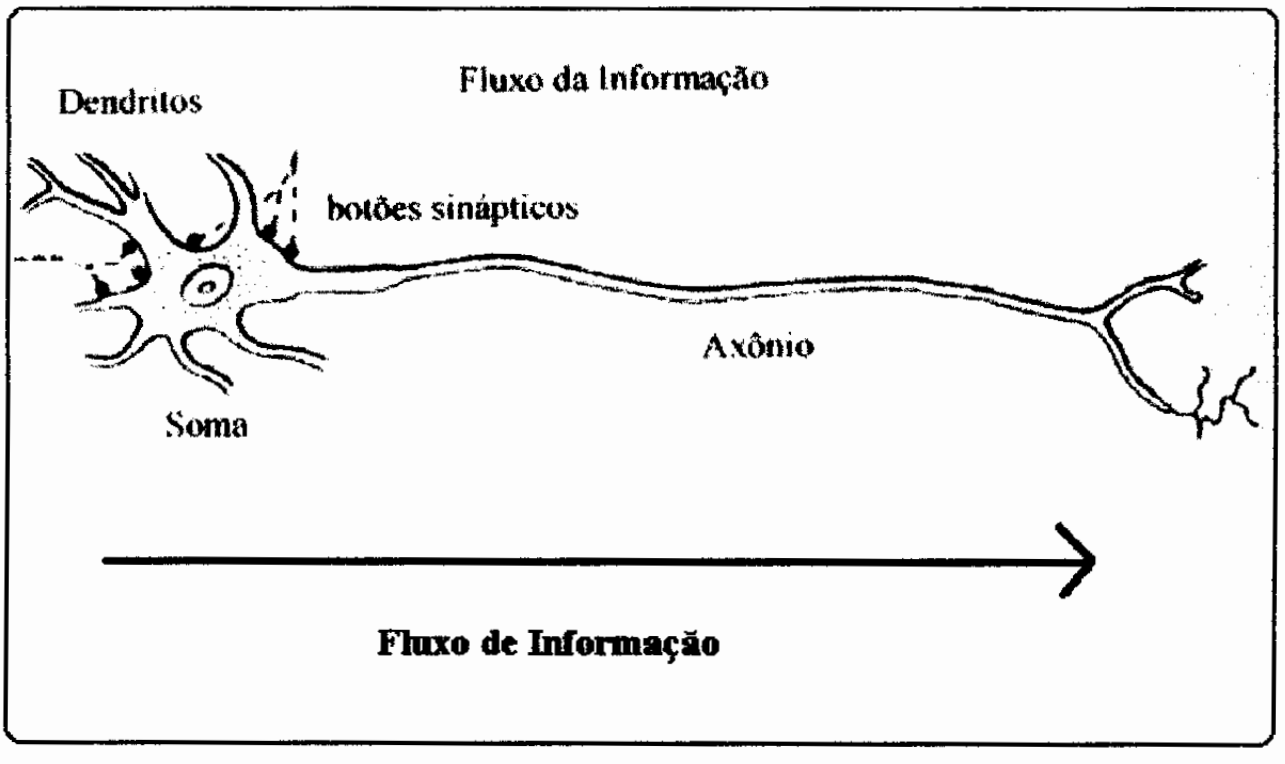


Fig 3.1: Neurônio Biológico -sentido do fluxo de informação (Kovács, 1997)

3.3 – REDE NEURAL BIOLÓGICA:

Quando estas unidades nervosas se unem formam o que chamamos de **rede neural.** As redes neurais formam o sistema nervoso, que na maioria dos mamíferos se divide em sistema nervoso central e sistema nervoso periférico. As redes neurais artificiais são fisiologicamente análogas às redes biológicas que formam o sistema nervoso central.

3.4 – NEURÔNIO ARTIFICIAL:

O neurônio artificial é uma imitação extremamente simplista do já apresentado neurônio biológico. O neurônio artificial têm diversas entradas (dendritos) e um corpo somático, onde ocorre o processamento das informações das entradas (estímulo), além disso apresenta uma única saída (axônio). Pode-se então esquematizar um neurônio simples da seguinte forma (Fig 3.2).

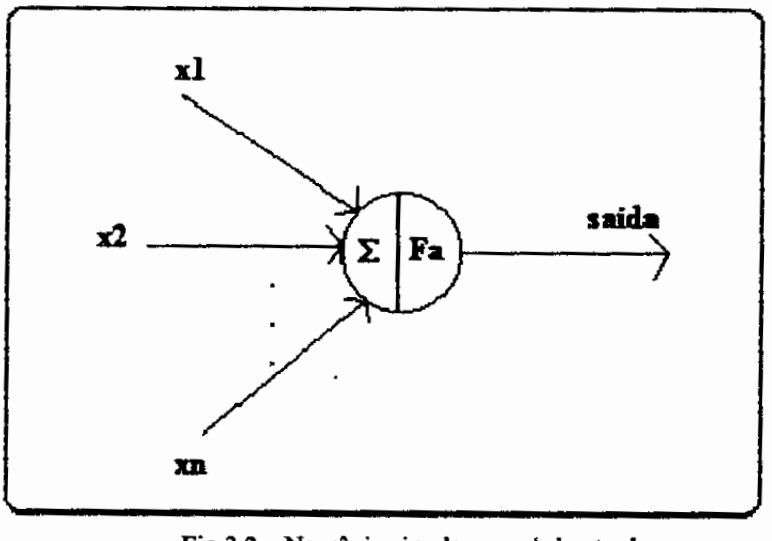


Fig 3.2 – Neurônio simples com 'n' entradas

Existem alguns termos-chaves para o entendimento do funcionamento do neurônio artificial. Estes estão relacionados abaixo:

- Pesos;
- Função de Ativação.

3.4.1 - Pesos:

Os pesos são uma das principais características dos neurônios artificiais. Eles representam o grau de importância de um determinada entrada para um dado neurônio. Dessa forma, durante o aprendizado da rede neural o seu valor representativo pode se alterar. Pode-se então concluir que, similarmente ao neurônio biológico, se uma entrada é bastante utilizada o seu peso correspondente também é estimulado, aumentando sua influência na rede neural.

Os valores de pesos podem ser organizados como um **conjunto de pesos** :

$$(w_0 \; w_1 \; w_2 \; w_3 \ldots\ldots \; w_n) \qquad (3.1)$$

Um ponto que deve ser destacado é que as equações utilizadas para a atualização dos pesos é diferente para cada arquitetura de rede neural artificial.

3.4.2 - Função de Ativação:

Esta função tem a finalidade de repassar o sinal para a saída do neurônio [6]. Pode também ser definido como uma função interna da unidade da rede neural que cria um nível de ativação dentro da mesma unidade. Dessa forma, o neurônio trata a informação conseguida pela soma ponderada conseguida pela atuação dos pesos nos valores de entrada.

A função de ativação pode ter um equacionamento muito simples como ser a própria soma ponderada (realizada pelos pesos) ou possuir um processamento mais complexo.

Esta função também é conhecida como **limiar lógico**. A seguir estão mostradas as formas mais utilizadas na caracterização de um neurônio(Fig 3.3) :

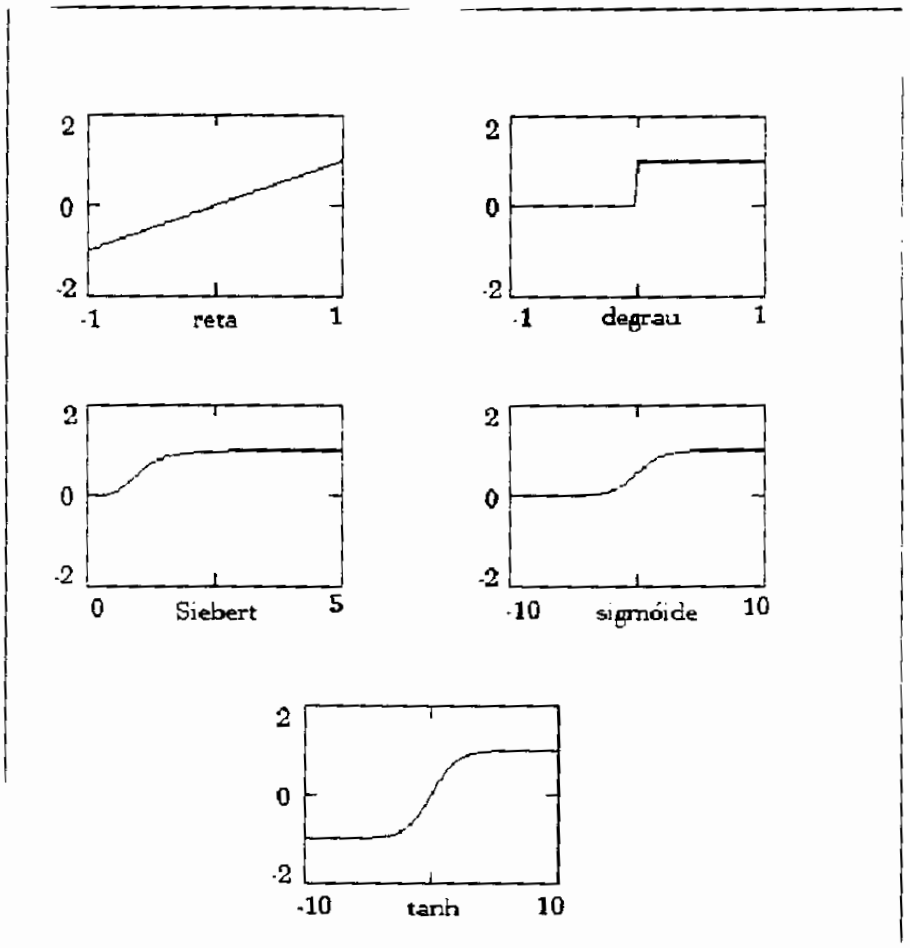


Fig 3.3 – Função de Ativação (Kovács, 1996)

O neurônio artificial completo, com todos os seus atributos e funções pode ser esquematizado da seguinte maneira (Fig 3.4):

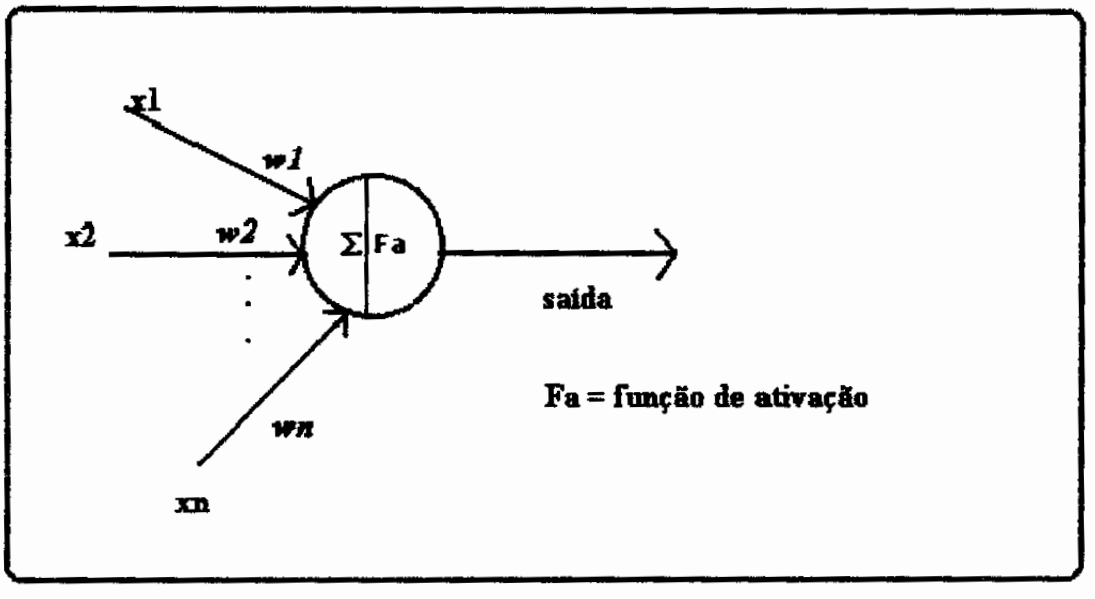


Fig 3.4 – Função de Ativação e Pesos (modificado – Tafner, Xerez e Rodrigues, 1996)

Com base no esquema pode-se escrever uma expressão para a saída do neurônio artificial:

$$saída = Fa(\sum_{i=0}^{n} x_i * w_i) \qquad (3.2)$$

3.6 – REDES NEURAIS ARTIFICIAIS:

Similarmente ao que ocorre aos neurônios biológicos, os neurônios artificiais também se agrupam formando redes. Também se agrupam de diferentes maneiras, formando diferentes arquiteturas, que como na perspectiva biológica (rede neural do córtex, rede neural periférica motora), possuem diferentes aplicações.

As saídas dos neurônios artificiais são conectadas às entradas de outros neurônios simulando uma árvore dendrital extremamente simples, com suas respectivas sinapses. Para que esta comparação fique bem clara, a figura (Fig 3.5) a seguir faz uma analogia entre os dois tipos de rede neurais, a biológica e a artificial .

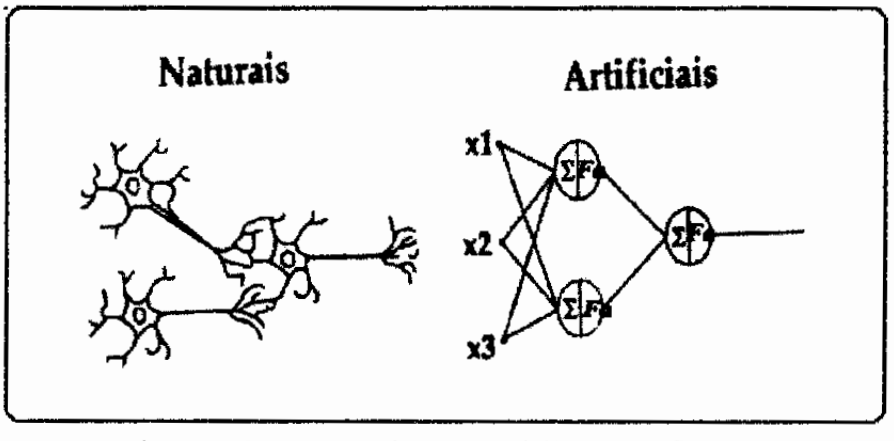


Fig 3.5 – Comparação entre as redes neurais biológica e artificial (modificado – Tafner, Xerez e Rodrigues, 1996)

Os neurônios artificiais se agrupam formando camadas. A **camada de entrada** tem o objetivo de guardar a informação das entradas para ser passada para a próxima camada de neurônios [6], ou seja, a camada de entrada é formada pelas próprias entradas da rede neural.

Dessa forma, o número de neurônios desta camada de entrada é dado pelo

número de entradas que a rede possuirá. Esta camada também pode ser chamada de **ponto de entrada** ou **camada de distribuição**.

Outra camada essencial é a **camada de saída**. Esta camada é formada por neurônios, que possuem a função de devolver ao mundo externo a resposta da rede neural aos estímulos colocados nas suas entrada. O número de neurônios desta camada é dada pelo número de saídas que se deseja como resposta.

As redes neurais também podem apresentar camadas intermediárias entre a camada de entrada e a camada de saída, estas são denominadas **camadas escondidas** ou **camadas ocultas**. São formados por neurônios que possuem a mesma estrutura dos que formam as camadas anteriores. Não existe uma regra que forneça o número de neurônios que compõem esta camada, ou o número de camadas ocultas que uma rede deva apresentar.

A figura (Fig 3.6) a seguir é o esquema de uma rede neural artificial, com a descrição de suas camadas.

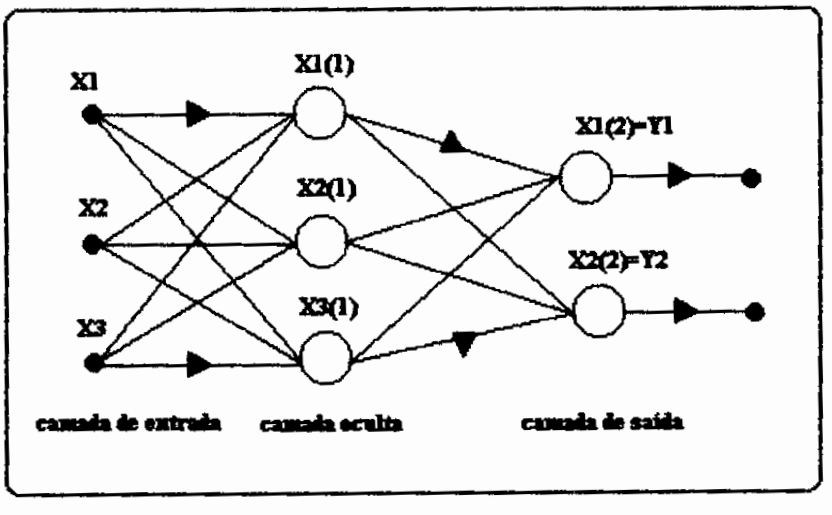


Fig 3.6 – Esquema das camadas de uma rede neural artificial (Loesch e Sari, 1996)

OBS: A partir deste ponto, o neurônio será representado apenas por um círculo vazio, para fim de simplificação dos desenhos.

Redes Neurais Artificiais

As diferentes conexões entre as camadas podem originar diversos tipos de arquiteturas, que são utilizadas para resolver problemas de natureza variada. Neste trabalho só será visto um tipo de conexão entre neurônios, que é a utilizada durante o projeto: as **redes *feedforward***. Na figura (Fig 3.7) a seguir está o esquema desta arquitetura de redes neurais .Uma rede tem essa denominação quando as saídas dos neurônios de uma determinada camada estão totalmente conectados com os neurônios da próxima camada.

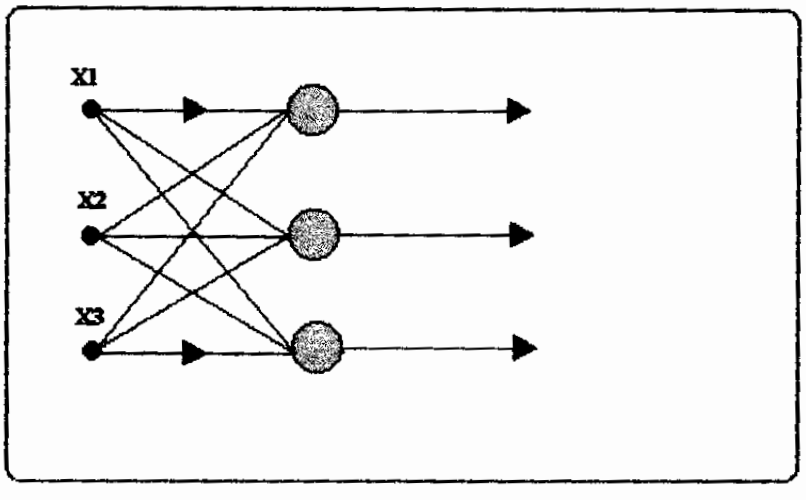


Fig 3.7 – Redes *feedforward* (Tafner, Xerez e Rodrigues, 1996)

O aprendizado das redes neurais artificiais é baseado em um treinamento, que tem como um dos pontos principais a apresentação de um conjunto de entradas e saídas desejadas durante este processo de aprendizagem. O treinamento é iterativo até a saída da rede apresentar valores próximos aos desejados. Quando valores determinados são introduzidos nas suas entradas, este objetivo é alcançado através do ajuste dos pesos relativos às conexões entre os neurônios da rede, a cada apresentação do grupo de entrada- saída. Os passos do treinamento serão descritos a seguir:

Redes Neurais Artificiais

1) Apresentar o conjunto entrada-saída para a rede;
2) Conferir o resultado e ajustar os pesos se necessário;
3) Se terminado o conjunto entrada-saída e a saída da rede não apresentar os valores desejados, deve-se apresentar o conjunto novamente para a mesma, até o objetivo ser atingido. A rede então é considerada treinada.

O aprendizado da rede pode acontecer de duas maneiras:

- Supervisionado – para este treinamento é necessário um treinador, ou seja, alguém que supervisione o treinamento. Neste caso, será apresentado para rede um conjunto de entrada-saída desejado e a cada etapa do treinamento as saídas da rede neural são comparadas com as desejadas, até os dois valores se tornarem suficientemente próximos. Os passos descritos anteriormente são característicos para um treinamento supervisionado.
- Não-Supervisionado – neste treinamento não são necessárias saídas desejada. Apenas os valores de entrada são necessários e estes são processados de acordo com regras internas da própria rede (características dos neurônios e arquitetura das conexões entre eles), portanto não necessitam de um supervisionador. É também denominado de auto-treinamento.

Nos próximos itens serão discutidas as características dos tipos de redes neurais artificiais utilizadas neste projeto, que são as **Redes *Perceptron* Multicamadas** e as **Redes Neurais de Base Radial**.

3.7 – REDES 'PERCEPTRON' MULTICAMADAS:

Na maioria das aplicações de redes neurais artificiais utiliza-se um tipo de rede denominada ***Perceptron Multicamadas***. As redes 'Perceptron' Multicamadas também chamadas de redes 'Feed-forward' são a arquitetura de rede mais difundida e largamente utilizada. São uma extensão da rede '*Perceptron*', e portanto heteroassociativas, com a adição de uma ou mais camadas ocultas, uma função de transferência diferente, e possuem uma maior capacidade de generalização. Além disso, essa rede possui robustez, ou seja, é imune a pequenas falhas, é possível retirar neurônios da rede sem que esta apresente uma queda no seu desempenho

Seu treinamento é baseado nas seguintes expressões para atualizações dos pesos (este treinamento é conhecido como *Backpropagation*, pois o erro entre os valores desejados de saída e os obtidos são propagados para o início da rede, já que estes influenciam na próxima etapa do treinamento) [10],[11]:

$$\begin{aligned} s_j^{(k)} &= w_{0j}^{(k)} + \sum_{i=1}^{N_x} w_{ij}^{(k)} . x_i^{(k-1)} \\ x_j^{(k)} &= f(s_j^{(k)}) \end{aligned} \tag{3.3}$$

onde:

f é uma função contínua diferenciável.

x = saída do neurônio do elemento i da camada k

s = soma ponderada dos pesos pelas entradas na saída do módulo somador e entrada ao módulo de ativação.

w = pesos das conexões sinápticas na entrada do elemento j da camada k, onde i é o índice da conexão.

As equações mostradas, anteriormente, representam o comportamento do processamento para frente da informação durante o treinamento. A primeira equação expressa a soma ponderada dos pesos pelas entradas na saída do módulo somador e esta saída é inserida na entrada do módulo da função de ativação. A segunda expressão indica exatamente a saída do módulo da função de ativação, que também representa a saída do neurônio da camada *k* em questão. Portanto, estas duas equações expressam a saída de um determinado neurônio *i* da camada *k*.

$$\begin{aligned} \varepsilon_j^{(k)} &= d_j - y_j \\ \delta_j^{(k)} &= \varepsilon_j^{(k)} . f'(s_j^{(k)}) \\ w_j^{(k)}(n+1) &= w_j^{(k)}(n) + 2\mu\delta_j^{(k)}(n).x_j^{(k)}(n) \end{aligned} \tag{3.4}$$

onde:

ε = erro na saída associado ao neurônio da camada de saída;

δ = erro derivativo quadrático na camada de saída;

μ = taxa de aprendizado da rede neural;

d = saída desejada;

y = saída obtida pela rede neural;

n = iteração corrente.

As três expressões escritas representam o fluxo para trás da informação. Após a saída ser obtida, esta é comparada com a saída desejada (primeira equação 3.4). Caso este erro não esteja de acordo com a precisão desejada, os pesos devem ser ajustados. Este é o aprendizado propriamente dito. O ajuste é proporcional ao gradiente, seguindo o fator de proporcionalidade denominado *taxa de aprendizado*.

A segunda expressão representa o erro derivativo quadrático, que é o valor retropropagado pela rede neural, e necessário para o cálculo do ajuste de pesos, representada pela terceira equação. Dessa forma, após a propagação dos erros derivativos quadráticos, o vetor de

pesos é atualizado e outro par de treinamento é utilizado na entrada da rede neural.

Este treinamento foi inventado e popularizado por Rummelhart, Hinton e Williams. Este algoritmo de treinamento resolveu limitações fundamentais que as primeiras redes neurais complexas (função de ativação não-linear e múltiplas camadas) apresentavam ao serem treinadas.

A rede 'Perceptron' Multicamadas tem a seguinte representação esquemática (Fig. 3.8):

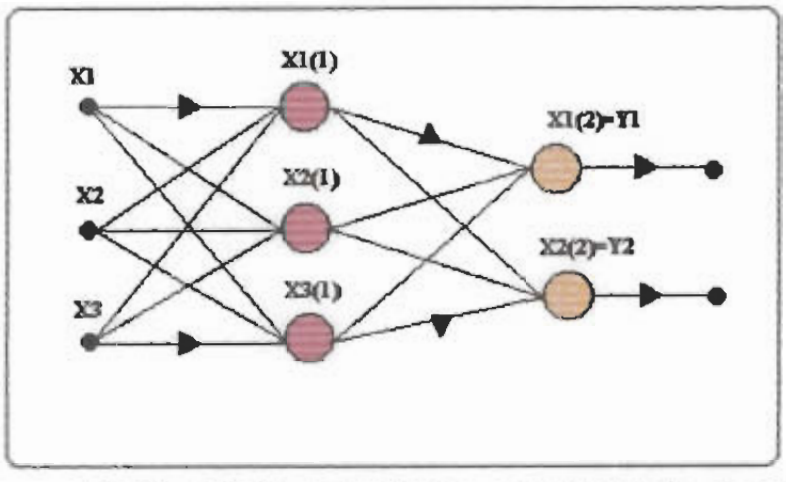


Fig 3.8 – Rede '*Perceptron*' Multicamadas (modificado – Bauchspiess, Drummond e Romariz, 1997)

O treinamento desta arquitetura de rede é dividida em dois passos para cada interação ou **época**, ou seja, para cada ciclo de aprendizagem. Um ciclo de aprendizagem é caracterizado pela apresentação do conjunto de treinamento, ajustamento dos pesos das conexões e cálculo do erro entre a saída conseguida no final do ciclo e a saída desejada.

1) passo *'forward'*: quando as entradas são apresentadas e propagadas através das camadas intermediárias até a camada de saída;
2) passo *'backward'*: quando um erro é calculado com base nos valores das saídas e 'retropropagado' para calcular as mudanças dos pesos.

Um parâmetro muito importante no treinamento '*Backpropagation*' é a taxa de aprendizado μ . Caso esta taxa seja arbitrada com um valor muito alto (exemplo 0,85), os valores dos pesos oscilam com amplitudes muito altas, porém se esta tiver um valor muito baixo, a convergência se torna muito lenta [10]. A escolha do valor ótimo da taxa de aprendizagem pode evitar alguns problemas no treinamento, como os citados anteriormente, uma das formas sugeridas para o cálculo deste parâmetro é a seguinte expressão [10]:

$$\mu = \frac{1.5}{(sqrt(\sum p_i^2))} \tag{3.5}$$

onde : p_i é o número de todos os exemplos que pertencem a um conjunto de treinamento.

A principal inconveniência que pode ocorrer durante o treinamento, e que caracteriza um caso comum, é quando a rede encontra um mínimo local. Isto acontece porque a superfície de erros de uma rede complexa é altamente convoluída, apresentando vales e dobras. Portanto durante o desenvolvimento do algoritmo, este pode ser apanhado em alguns desses mínimos locais. A situação é mostrada na figura (Fig 3.9) a seguir.

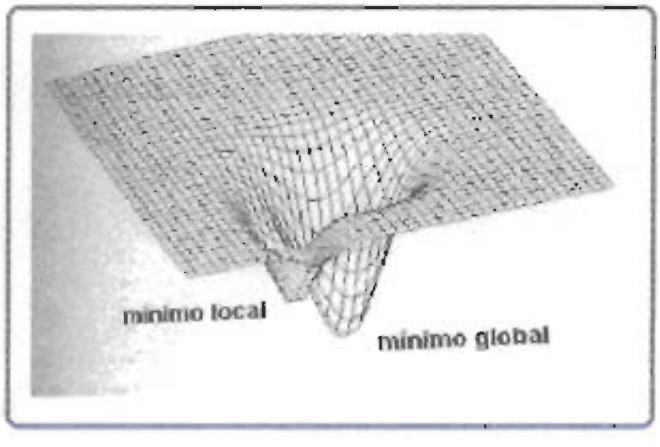


Fig 3.9 – Representação de mínimo local e mínimo global (Kovács, 1996)

Outro problema ocorre quando os pesos tornam-se ajustados a magnitudes muito grandes, forçando muitos dos neurônios a operarem com altos valores e derivadas muito baixas. Como a atualização de pesos depende das derivadas, o processo pode se tornar virtualmente paralisado.

Uma observação a ser feita é a inicialização dos pesos no início do treinamento. Em geral, os pesos são inicializados com valores aleatórios próximos de zero. No caso do ambiente MATLAB, os pesos são inicializados com base nos valores de entrada do conjunto de treinamento, além de considerar o número de neurônios e as funções de ativação das camadas oculta e de saída.

3.8 – REDES NEURAIS DE BASE RADIAL (RBF):

A arquitetura da rede neural de base radial, é mostrada de forma esquematizada na figura abaixo (Fig 3.10):

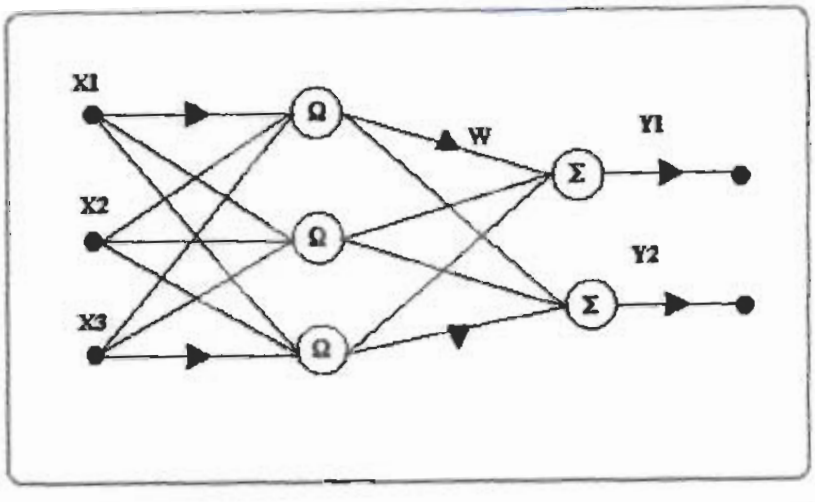


Figura 3.10: Rede neural de Base Radial (modificado – Drummond, Bauchspiess e Oliveira, 1999)

A rede RBF consiste de três camadas, sendo a primeira formada pelas n entradas, que estão completamente conectadas aos neurônios da segunda camada. A terceira camada é a camada de saída. Esta última camada é formada por nós caracterizados por um somatório simples com uma função de ativação linear [12][13].

A segunda camada tem como ativação uma função de base radial (gaussiana), sendo portanto esta radialmente simétrica.

$$f(x) = \exp[-(x - K_e)^2 / 2\sigma^2] \tag{3.6}$$

onde K_e e σ são parâmetros que estão relacionados ao valor médio e ao espalhamento (abertura) da gaussiana.

O treinamento desta rede pode ser feito em dois passos. O primeiro é o ajuste de funções de base radial por agrupamento estatístico, ou seja nesta fase é determinado o espalhamento σ da função gaussiana [15].

$$\sigma_i = ((\sum_{p=1,m} abs(c_i - c_{ip}))/m)^{1/2} \tag{3.7}$$

onde c_{ip} é o centro do p-ésimo agrupamento próximo ao agrupamento i.[13][14]

Este primeiro passo é a fase de treinamento não-supervisionado. O segundo passo consiste em aplicar o algoritmo de regressão linear para o ajuste dos pesos das conexões da camada de saída. Esta é a fase de treinamento supervisionado.

Com a RBF tem-se, em geral, um treinamento mais rápido e uma melhor generalização quando comparado ao '*Perceptron*' multicamadas, além de não apresentar o característico problema com mínimos locais que pode surgir com as MLP's. Uma outra grande vantagem é a possibilidade de se realizar o treinamento incremental desta rede, ou seja, podem-se utilizar as informações já assimiladas pela rede em estágios anteriores em um novo treinamento.

Identificação e Neurocontrole

4 – IDENTIFICAÇÃO E NEUROCONTROLE

Como já foi discutido, a grande maioria dos sistemas que estão disponíveis na natureza são sistemas não-lineares, que por não satisfazerem o teorema da superposição, possuem soluções para suas equações dinâmicas (modelo não-linear do sistema real) altamente complexas. Quando o projeto de um controlador clássico é realizado, é suposto que a planta que se deseja controlar é linear. Esta suposição ocasiona um comportamento nem sempre satisfatório do controlador (por exemplo um PID – proporcional-integral-derivativo), pois esta hipótese apenas se aplica quando se considera uma referência ou uma perturbação de pequena amplitude introduzida no sistema modelado.

As redes neurais artificiais são uma alternativa atraente para a resolução de problemas de controle, pois o projetista do controlador não teria que ficar restringido a suposição de que o modelo do sistema, em questão, seja linear. O controle utilizando redes neurais artificiais é denominado ***neurocontrole***. Este tipo de controle pode resolver diversos problemas como: projeto de controladores robustos, determinação de ganhos ótimos para o controlador, estimação de parâmetros fisicamente expressivos para o modelo, dentre outros.

Porém, um passo fundamental para o projeto de um neurocontrolador é ter um conhecimento adequado do sistema que se deseja controlar. Para isso ocorrer, é necessário que se realize um modelamento correto ou então uma ***identificação*** do sistema. Uma forma para a realização desse processo de identificação é a utilização de redes neurais artificiais. Mas por que realizar uma identificação do comportamento do sistema, se o modelo deste for conhecido?

Isso é necessário quando se deseja determinar os valores dos parâmetros, ou características de um modelo que dependem de alguma não-linearidade na informação obtida através de experimentos. Portanto, mesmo conhecendo o modelo do processo pode-se, por exemplo, estimar alguns de seus parâmetros através de informações obtidas do sinal de entrada e de saída. Dessa forma, existem diversos tipos de identificação que são úteis ao estudo do comportamento de um sistema.

Neste capítulo serão estudados os seguintes itens, para que os conceitos de neurocontrole e identificação fiquem claros, quando forem aplicados ao pêndulo inverso sobre carro:

- Introdução;
- Controle Convencional *x* Neurocontrole;
- Identificação de Sistemas por Redes Neurais Artificiais;
- Neurocontroladores.

4.1 – INTRODUÇÃO:

Neste capítulo serão discutidas as formas para modelar um processo e algumas maneiras para implementar a leis de controle diretamente, através da utilização de redes neurais artificiais. Com estas aplicações poderá ser verificada a diversidade de usos deste procedimento de controle inteligente.

Para começar a descrever as formas de identificação e controle através das redes neurais artificiais, é interessante ter noções sobre os conceitos de identificação e de controle que serão necessários neste trabalho. A seguir, será feito um pequeno resumo sobre estes conceitos.

- ***Identificação:*** processo de obtenção de um modelo de um processo matemático através de informações experimentais sobre dados de entrada e saída. A seguir está mostrado um esquema do processo.

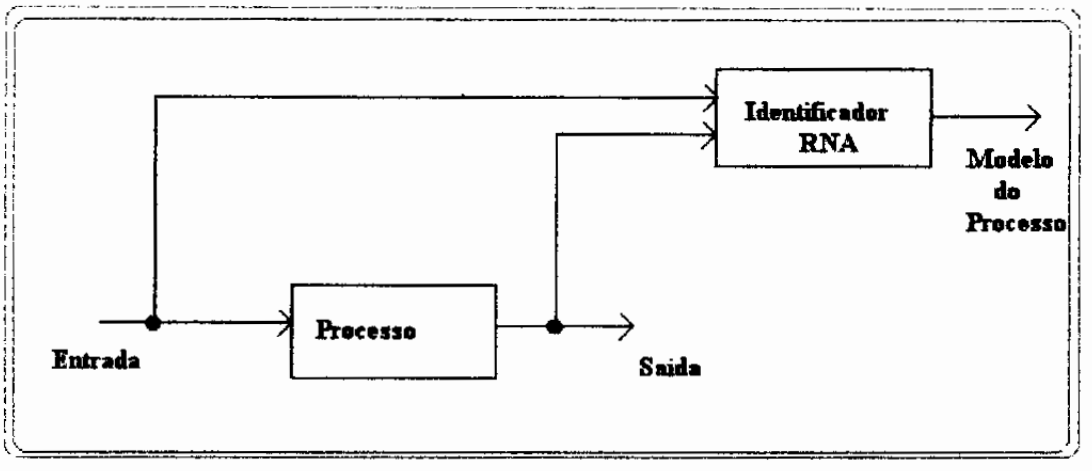


Fig 4.1 – Esquema de identificação de um processo (Gupta e Sinha, 1996)

Para a discussão sobre controle será feita uma divisão entre *controle regulatório* e *controle de supervisão*, para um entendimento mais conciso dos conceitos.

- ***Controle Regulatório:*** é um mecanismo de controle baseado na medida de sensores durante o funcionamento do sistema. Além disso, este tipo de controle emprega um algoritmo de realimentação para que haja um ajuste do processo [3], controle do tipo ***feedback***. O tipo de controlador mais utilizado neste caso é o PID (proporcional-integral-derivativo) A seguir está o esquema de um controlador regulatório.

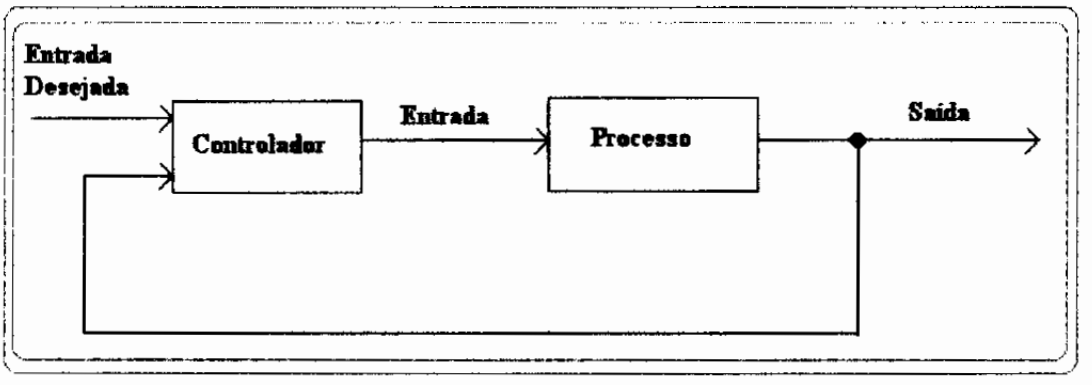


Fig 4.2- Esquema para o controle regulatório de um processo (Gupta e Sinha, 1996)

- ***Controle de Supervisão:*** tem a função de auto-sintonizar ou ajustar a malha de controle (ou seja, o controlador). Este controle ajuda a garantir que a estratégia de controle é ótima para as condições de funcionamento do processo [3]. A seguir está mostrado um esquema da arquitetura de controle de supervisão.

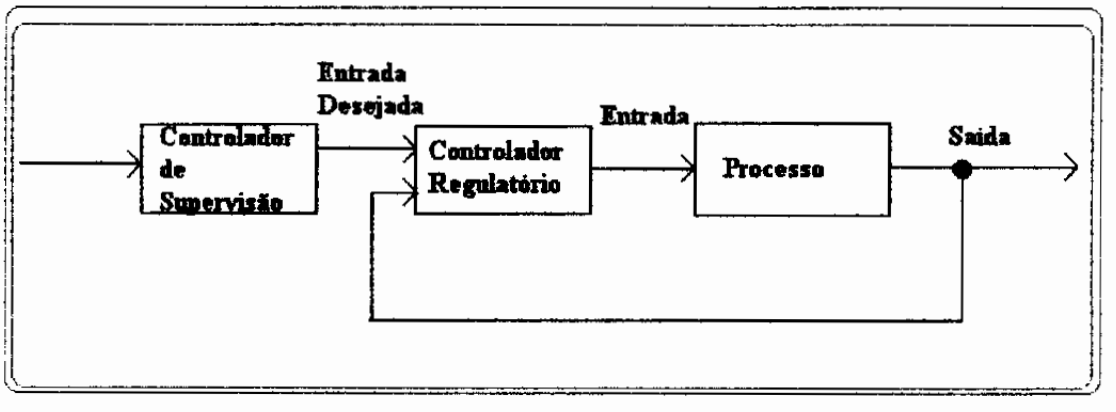


Fig 4.3- Esquema para o controle de supervisão de um processo (Gupta e Sinha, 1996)

4.2- CONTROLE CONVENCIONAL X NEUROCONTROLE:

Quando se menciona o termo ***controle convencional***, supõe-se automaticamente que o sistema que se deseja controlar é linear e invariante no tempo. Com isso, o modelo do sistema perde muitas de suas características que são importantes para o desenvolvimento do projeto de um controlador que funcione adequadamente. Como já foi mencionado, a linearidade só pode ser uma suposição satisfatória quando se tem interesse apenas em certos pontos de operação da planta.

Com a utilização de redes neurais artificiais pode-se liberar o sistema desta suposição de linearidade, fazendo com que o seu modelo utilizado possa incluir as não-linearidades do processo. Dessa forma, o ***neurocontrolador***, que é o controlador baseado em redes neurais, pode ser usado em uma variedade de problemas de controle para estimação de parâmetros de modelos para o desenvolvimento de um controlador não-linear.

O neurocontrole apresenta soluções com resultados mais satisfatórios quando aplicados a problemas reais, como: refinarias de petróleo, sistemas químicos, siderúrgicas, montadoras de automóveis. Porém, a melhor solução para problemas de controle pode ser a combinação entre o controle linear e o controle neural [3].

4.3- IDENTIFICAÇÃO DE SISTEMAS POR REDES NEURAIS:

O problema de identificação consiste na obtenção do modelo de um sistema em questão. Neste item serão apresentados quatro formas para a identificação de um modelo para um sistema.

A rede neural considera o processo de identificação como um treinamento baseado na otimização de uma função não-linear. Onde a solução é encontrar os pesos da rede neural, que são os valores que minimizam a função de custo J, mostrada abaixo:

$$\min_{W} J(W) \tag{4.1}$$

O tipo da função J e o algoritmo utilizados para o treinamento determinam o tipo de rede neural utilizada para a identificação do modelo. A seguir serão descritas algumas formas de identificação que utilizam redes neurais

4.3.1-Identificação Não-Paramétrica:

Este é o tipo de identificação mais utilizada, pois desenvolve modelos com o formato de caixa-preta, baseados nos dados de entrada-saída que determinam o comportamento do sistema. São estes mesmos dados que formam a matriz de treinamento que será apresentada à rede neural artificial. O esquema para este tipo de identificação não-paramétrica está mostrada na figura a seguir.

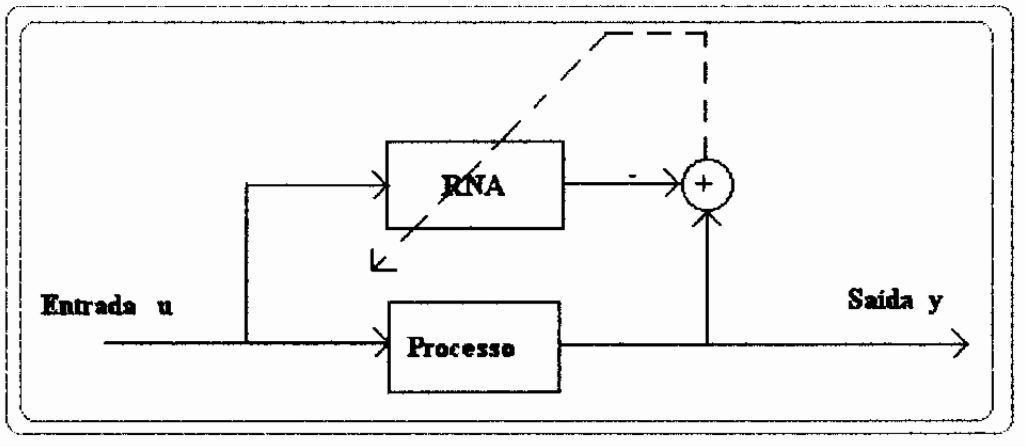


Fig 4.4- Esquema para a Identificação Não-Paramétrica (Gupta e Sinha, 1996)

O treinamento da rede só está terminado quando a sua saída estiver satisfatoriamente correspondendo a saída do processo que se deseja modelar. Portanto, o treinamento pode ser expressado como uma função baseada no erro existente entre a saída da rede e a saída do processo que deve ser minimizada, enquanto os pesos vão sendo ajustados. Para o treinamento *Backpropagation*, a função custo mais utilizada para a minimização [10] é a mostrada abaixo .

$$J(W) = \sum_{t=0}^{T} \left[y(t) - y_m(W,t) \right]^2 \qquad (4.2)$$

onde y(t) : saída do modelo do processo;

y_m (W,t): saída da rede neural.

Após o treinamento a rede neural funciona como uma caixa-preta que contém o modelo não-paramétrico (acesso apenas a informações sobre entrada e saída) do sistema em estudo. Este tipo de identificação é muito utilizada industrialmente para análise e monitoramento de processos e será o utilizado nesta monografia.

4.3.2 - Identificação Paramétrica:

Neste caso a identificação consiste em determinar valores de parâmetros e características de um modelo. Esta é exatamente a função da rede neural após ter sido realizado o treinamento.

Em geral, os valores de parâmetros de um modelo são estimados pela solução de problemas de otimização. A rede neural é indicada para esta identificação de parâmetros, quando estes dependem de alguma não-linearidade na informação originária de experimentos. Dessa forma, o próprio problema de otimização se torna não-linear, o que significa que a sua solução será muito mais complexa. A utilização das redes neurais evita complicados cálculos matemáticos, proporcionando valores adequados para os parâmetros.

A rede neural pode apresentar alguns problemas para a determinação destes valores se o treinamento não for adequado. Por exemplo, caso a matriz de treinamento seja esparsa pode haver uma generalização destes valores fazendo com que eles não sejam adequados para determinação da estrutura do modelo [3].

4.3.3 - Identificação do Modelo Inverso:

Para este caso uma rede neural é treinada para desenvolver um modelo inverso para um determinado processo. A informação apresentada na entrada da rede é a saída do processo, enquanto que a saída da rede é a entrada do processo. Neste tipo de identificação são treinadas duas redes, uma para identificar a estrutura do modelo e outra para estimar os valores dos parâmetros. Um esquema desta identificação é mostrada na figura a seguir.

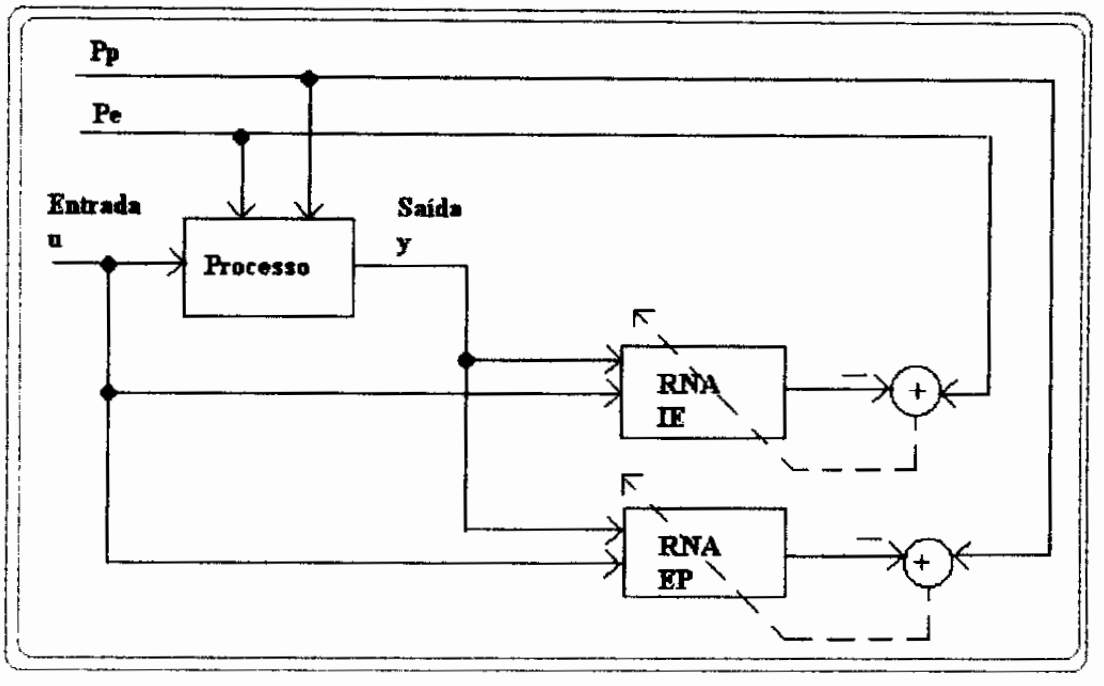


Fig 4.5- Esquema da Identificação do Modelo Inverso (Gupta e Sinha, 1996)

Onde: RNA-IE: rede neural artificial identificadora da estrutura do modelo;

RNA-EP: rede neural que estima os valores dos parâmetros do modelo;

Pe: vetor de componentes da estrutura do modelo;

Pp: vetor de parâmetros.

A função que deve ser minimizada neste caso, também é baseada na saída da rede e na saída do modelo, porém os dados de saída são as informações fornecidas para a entrada do processo [3].

$$J(W) = \left\| u(k) - u_m(W,k) \right\|^2 \tag{4.3}$$

onde u(k): entrada do processo;

u_m(k):saída da rede neural.

Para o desenvolvimento dos modelos inversos são utilizadas informações do comportamento do sistema em regime permanente. Porém, podem ocorrer alguns problemas no treinamento da rede que são causados diretamente pelas próprias características do sistema. Por exemplo, se o processo apresentar um atraso de transporte, isso significa que alguma mudança na entrada apenas se refletirá na resposta após um determinado tempo, portanto o processo inverso é não-causal. Um outro problema bastante comum é o mapeamento entre a entrada e a saída não ser uma função, mas apenas uma relação.

4.3.4 - Identificação de Parâmetros para Auto-Sintonia:

Esta identificação consiste na determinação de parâmetros para o controlador para que este proporcione um comportamento desejado para o sistema em malha fechada. Na maioria dos casos, este procedimento para identificação é utilizado para a determinação dos parâmetros para o controlador PID.

Para o treinamento das redes neurais são utilizadas informações sobre o comportamento do processo e do comportamento desejado pelo sistema de controle em malha fechada. Para este processo de realização da identificação dos parâmetros do controlador podem ser utilizados dois métodos, que estão descritos a seguir.

- *Primeiro Método:* neste caso, uma rede não-paramétrica do processo é treinada, que substituirá o sistema durante a utilização de um algoritmo de otimização não-linear (algoritmo de treinamento da rede neural) que fará a identificação dos parâmetros do controlador [3]. Este método possui uma certa complexidade computacional, mas não requer resposta em tempo real. A seguir está representado um esquema deste procedimento.

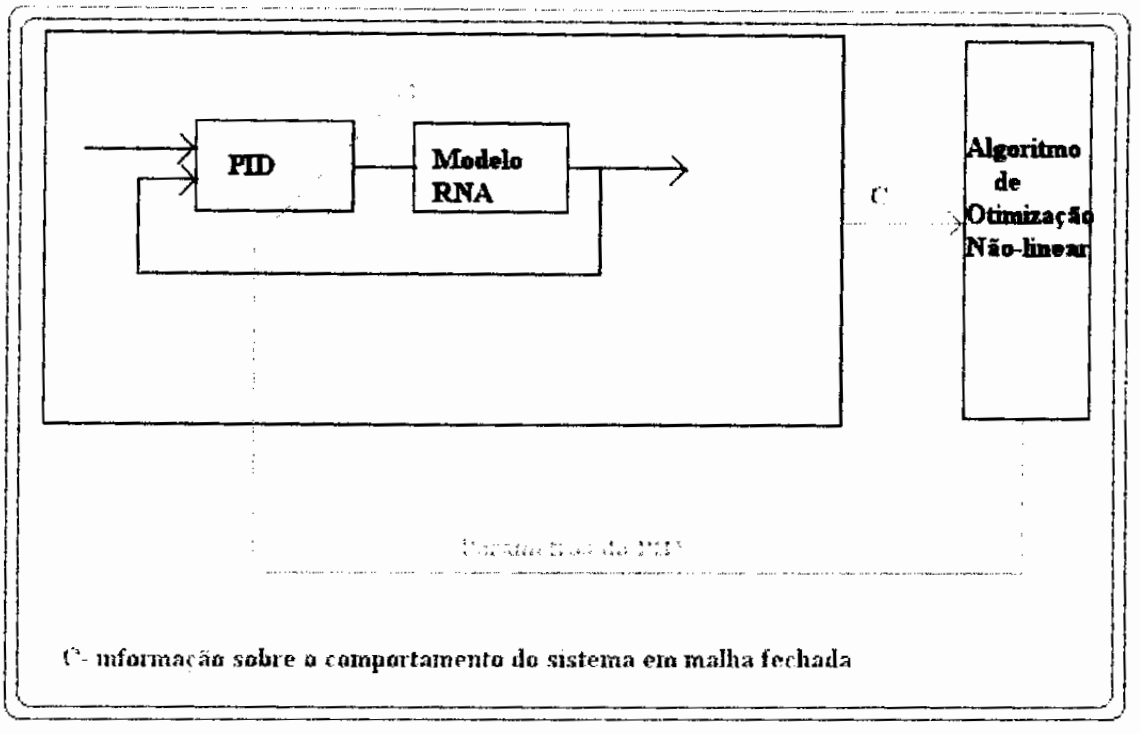


Fig 4.6- Esquema para auto-sintonia de um controlador PID usando uma rede neural como modelo do processo (modificado - Gupta e Sinha, 1996)

- *Segundo Método:* a rede neural funciona como o auto-sintonizador, sendo sua saída os parâmetros do controlador. Os dados para o treinamento são obtidos através de informações sobre o comportamento do processo que se deseja controlar. A vantagem deste método é que a rede pode ser treinada durante a simulação. Um esquema desta auto-sintonia está a seguir.

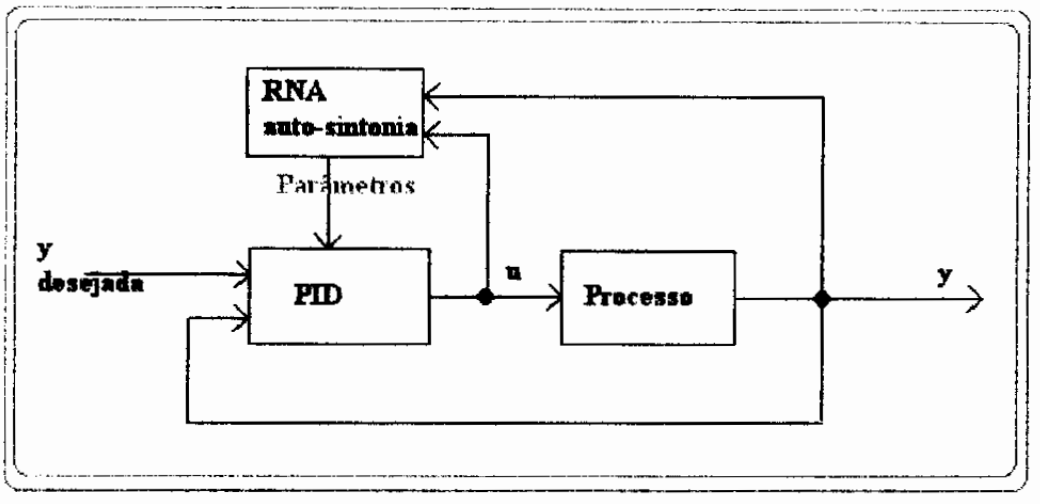


Fig 4.7 – Esquema de auto-sintonia por redes neurais de um controlador PID (modificado - Gupta e Sinha, 1996)

4.4- NEUROCONTROLADORES:

Neste item serão discutidos cinco procedimentos nos quais as redes neurais podem ocupar o lugar do próprio controlador. Isto ocorre quando a saída da rede fornece o sinal de controle para o processo. Abaixo serão descritos (de forma resumida) estes neurocontroladores.

4.4.1 - Modelamento do Controlador:

Neste procedimento as redes neurais são treinadas para emular controladores que já existem. A rede neural tem as mesmas informações de entrada do controlador e a diferença entre a sua saída e o sinal de controle que vem do próprio controlador dirigem o algoritmo de atualização dos valores dos pesos.

Portanto, a função custo a ser minimizada pode ser representada pela seguinte expressão[3]:

$$J(W) = \sum_{t} \left[u(t) - u_m(W,t) \right]^2 \tag{4.4}$$

onde u(t): sinal de controle advindo do controlador;

$u_m(t)$: sinal de controle advindo da rede neural.

Abaixo está um esquema para o treinamento deste neurocontrolador.

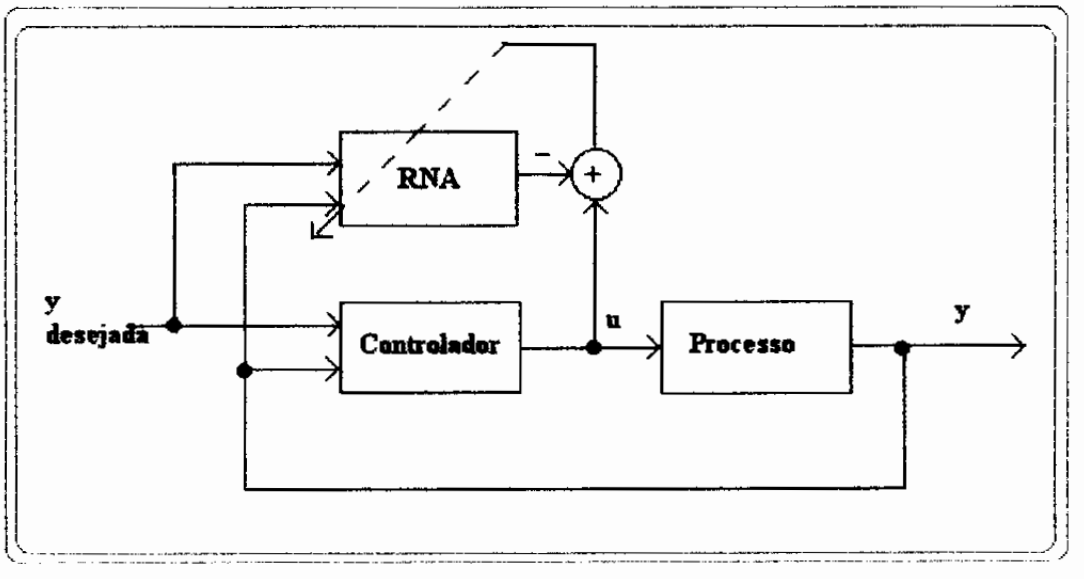


Fig 4.8- Esquema do modelamento de um controlador (Gupta e Sinha, 1996)

Porém, um problema é detectado quando se utiliza redes neurais para o modelamento de controladores. Os controladores são, em geral, sistemas dinâmicos, ou seja, possuem integradores, diferenciadores, entretanto as redes neurais, em sua maioria, são estruturas estáticas. Portanto, as informações dinâmicas devem ser explícitas quando apresentadas na entrada da rede, as informações sobre derivadas e integrais

devem estar disponíveis como entradas. Se for tomado como exemplo um PID, este teria como entrada o erro entre a referência e a saída do processo, a sua derivada e a sua integral.

4.4.2 - Otimização do Neurocontrolador sem Modelo:

Neste subitem será discutido o controle neural adaptativo sem a existência prévia de um modelo do processo em estudo. Neste caso, a vantagem deste procedimento é a ausência do modelo, mesmo durante o desenvolvimento do projeto do controlador .

O neurocontrolador é treinado baseado em informações do comportamento do sistema vindas do próprio sistema em operação, portanto durante todo o processo de aprendizado a planta não possui nenhuma lei de controle em funcionamento. Essa é uma desvantagem que o método apresenta para sua aplicação em processos industriais, pois nestes sistemas "falhas" devido a não existência de um controlador, durante o treinamento, não são aceitáveis [3].

4.4.3 - Otimização do Neurocontrolador baseado no Modelo:

Este procedimento tem a finalidade de desenvolver um neurocontrolador ótimo utilizando um modelo do processo. Como o modelo do sistema é conhecido então a rede neural pode ser treinada durante a simulação, objetivando a minimização de um função, em geral representada pela soma do erro quadrático [10].

$$J(W) = \frac{1}{2}[y_d - y_m]^2 \tag{4.5}$$

onde y_d : saída desejada ;

y_m : saída do modelo.

Esta não é a única expressão para a função *J*. Alguns outros exemplos são: a integral do erro quadrático, uma função baseada no valor do *overshoot* da resposta do processo, a extensão do processo da integral do erro quadrático, que penaliza a ação de controle excessiva [3]. Porém, a otimização da função (4.5) é a mais utilizada quando se trata do treinamento de um rede neural.

A seguir está um esquema para o treinamento deste neurocontrolador, durante uma simulação em malha fechada.

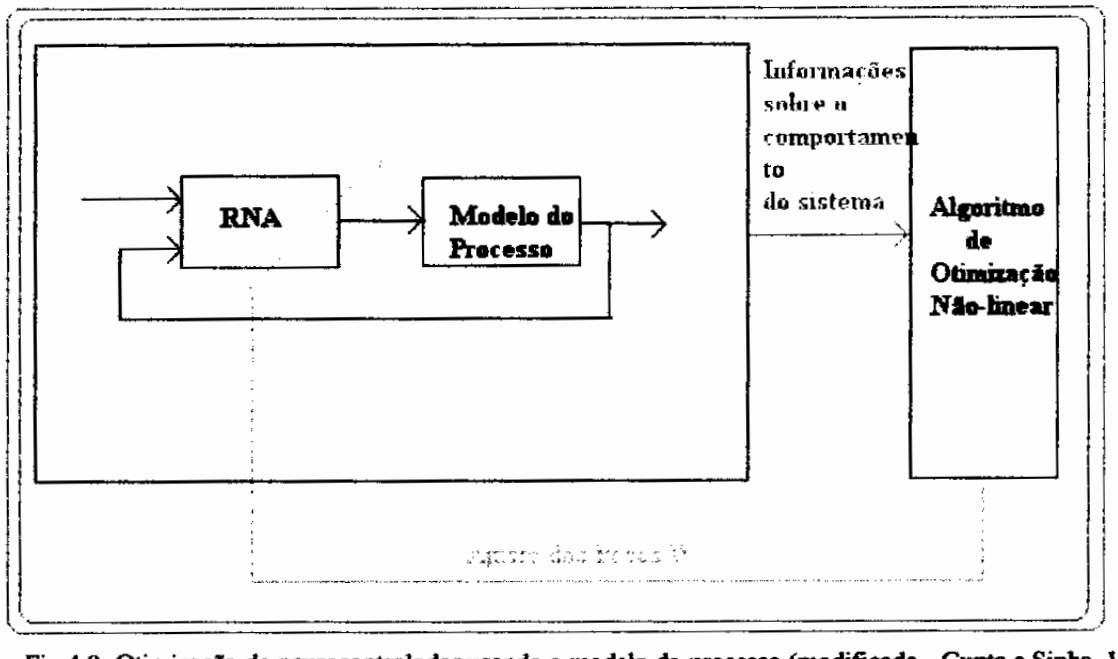


Fig 4.9- Otimização do neurocontrolador usando o modelo do processo (modificado - Gupta e Sinha, 1996)

Este algoritmo de otimização não-linear é o próprio algoritmo de treinamento da rede neural (como no caso da auto-sintonia de controladores). Para melhorar o seu tempo de

convergência, estes devem ser baseados no gradiente. Como o treinamento da rede neural depende do comportamento do modelo do sistema, se este não for suficientemente condizente com as características principais da planta real, o desempenho do controlador não será satisfatório. Uma forma para contornar este inconveniente é a substituição do modelo por uma outra rede neural, que deverá ser treinada com dados obtidos do comportamento do sistema em operação real.

Este é o procedimento adotado nesta monografia para controlar o pêndulo inverso sobre o carro. Os detalhes sobre este procedimento estão descritos no capítulo seguinte.

4.4.4 - Neurocontrolador Robusto:

A robustez é uma característica importante quando se trata de controle. Este neurocontrolador é treinado para obter um desempenho ótimo sobre um espaço de modelos do processo. Isto significa, que mesmo o controlador sendo altamente específico para o controle de um determinado processo, este deve tolerar desvios no comportamento do sistema.

Para que exista esta tolerância a variações do sistema ou de outros fatores é preciso que o próprio neurocontrolador seja treinado sobre um espaço de modelos do processo, que será denominado Π, para ter esta performace robusta.

A escolha da função não-linear a ser minimizada é livre, porém os critérios de avaliação sobre esta função podem mudar também. Pode ser usado, por exemplo, o valor médio da função ou então seu pior caso sobre o estado considerado.

Para um melhor entendimento do procedimento, um esquema do neurocontrolador robusto é mostrado a seguir.

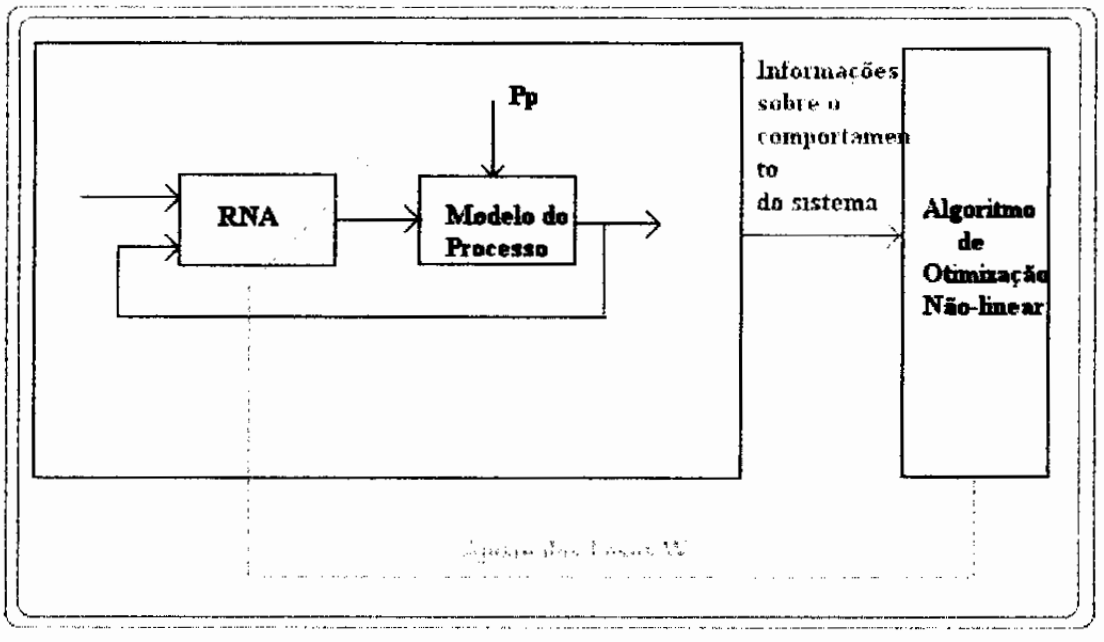


Fig 4.10- Otimização de um neurocontrolador para uma performace robusta (modificado - Gupta e Sinha, 1996)

A entrada **Pp** é um vetor de parâmetros do modelo do processo, podendo representar uma ou mais estruturas de modelo. Este vetor caracteriza o espaço de modelos do processo, Π.

4.4.5 - Neurocontroladores Parametrizados:

Quando se utiliza redes neurais para uma determinada aplicação, esta deve ser treinada especificamente para aquele caso, mesmo quando apenas pequenas mudanças ocorrem no critério de controle ou quando se deseja utilizar um neurocontrolador já existente para controlar uma planta similar.

Para que se possa reutilizar um neurocontrolador é necessário incluir nos seus dados de entrada informações sobre os parâmetros do modelo do processo.

Realizando uma manipulação destes parâmetros, o neurocontrolador, em questão, pode ser aproveitado para outros sistemas ou leis de controle.

Em geral, podem ser acrescentados à entrada da rede dois tipo de parâmetros:

- Parâmetros de controle;
- Parâmetros do modelo do processo.

Um esquema do procedimento para se obter este tipo de neurocontrolador está mostrado abaixo.

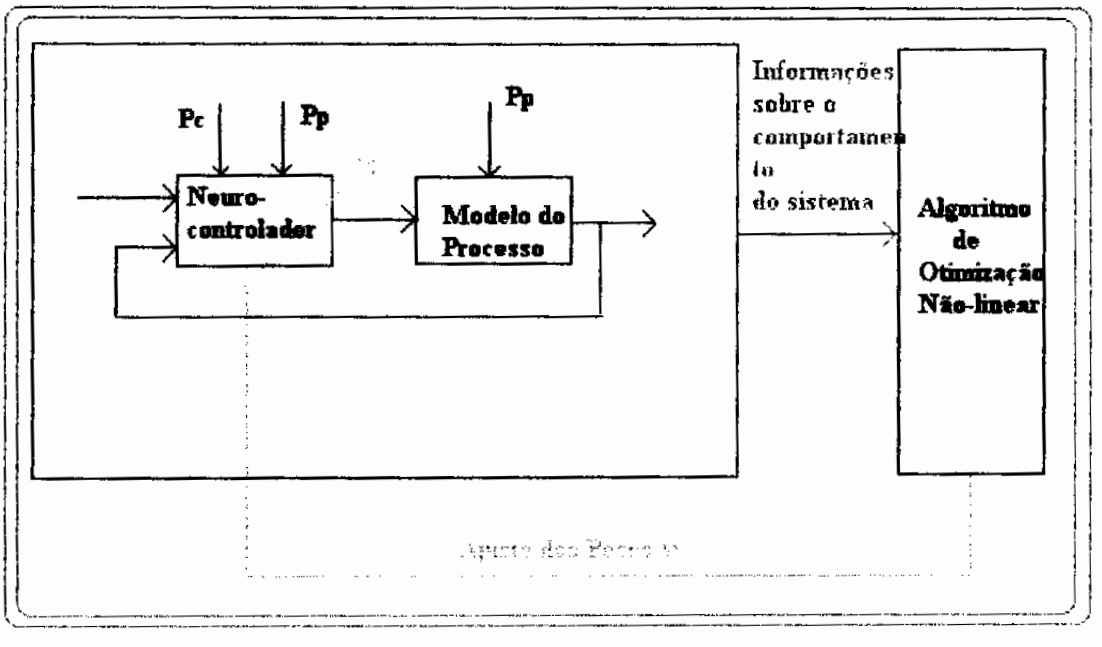


Fig 4.11- Desenvolvimento de um neurocontrolador parametrizado (modificado - Gupta e Sinha, 1996)

Normalmente para o projeto deste neurocontrolador, se deseja minimizar o pior caso da função não-linear dada pela seguinte expressão [3]:

$$J(W) = \max_{\substack{P_p \in \Pi \\ P_c \in \Gamma}} J_1(W, P_p, P_c)$$
$$J_1(W, P_p, P_c) = \int \left[(y_d - y_m(P_c, P_p))^T . (P_{c,1}) . (y_d - y_m(P_c, P_p)) + u_m(P_c, P_p)^T . (P_{c,2}) u_m(P_c, P_p) \right] dt \quad (4.7)$$

onde *Pc,1* e *Pc,2* são elementos da matriz de pesos do neurocontrolador parametrizado.

Deve ficar bem claro que o desenvolvimento de um neurocontrolador deste tipo envolve a otimização da função sobre dois espaços parametrizados: o espaço de estruturas do modelo Π e o espaço de parâmetros de controle Γ.

Portanto, o procedimento de identificação e o esquema de controle utilizado neste trabalho foram os descritos nas seções 4.3.1 e 4.4.3, respectivamente.

5 – METODOLOGIA PARA IDENTIFICAÇÃO E NEUROCONTROLE

Neste capítulo serão discutidas as metodologias utilizadas para o processo de identificação e para o projeto do neurocontrolador. Toda a teoria necessária para o entendimento destes processos foi desenvolvida nos capítulos anteriores, portanto aqui serão descritas as rotinas que foram desenvolvidas para o desenvolvimento do neurocontrole, incluindo a parte da identificação.

Este item está dividido em duas partes distintas, para uma melhor compreensão do procedimento utilizado. São elas:

- Metodologia para Identificação, e
- Metodologia para a determinação do Neurocontrolador.

5.1 – METODOLOGIA PARA IDENTIFICAÇÃO:

Como foi mencionado no capítulo 04, para a identificação do sistema, apesar de se conhecer o modelo do pêndulo inverso sobre carro, foi utilizado o procedimento para a identificação não-paramétrica, ou seja, que considera a planta uma caixa-preta desconhecida. Para tanto é necessário obter uma rede neural, que ao final do treinamento esteja substituindo satisfatoriamente o próprio sistema. Esse procedimento foi realizado, pois para o treinamento da rede que será o controlador, o erro da mesma se retropropagará pela rede que está substituindo o sistema. Esta concepção ficará mais clara no item seguinte.

O treinamento da rede foi realizado utilizando as funções da *Toolbox* de Redes Neurais do MATLAB®, porém foi desenvolvida uma rotina, denominada *final*, que simplifica o uso destas funções tanto para o treinamento como para a simulação. Esta rotina e suas janelas (para interface com o usuário) estão listadas no Apêndice.

Para a montagem da matriz de treinamento, em geral, realiza-se experimentos com o sistema em questão, para que seja possível determinar valores de entrada e de saída desejados para a rede neural. Porém, para este caso específico não foi possível realizar estes testes experimentais. Um protótipo começou a ser construído, entretanto não ficou completamente terminado para a realização destes testes. Uma fotografia deste protótipo está mostrada a seguir.

Este protótipo foi construído utilizando uma haste de alumínio acoplada a um peso, localizado em sua ponta superior. Este era o pêndulo invertido propriamente dito. Para o carro do pêndulo foi utilizado o carro de uma impressora. Porém, não foi possível conseguir resultados satisfatórios deste sistema real, em tempo hábil, para que fosse viável seu acoplamento ao neurocontrolador projetado.

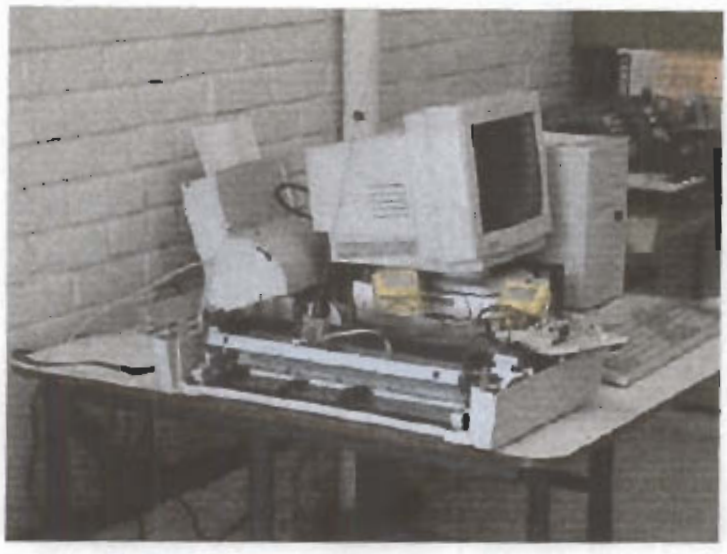


Fig 5.1 – Fotografia do protótipo do pêndulo inverso sobre carro

Dessa forma, foi montada uma matriz de entrada utilizando a função *combvec*, que gera uma combinação dos vetores definidos previamente. Para o pêndulo inverso foram definidos os seguintes vetores para cada variável de estado do sistema:

```
deg2rad=pi/180;

% definição dos intervalos das variáveis de estado:

 teta = [-20:5:20]*deg2rad;
 vteta = [-5:2.5:5]*deg2rad;
 force = -1:0.25:1;
 desloc=[-1:0.2:1];
 vdesloc=[-0.5:0.2:0.5];
 teta2=[-20:10:20];
```

Após a definição dos intervalos de valores de cada variável de estado, foi montada a matriz de entrada para o treinamento, como mostrado abaixo:

```
A1=combvec(teta,vteta);
A2=combvec(desloc,vdesloc,force);
A3=zeros(3,36);
inicial2=[[A1 A1 A1 A1 A1 A1 A1 A1;A2 A3] [teta2; zeros(4,length(teta2))]];
```

A repetição de A1, na montagem da matriz de entrada 'inicial2', se dá devido a necessidade de se ter as dimensões necessárias para que esta última possua todas as suas

linhas com o mesmo número de colunas. Outro ponto que deve ser esclarecido é a existência do vetor 'teta2'. Este vetor é apenas um outro conjunto de pontos da variável de estado θ; e que tem como função tornar a matriz de entrada mais 'rica' em informação, acrescentando diferentes valores para o ângulo. Cada conjunto de valores de variáveis de estado representam um estado particular, que foi arbitrado ao sistema pendular.

Não foi feito uma combinação de todos os vetores simultaneamente, pois este procedimento geraria uma matriz muito grande, mais de 190.000 amostras, o que tornaria o treinamento da rede impraticável. Dessa forma, tem-se uma matriz de entrada de 365 amostras, ou seja, a mesma terá uma dimensão de 5x365.

Para a geração da matriz de saída foi utilizado o modelo não-linear do sistema, que foi implementado utilizando a rotina **pendnl** (incluída no Apêndice). Para a solução destas equações foi usada a função *ode23*. Esta função é uma rotina, pertencente ao ambiente MATLAB, para o cálculo de soluções de equações diferenciais ordinárias utilizando o método de Bogacki-Shampine. A utilização desta função está mostrada a seguir:

```
% Simulação do modelo não-linear:

 timestep = 0.01;
 Q = length(inicial2);
 final_nlinear2 = [ ];

 for i=1:Q
   [ode_time,ode_state] = ode23('pendnl',[0 timestep],inicial2(:,i));
   final_nlinear1 = ode_state(length(ode_state),1:4)';
   final_nlinear2=[final_nlinear2 final_nlinear1];
end
```

Esta simulação tem uma importante característica: o seu tempo de execução. Este teve de ser muito pequeno, pois após um intervalo de tempo, o mesmo apresentava uma falha no cálculo das iterações devido a uma singularidade nas equações que descrevem este modelo. Esta singularidade é devida a presença de cossenos do ângulo θ (ângulo da haste em relação a referência) no denominador das expressões. Portanto, para evitar este problema, foi utilizado um tempo de iteração de 10ms.

Cada conjunto de valores de variáveis de estado que formam esta matriz de saída representa um novo estado do sistema pendular, ou seja, dado um estado de entrada a matriz de saída indicará o estado do sistema após 10ms.

Depois de determinadas as matrizes de entrada e saída, determinou-se o tipo de rede neural que seria utilizada para a identificação. Para o caso deste sistema pendular foram testadas duas arquiteturas de redes: a RBF e a *Perceptron* Multicamadas, sendo que a segunda foi escolhida devido ao seu melhor desempenho nos testes.

Dessa forma, foi utilizada uma rede *Perceptron* Multicamadas, com o treinamento *Backpropagation*. Esta rede era formada por 15 neurônios na camada oculta e 4 neurônios na camada de saída, onde esta camada teria como saída o novo estado do sistema pendular inverso. Para a determinação dos pesos iniciais foi utilizada a função *initff*, que determina os valores dos pesos baseada na distribuição dos valores da matriz de entrada. Para o treinamento foi escolhido o algoritmo *trainlm*. Este algoritmo é um algoritmo do tipo *Backpropagation*, porém garante um melhor desempenho, pois utiliza um método de aproximação de Newton denominado Levenberg-Marquardt. Esta técnica de otimização é mais poderosa do que o decremento do gradiente (técnica mais utilizada em rotinas de treinamento), mas requer mais memória computacional [16].

A regra de ajuste dos pesos é dada por:

$$\Delta W = (J^T . J + \mu . I)^{-1} . J^T . e \qquad (5.1)$$

onde:

J – matriz Jacobiana de derivadas de cada erro para cada peso;

μ - escalar;

e – vetor de erro.

Caso o escalar μ tenha um valor muito grande, a expressão (5.1) se aproxima do expressão para o decremento do gradiente, porém se este é pequeno a expressão se torna o método de Gauss-Newton. Este método é mais rápido e mais acurado.

Para uma melhor visualização das variáveis de entrada e de saída da rede neural durante o treinamento, estas estão representadas, a seguir, em um diagrama de blocos simplificado.

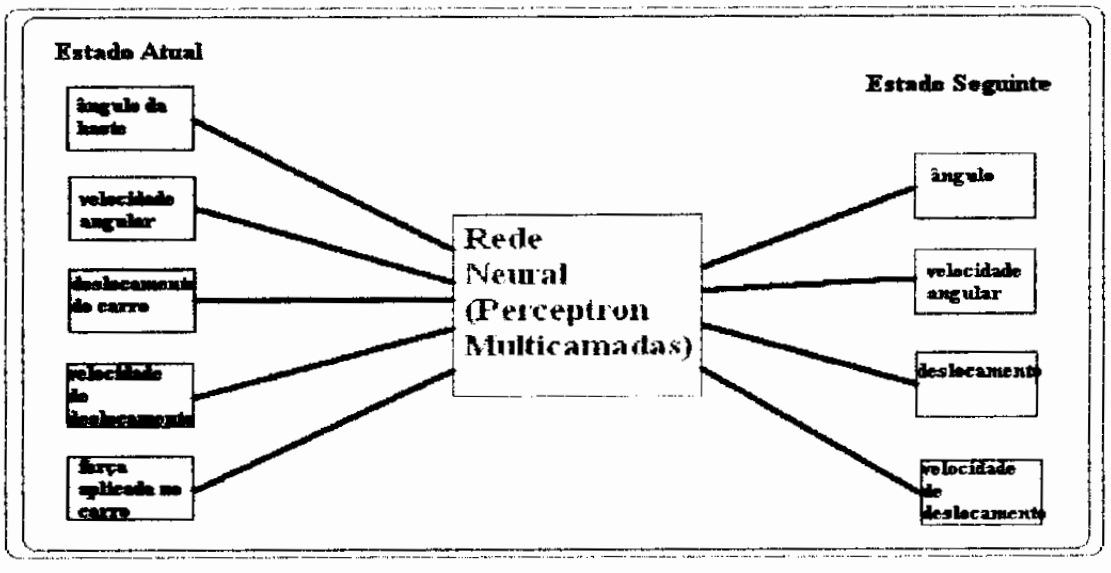


Fig 5.2 – Diagrama de blocos para a visualização das variáveis de entrada e saída da rede neural de identificação

Abaixo estão mostrados os comandos utilizados para o treinamento desta rede de identificação:

```
Q=365; % número de amostras da matriz de treinamento
S1 = 15; % número de neurônios da camada oculta
[mW1,mb1,mW2,mb2] = initff(inicial2,S1,'tansig',final_nlinear,'purelin');


df = 1;
me = 1000;
eg = (0.01^2)*Q*2;

tp = [df me eg];


[mW1,mb1,mW2,mb2,ep,tr] =
trainlm(mW1,mb1,'tansig',mW2,mb2,'purelin',inicial2,final_nlinear,tp);
```

onde:

mW1 e mW2 – vetor de pesos;

mb1 e mb2 – vetor de bias;

df – número de épocas por display (gráfico representativo da soma do erro quadrático) durante o treinamento ;

me – número máximo de épocas para o treinamento; e

eg – expressão para o cálculo da soma do erro quadrático desejada.

A expressão da soma do erro mostrada anteriormente, na rotina, foi especificada para este trabalho. Esta apenas determina o valor do erro que o projetista deseja que a rede neural alcance para que o treinamento possa ser interrompido. Poderia ter sido arbitrado qualquer valor condizente, ao invés de se utilizar a expressão. O MATLAB, durante a execução da rotina de treinamento para a rede neural, calcula a soma do erro quadrático no término de cada época, para que este valor calculado seja comparado com o valor desejado. A expressão utilizada pelo algoritmo de treinamento está mostrada abaixo:

$$S = \sum (Ed - En)^2 \tag{5.2}$$

onde:

Ed – saída desejada da rede neural;

En – saída atual da rede neural para um dado vetor de entradas ou conjunto de vetores.

Pode ser observado que as funções de ativação escolhidas para a camada oculta e para a camada de saída foram, respectivamente, a tangente-sigmóide e a puramente linear. Estas funções foram assim determinadas, pois são as utilizadas no algoritmo de treinamento para o controlador neural, que será discutido mais adiante.

Os testes para a validação da identificação do sistema pendular serão mostrados no capítulo seguinte.

5.2- METODOLOGIA PARA A DETERMINAÇÃO DO NEUROCONTROLADOR:

Neste item será discutido sobre a metodologia que foi idealizada para a determinação do neurocontrolador e a que foi realmente utilizada para que este apresentasse um desempenho satisfatório.

Primeiramente será descrito a metodologia que foi estudada, e não apresentou resultados que estivessem de acordo com o que era desejado. Mas o que se deseja de um controlador para o sistema pendular? Neste caso, era necessário que o controlador fizesse com que o carro fosse capaz de seguir uma trajetória (uma onda quadrada por exemplo) e que também mantivesse a haste do pêndulo equilibrada na sua posição de referência ($\theta = 0$).

Para tanto foi montado um conjunto de treinamento, onde a matriz de entrada teve seus valores arbitrados como na seção anterior. Os valores para os estados do sistema foram:

```
deg2rad=pi/180;

teta = [-20:4:20]*deg2rad;
vteta = [-2:0.4:2]*deg2rad;
desloc=[-1:0.2:1];
vdesloc=[-0.5:0.2:0.5];
demand=[-4:1:4]*deg2rad;
teta2 = [-20:2.5:20]*deg2rad;


 D=combvec(teta,vteta);
 E=combvec(desloc,vdesloc,demand);
```

Como pode ser notado, foi utilizado o comando *combvec* da mesma forma que no processo de identificação, que gerou uma matriz de 5 linhas e 341 colunas. Onde a variável *demand* é o ângulo no qual a haste está afastado inicialmente da sua posição de referência, tendo como objetivo dar o primeiro movimento ao carro.

Para a determinação da matriz de saída foi utilizado um modelo linear de referência. Este modelo tem a função de indicar o comportamento que se deseja que o sistema com o controlador inserido possua. Neste ponto do processo houve um problema. Não foi conseguido determinar um modelo de referência que tivesse uma resposta condizente com a desejada durante os testes. Vários comportamentos foram testados, isto é, foram testadas diferentes localizações de pólos e zeros, dentre outras modificações utilizadas.

A seguir está exemplificada uma das tentativas de determinação do modelo de referência:

```
% Estados

teta  = y(1);
vteta = y(2);
desloc = y(3);
vdesloc=y(4);
demand=y(5);

% Càlculo das derivadas

dteta  = vteta;
dvteta = -131.5789*(demand)-67.0526*(teta);
ddesloc=vdesloc;
dvdesloc= 52.6316*(demand)+46.42105*(teta);
ddemand = 0;

% Retorno das Derivadas

dy = [dteta; dvteta;ddesloc; dvdesloc;ddemand];
```

Caso fosse conseguido determinar este modelo de referência, se realizaria o treinamento da rede que representaria o controlador. Este treinamento seria realizado utilizando a rede neural de identificação, para determinar os pesos da nova rede neural, sem contudo haver nenhuma alteração do valor dos pesos da primeira rede.

Para um entendimento mais claro do procedimento do treinamento, um diagrama de blocos está mostrado a seguir:

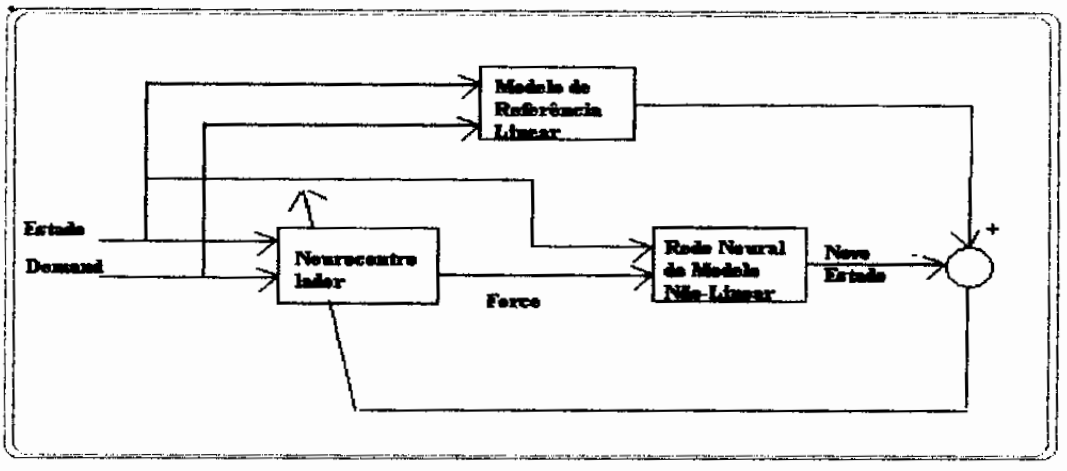


Fig 5.3 – Diagrama de blocos do procedimento de treinamento idealizado do controlador(figura retirada da referência [16])

Dessa forma, o neurocontrolador deveria aprender o comportamento determinado pelo modelo de referência. Para este treinamento foi utilizada a função *traincon* da *Toolbox* de Redes Neurais. A sua utilização está mostrada abaixo:

```
S1 = 15;
[cW1,cb1,cW2,cb2] = initff(inicial_neuro,S1,'tansig',1,'purelin');

Q=341;
df = 1;
me = 800;
eg = (0.04^2)*Q*2;
mu = 1e-5;

tp = [df me eg NaN mu];

[cW1,cb1,cW2,cb2,ep,tr] =
traincon(cW1,cb1,cW2,cb2,mW1,mb1,mW2,mb2,inicial_neuro,final_neuro,tp)
```

Portanto, após o treinamento o neurocontrolador teria como saída a força que deveria ser aplicada ao carro, para que o sistema pendular ficasse estável e fosse capaz de seguir um sinal de referência, sendo esta força determinada tendo como base os estados conseguidos e os desejados para o sistema. Para o sistema estudado durante este trabalho ele não foi eficiente durante a fase de testes, não controlando a posição do carro e nem mantendo

o equilíbrio da haste. Portanto foi necessário adotar uma outra metodologia para a determinação de um neurocontrolador para o sistema pendular invertido com carro.

Neste procedimento a rede neural que ocupará o lugar do controlador será treinada utilizando um conjunto de treinamento, no qual a matriz de entrada será o estado atual do sistema, excetuando a variável que indica a posição de deslocamento, que será a diferença entre a posição do carro que se obteve e a desejada (sinal de referência, que o carro deverá seguir), e a matriz de saída será a própria força que deverá ser aplicada ao carro para manter o sistema estável e equilibrado. Portanto, não será mais necessário realizar um treinamento baseado na retropropagação do erro entre os estados atual e desejado, utilizando uma rede de identificação do processo.

Para a montagem da matriz de entrada foi utilizado o mesmo procedimento que o anterior, foram arbitrados valores para as variáveis de estado. Estes vetores estão mostrados abaixo:

```
teta = [-20:4:20]*deg2rad;
vteta = [-2:0.4:2]*deg2rad;
desloc=[-1:0.2:1];
vdesloc=[-0.5:0.2:0.5];
demand=[-4:1:4]*deg2rad;
teta2 = [-20:2.5:20]*deg2rad;

D=combvec(teta,vteta); E=combvec(desloc,vdesloc,demand);
```

Para a determinação do vetor de saída foi utilizado conceitos de uma outra técnica de controle inteligente, denominada ***Lógica Fuzzy***. Como a utilização deste método foi devido a problemas na execução da metodologia proposta neste trabalho (não foi conseguido determinar um modelo de referência para a montagem da matriz de saída), uma introdução sobre os conceitos da teoria *fuzzy* foi acrescentado no Apêndice deste trabalho.

Portanto, foram determinadas certas funções de pertinência não-lineares para as variáveis de estado do sistema pendular inverso. Como entrada, foi utilizada a matriz de variáveis de estado que foi arbitrada anteriormente. Para a determinação do vetor de saída foi utilizada a *Toolbox* de Lógica *Fuzzy* do MATLAB. As funções de pertinência e as regras utilizadas para a determinação deste vetor de saída foram sugeridas por um exemplo desta *toolbox* [17].

Abaixo está mostrado um diagrama, que visa um melhor esclarecimento do procedimento da montagem do conjunto de treinamento.

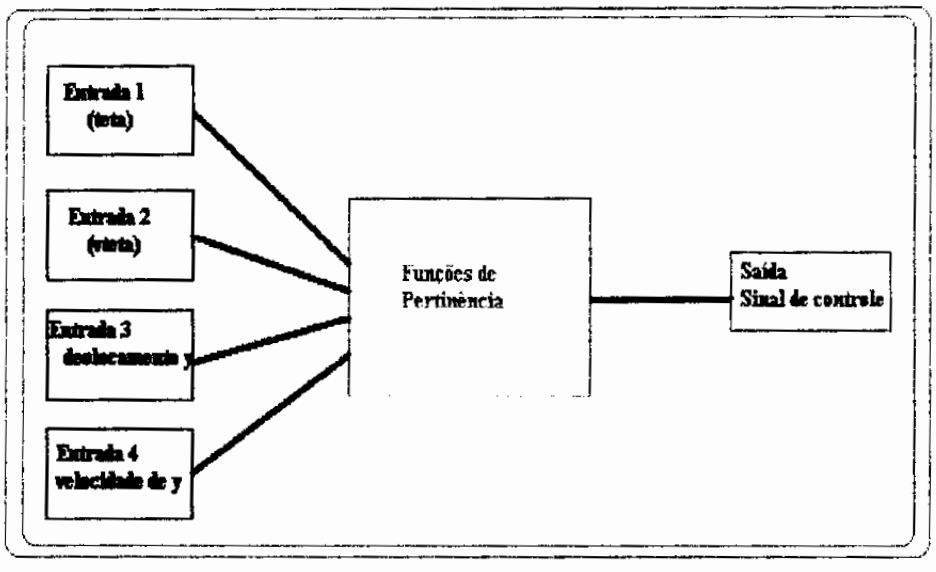


Fig 5.4- Diagrama de blocos do procedimento de determinação do vetor de saída de treinamento.

O bloco denominado funções de pertinência contém tanto as funções não-lineares como as regras para sua utilização. Foram necessárias 16 regras para o procedimento cobrir todas as possibilidades do comportamento do sistema para a determinação de valores condizentes do sinal de controle. Algumas destas regras estão descritas abaixo, todas estão incluídas no Apêndice C .

```
1. If (teta1 is teta1mf1) and (vteta2 is vteta2mf1) and
(desloc3 is desloc3mf1) and (vdesloc4 is vdesloc4mf1) then (u is umf1) (1)
2. If (teta1 is teta1mf1) and (vteta2 is vteta2mf1) and
(desloc3 is desloc3mf1) and (vdesloc4 is vdesloc4mf2) then (u is umf2) (1)
3. If (teta1 is teta1mf1) and (vteta2 is vteta2mf1) and
(desloc3 is desloc3mf2) and (vdesloc4 is vdesloc4mf1) then (u is umf3) (1)
4. If (teta1 is teta1mf1) and (vteta2 is vteta2mf1) and
(desloc3 is desloc3mf2) and (vdesloc4 is vdesloc4mf2) then (u is umf4) (1)
```

Onde tetamf1 e tetamf2 são as duas funções de pertinência que estão relacionadas aos valores da entrada teta, o mesmo acontecendo para as outras variáveis de estado e, inclusive para a saída 'u'. Foram utilizadas funções não-lineares do tipo sigmóide. Além disso, o processo de fuzzificação e defuzzificação foram dados pelo produto dos valores obtidos utilizando as funções de pertinência(método de Larsen) e pela média dos máximos, respectivamente. O método de interpolação(restrição dos termos nebulosos) utilizado nas regras foi o de Sugeno, no qual é assumido que as funções são monotônicas e as conclusões das regras sejam dadas por funções:

$$s_i(x_1, x_2, ..., x_m) = d_0 + d_1.x_1 + ... + d_m.x_m \tag{5.3}$$

onde d é uma constante.

Para a utilização efetiva destas regras e funções, para a montagem do vetor de saída, foi montado o seguinte diagrama de blocos no SIMULINK:

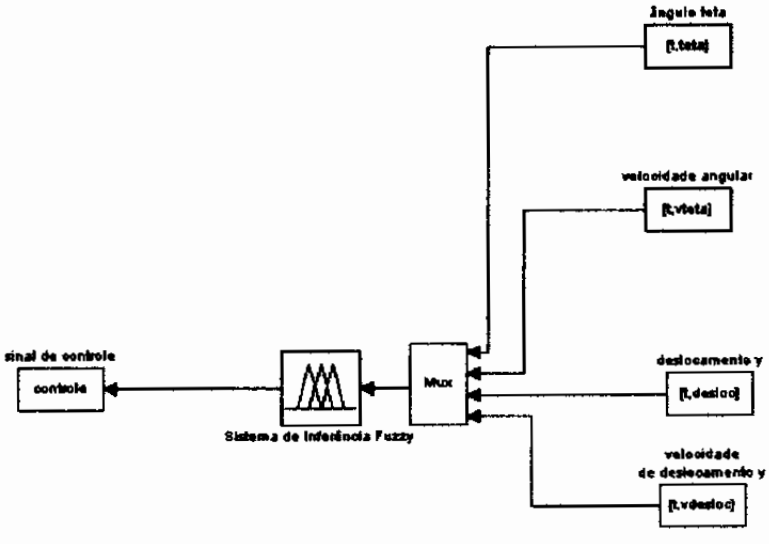


Fig 5.5 – Diagrama de blocos para a determinação do vetor de saída

Após a montagem do conjunto de treinamento, este foi realizado utilizando o mesmo algoritmo de treinamento da identificação do sistema, o *trainlm*, baseado no método de Levenberg-Marquardt, citado anteriormente. Para tanto foi utilizada uma rede com quinze neurônios na camada oculta e um neurônio na camada de saída, sendo as funções de ativação destas camadas ,a *tansig* e a *purelin*, respectivamente. A saída desta rede é a própria força aplicada ao carro do pêndulo inverso, para que este possa se equilibrar e seguir uma trajetória arbitrada, como já foi explicado.

Os testes e os resultados que validam o desempenho satisfatório do controlador planejado estão no capítulo a seguir.

Utilizando este procedimento para a determinação do controlador, a parte de identificação e de neurocontrole se tornaram completamente independentes, e não mais complementares como foi proposto inicialmente, quando a rede de identificação fazia parte do procedimento de determinação do rede neural responsável pelo controle.

Resultados

6 – RESULTADOS DA IDENTIFICAÇÃO E DO NEUROCONTROLE

Neste capítulo serão mostrados os gráficos obtidos dos testes realizados com as redes utilizadas para identificar e para controlar o sistema do pêndulo inverso com carro. Além disso, também serão incluídos e discutidos os procedimentos adotados para a realização destes testes.

Para manter a coerência com o capítulo anterior, este também será dividido em duas seções, a primeira dedicada a identificação do sistema e a outra ao controle do mesmo.

Os itens estão descriminados abaixo:

- Resultados da Identificação, e
- Resultados do Controlador Neural.

6.1 – RESULTADOS DA IDENTIFICAÇÃO:

Neste item serão mostrados os resultados obtidos da rede treinada para identificar o modelo não-linear do pêndulo inverso com carro, além disso também serão comentados o procedimento para a realização do teste.

Para este caso a rede demorou 223 épocas para ser treinada, ou seja, para o seu erro (calculado durante as iterações do algoritmo de treinamento) alcançar a soma do erro quadrático desejado (equações definidas no capítulo anterior) de 0.292. O gráfico mostrado a seguir representa o comportamento do treinamento da rede de identificação:

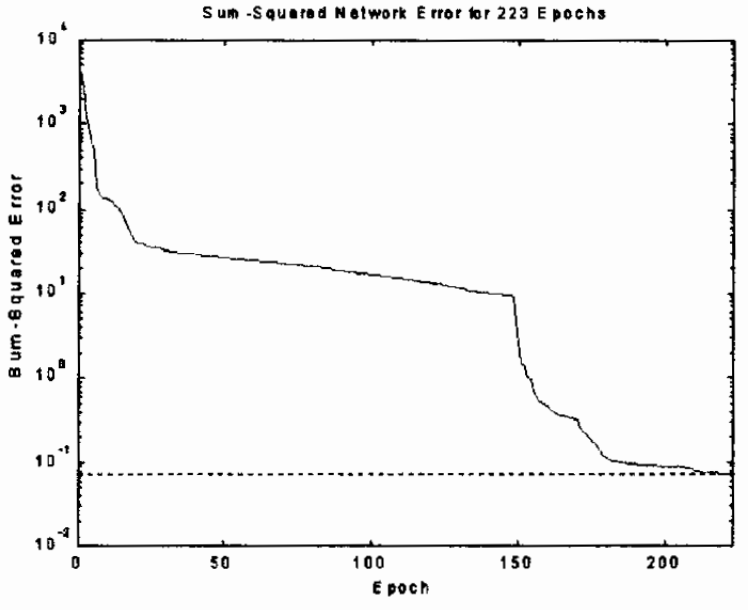


Fig 6.1 – Gráfico da soma do erro-quadrático x número de épocas

Após o treinamento, foram realizados dois testes para a validação desta rede neural. O primeiro teste foi feito utilizando uma matriz de entrada gerada de forma que os pontos estivessem dentro do intervalo dos valores das variáveis de estado utilizados durante o treinamento, mas que não coincidissem com estes. Parte da rotina utilizada para gerar esta matriz está representada a seguir:

```
deg2rad=pi/180;

 teta = [-5:1:5]*deg2rad;
 vteta = [-2:0.6:2]*deg2rad;
 desloc=[0:0.15:0.5];
 vdesloc=[-0.5:0.1:0.5];
 demand=[-0.5:0.2:0.5];

 D=combvec(teta,vteta);
 E=combvec(desloc,vdesloc,demand);
 F=zeros(3,44);

 teste_neuro = [D D D D;E F ]; % matriz de entrada para teste de desempenho
da rede neural de identificação
```

Esta matriz de pontos não representa uma trajetória, mas apenas pontos individuais. Ou seja, cada coluna desta matriz representa um estado do sistema, sem haver uma ligação com os valores da coluna anterior.

Depois de gerada esta matriz, foi feita a comparação entre o resultado conseguido através do modelo não-linear e o resultado da rede neural de identificação. Portanto, para a obtenção do novo estado do sistema no instante de 10ms, após o estado de entrada, onde o passo de interação também é de 10ms, foram utilizados os seguintes comandos:

```
% Simulação do modelo não-linear:


 timestep = 0.01;
 Q = length(teste_neuro);
 resultado = [ ];

 for i=1:Q
   [ode_time,ode_state] = ode23('pendnl',[0 timestep],teste_neuro(:,i));
   teste_nlinear1 = ode_state(length(ode_state),1:4)';
   resultado=[resultado teste_nlinear1];

end
```

Para a simulação da rede neural foi utilizado o comando *simuff*, (algoritmo de teste para a rede *Perceptron* Multicamadas da *Toolbox* de Redes Neurais) da seguinte maneira:

```
Rede=simuff(teste_neuro,mW1,mb1,'tansig',mW2,mb2,'purelin');
```

Abaixo estão mostrados os gráficos que possibilitam a comparação descrita anteriormente. Os resultados da rede de identificação estão representados por asteriscos na cor verde, enquanto que os pontos resultantes do modelo não-linear do sistema estão representados em azul. Para uma melhor visualização da resposta da rede, os pontos em azul estão ligados, porém, como foi comentado, estes não compõem uma trajetória.

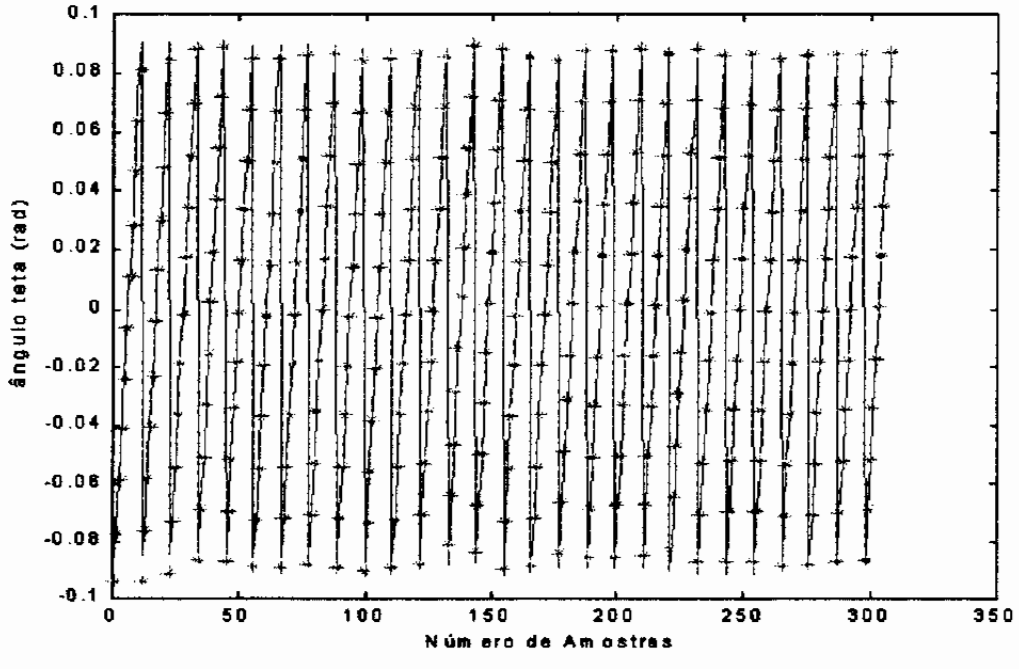


Fig 6.1 – Resultado da comparação entre os valores do ângulo teta obtidos pelo modelo não-linear e a rede de identificação

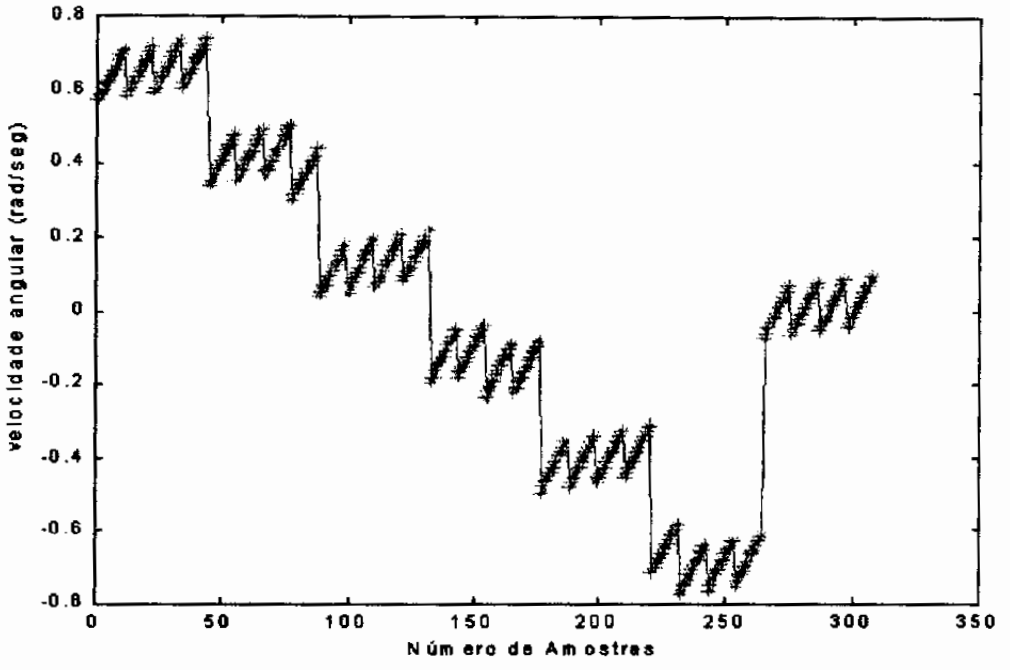
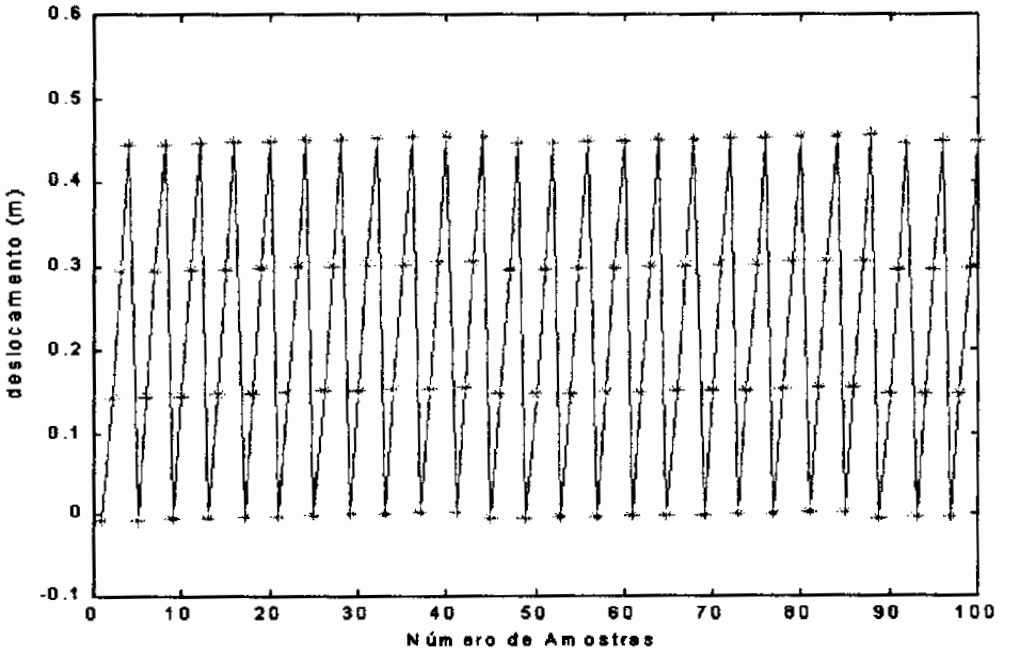


Fig 6.2 - Resultado da comparação entre os valores da velocidade angular obtidos pelo modelo não-linear e a rede de identificação



6.3 - Resultado da comparação entre os valores do deslocamento y obtidos pelo modelo não-linear e a rede de identificação

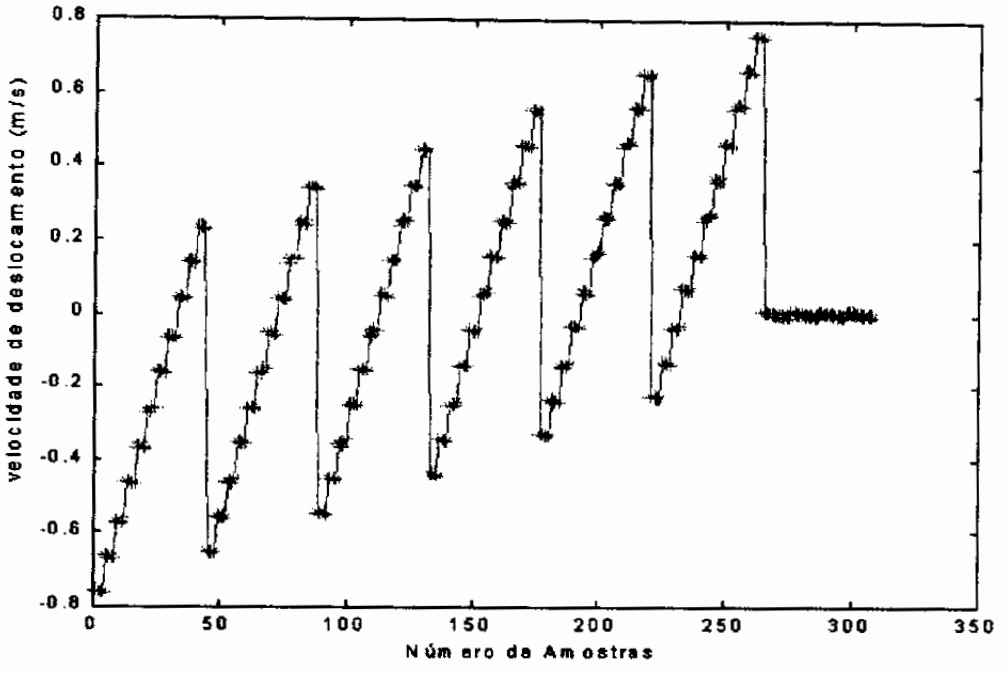


Fig 6.4 - Resultado da comparação entre os valores da velocidade do deslocamento y obtidos pelo modelo não-linear e a rede de identificação.

Observando os gráficos pode-se concluir que a resposta da rede de identificação foi realmente satisfatória, ao ser comparada com o comportamento do modelo não-linear, ou seja, a rede neural apresenta valores muito próximos com aqueles resultantes das expressões do modelo não-linear. Para corroborar estes resultados foi simulada uma trajetória de queda da haste, por cerca de 50 ms. Este tempo foi utilizado durante os dois testes, devido ao problema de singularidade que o modelo adotado apresenta, e que foi discutido previamente.

Para a simulação desta trajetória pelo modelo e pela rede neural foi utilizada a seguinte rotina descrita abaixo:

```
% Testando o Modelo
% ================

deg2rad=pi/180;

teta = 5 * deg2rad;
vteta = 0*deg2rad;
desloc=0;
```

```
vdesloc=0;
force = 0;

init_state = [teta; vteta;desloc;vdesloc];


[p_time,p_states] = ode23('pendnl',[0 0.05],[init_state; force]);
p_states = p_states';


%    Resposta Rede Neural de Identificação
%    ================================


time = 0:0.01:0.05;
state = init_state;
states = zeros(4,length(time));
states(:,1) = state;
for i=4:length(time)
  state = state + simuff([state;force],mW1,mb1,'tansig',mW2,mb2,'purelin');
  states(:,i) = state;
end
```

Os gráficos obtidos neste teste estão mostrados a seguir:

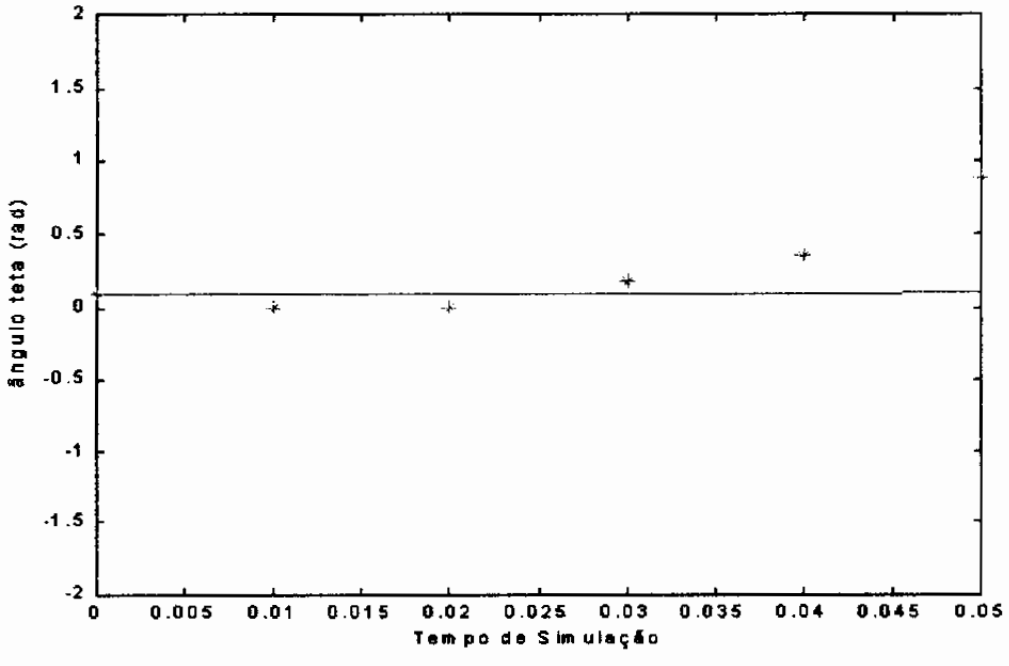


Fig 6.5 – Gráfico de comparação entre os valores do ângulo teta do modelo não-linear (pontos em azul) e da rede (pontos em verde) durante uma trajetória de queda da haste

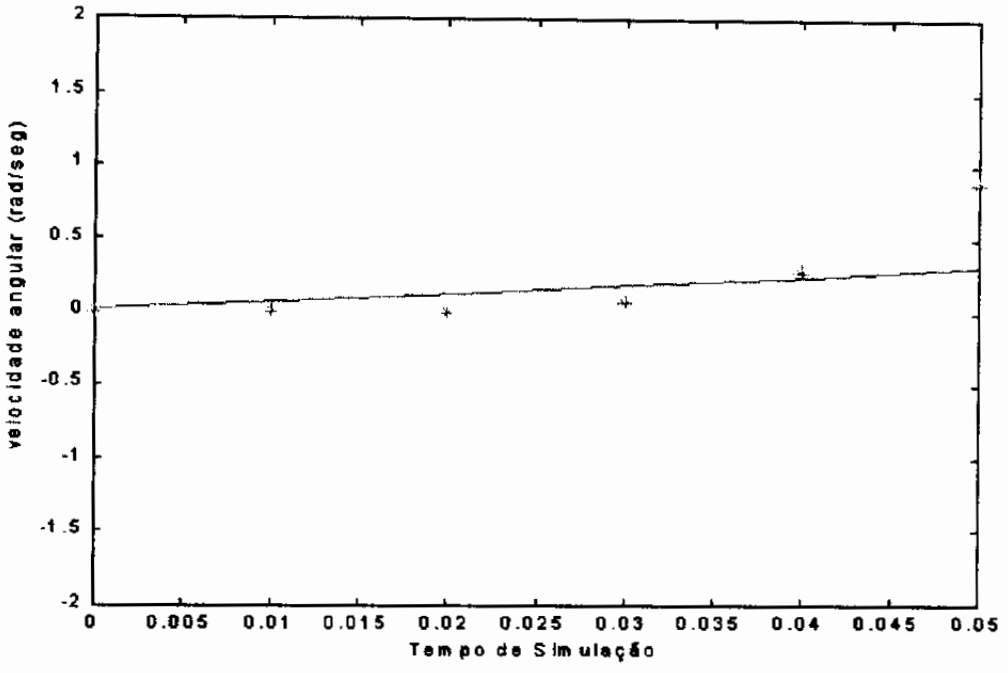


Fig 6.6 - Gráfico de comparação entre os valores da velocidade angular do modelo não-linear (curva em azul) e da rede (pontos em verde) durante uma trajetória de queda da haste

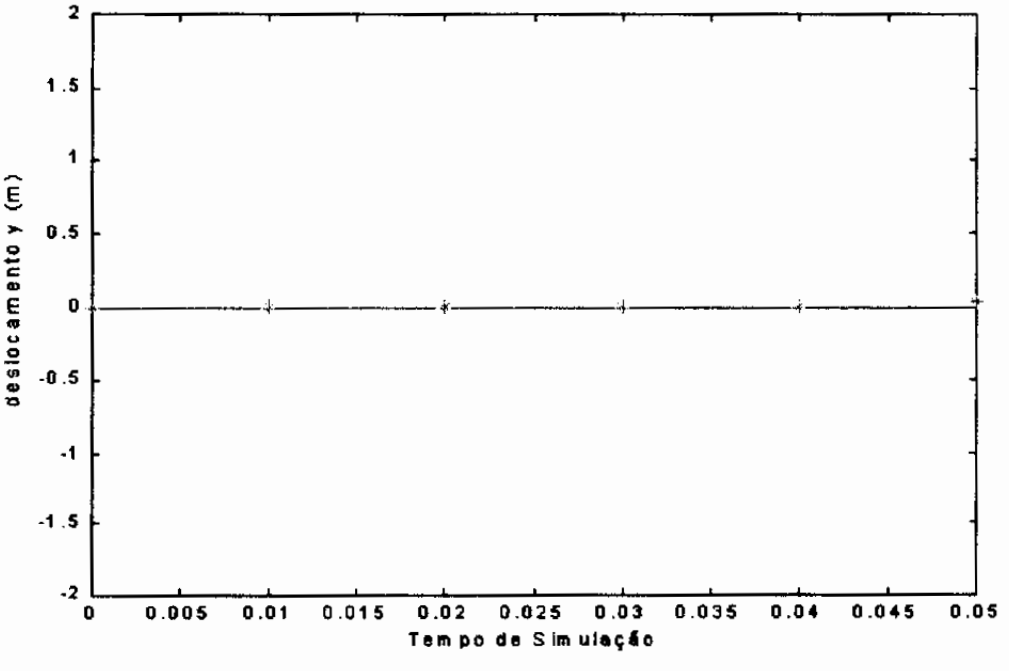


Fig 6.7 - Gráfico de comparação entre os valores do deslocamento y do modelo não-linear (curva em azul) e da rede (pontos em verde) durante uma trajetória de queda da haste

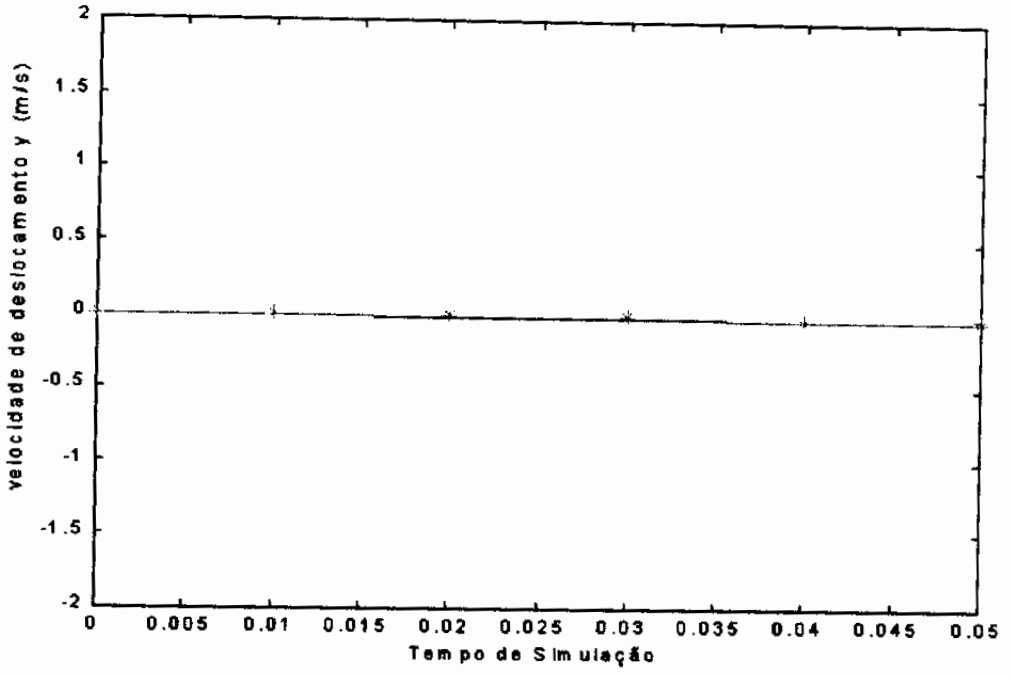


Fig 6.8 - Gráfico de comparação entre os valores da velocidade de deslocamento y do modelo não-linear (curva em azul) e da rede (pontos em verde) durante uma trajetória de queda da haste

Os pontos resultantes da rede não foram ligados como os resultantes do modelo não-linear para uma melhor visualização do comportamento da mesma durante a trajetória.

A partir da realização destes dois testes foi possível validar a rede de identificação do modelo não-linear do sistema pendular, ou seja, a rede neural apresentou valores, nos dois testes, muito próximos dos conseguidos através da simulação do próprio modelo do pêndulo inverso. A única restrição foi o tempo de simulação utilizado, porém esta não é devido a rede neural diretamente, mas devido à singularidades do modelo do sistema adotado neste trabalho.

6.2 – RESULTADOS DO CONTROLADOR NEURAL:

Neste item serão discutidos e mostrados os resultados conseguidos do acoplamento do controlador neural ao sistema pendular inverso não-linear. Além disso,

também será mostrado o procedimento para a validação deste procedimento para o projeto de um controlador.

O treinamento da rede neural para o substituir o controlador se deu em 38 épocas, ao atingir um valor de 0.27 para a soma quadrática do erro. Abaixo está mostrado um gráfico do comportamento da mesma durante o treinamento.

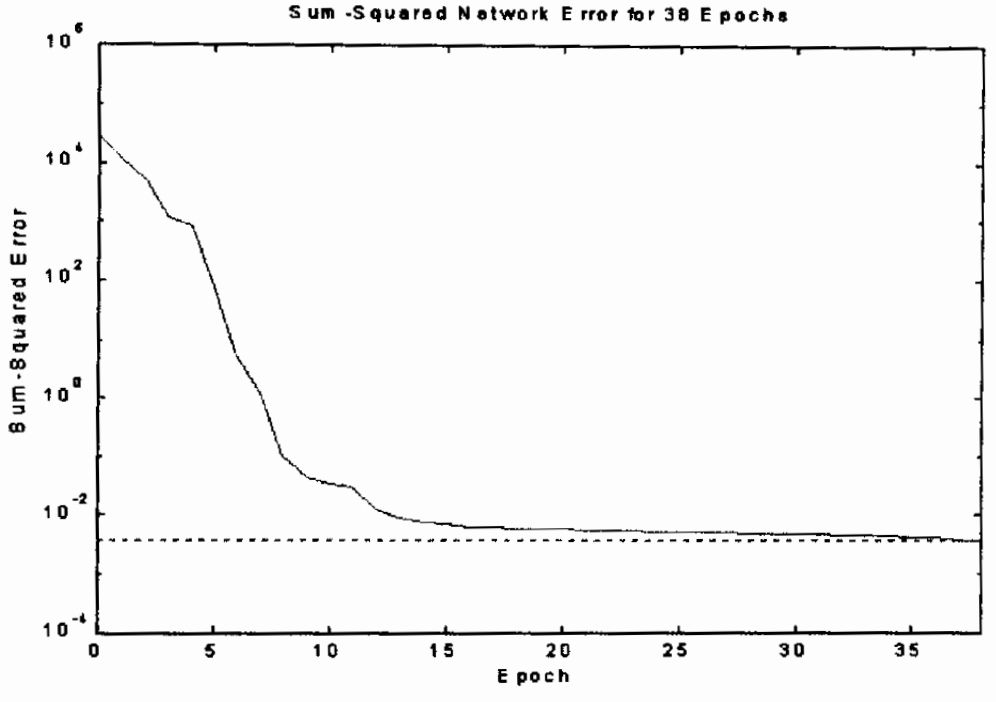


Fig 6.9 – Gráfico da soma do erro quadrático x número de épocas

Para a realização deste teste foi gerado um diagrama de blocos no SIMULINK, que simula o sistema pendular em malha fechada juntamente com o controlador. Este diagrama foi montado utilizando o modelo adotado neste trabalho com os valores físicos, como massa do pêndulo e comprimento da haste, do protótipo que estava sendo construído. Para uma melhor visualização do comportamento do sistema pendular aclopado ao neurocontrolador , foi utilizado um bloco de animação do pêndulo inverso sobre carro que estava disponível em um exemplo da *Toolbox* de Redes Neurais do MATLAB. Este bloco de

animação desenha o pêndulo como se o seu peso fosse distribuído na haste, porém deve ser lembrado que o modelo adotado neste projeto assume que a massa está concentrada na extremidade superior da mesma, sendo este modelo que está sendo controlado durante o teste.

Também foram colocados neste diagrama, blocos com os quais seja possível monitorar o comportamento de sinais que são importantes para o teste. Estão sendo monitorados a posição 'y' desejada, a posição 'y' conseguida pelo sistema em malha fechada com o controlador, o ângulo θ conseguido e o sinal de controle, que é o próprio sinal de saída da rede neural (controlador).

A rede neural responsável pelo controle está representada em dois blocos, onde o primeiro simula a camada oculta e o segundo a camada final. Os parâmetros destes blocos são os pesos para cada camada adquiridos durante a fase de treinamento.

O diagrama de blocos do teste está representado na figura a seguir:

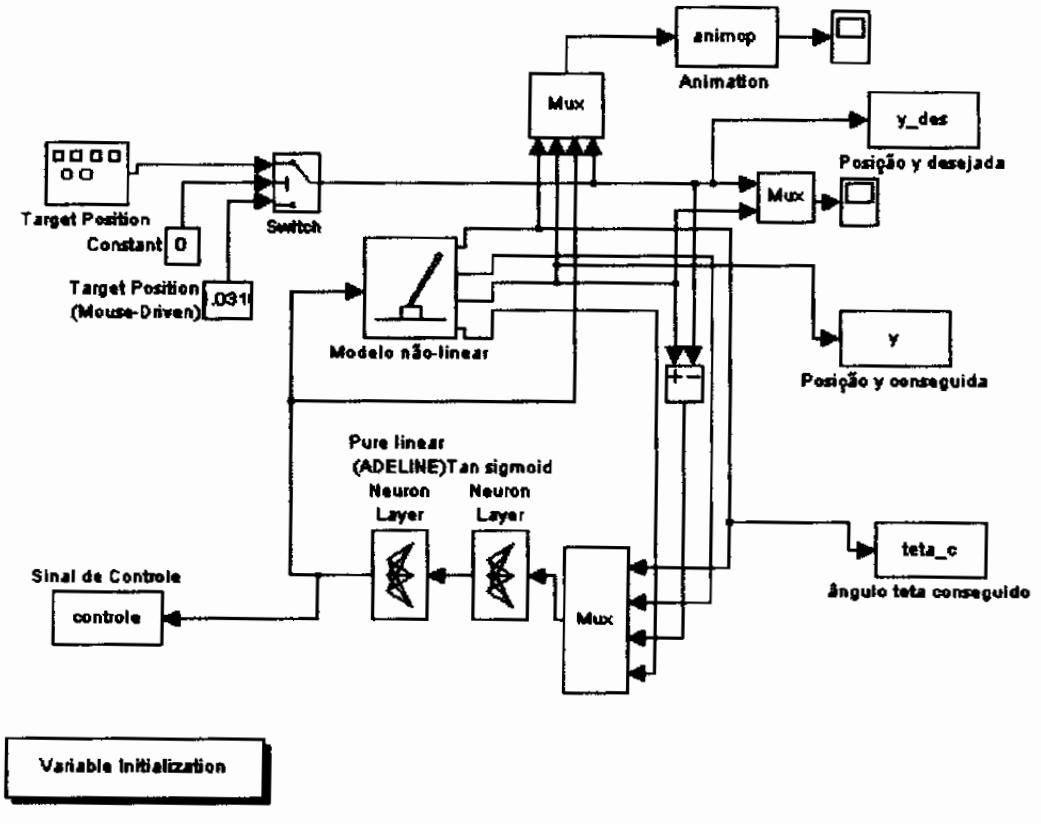


Fig 6.10 – Diagrama de blocos do sistema pendular invertido em malha fechada acoplado ao controlador

Ao simular o sistema mostrado acima, se consegue uma janela que permite a visualização do funcionamento do pêndulo inverso, quando o carro tenta seguir uma trajetória, que no caso deste teste foi escolhida como uma onda quadrada de amplitude de 0.5 m e de frequência de 0.625 Hz. Porém, deve ficar claro que poderia ser utilizado qualquer sinal como trajetória, desde que este tivesse sua amplitude dentro do intervalo no qual a rede (controlador) foi treinada.

A seguir está a posição do sistema pendular em um determinado instante da simulação, que poderá ser observado através da janela mencionada:

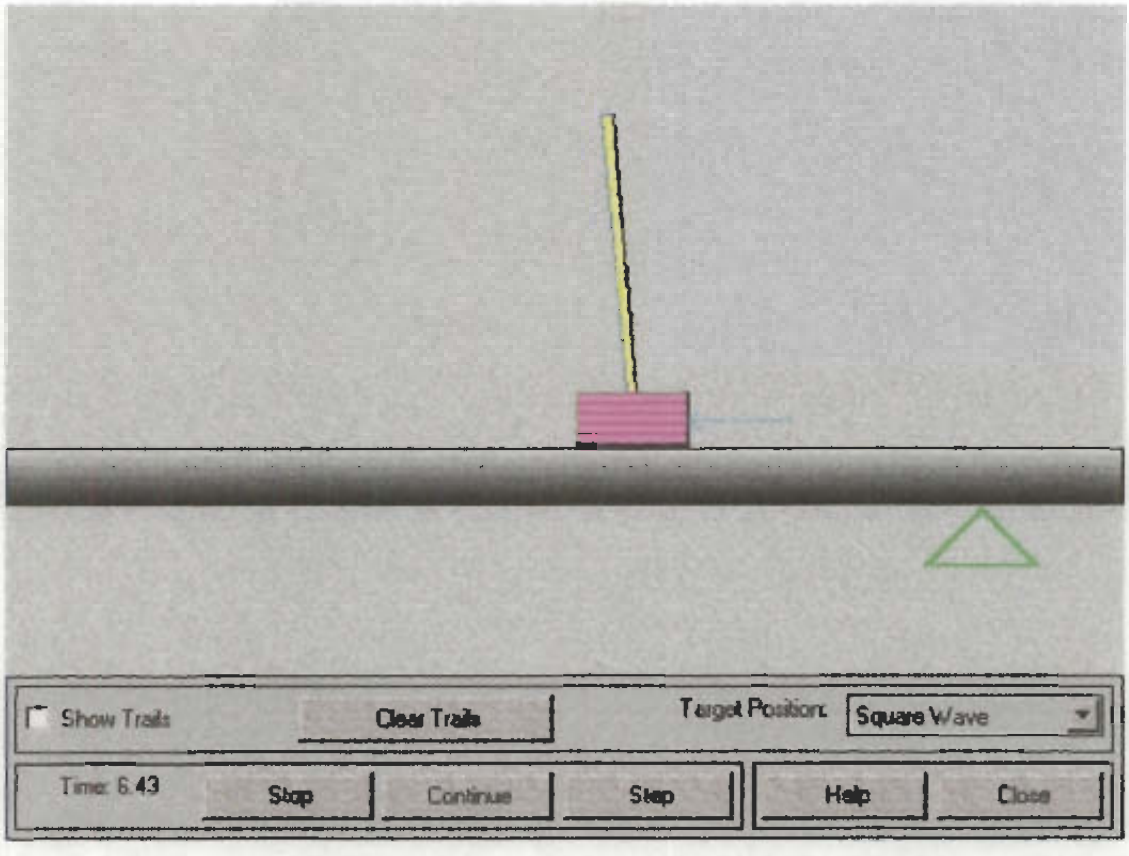


Fig 6.11 – Janela para visualização do comportamento do sistema pendular invertido com o controlador em malha fechada

Para uma melhor visualização do resultado do controlador, ao seguir uma trajetória, serão mostrados os gráficos da comparação entre o deslocamento do carro e o deslocamento desejado. Serão também apresentados gráficos dos valores do ângulo teta para a verificação do equilíbrio da haste em torno de sua posição de equilíbrio instável ($\theta = 0$, haste para cima, perpendicular ao plano do carro).

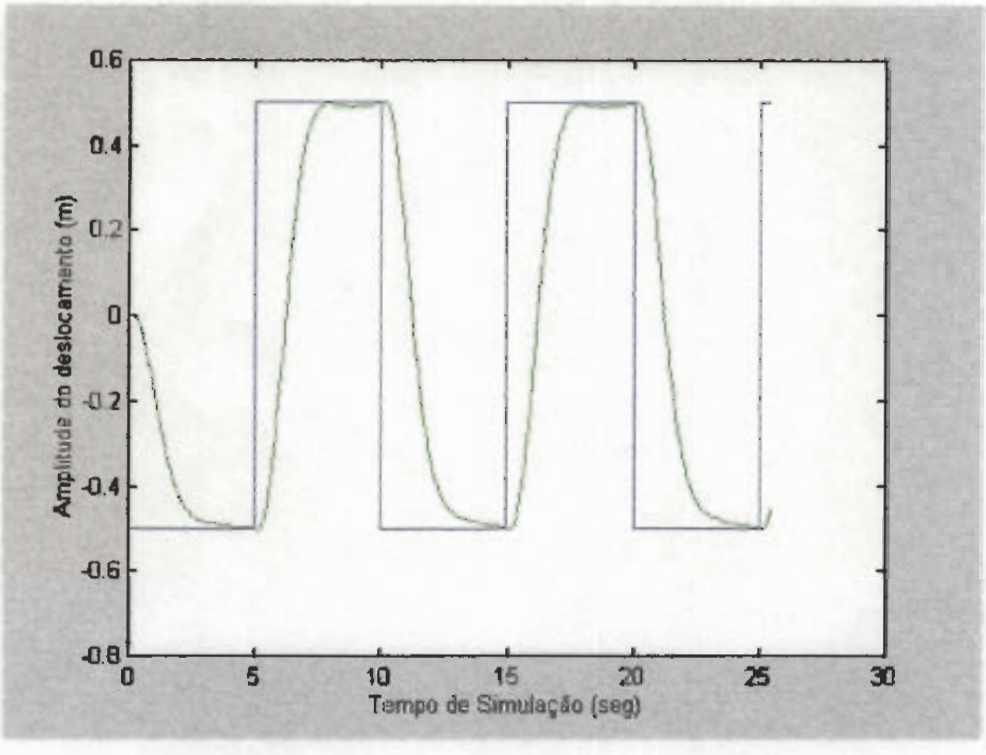


Fig 6.12 – Gráfico de comparação entre o deslocamento desejado (curva azul) e o deslocamento realizado pelo carro (curva verde)

Observando o gráfico acima pode-se notar um certo atraso na resposta do sistema ao acompanhar a trajetória desejada. Porém mesmo tendo este atraso de resposta, este consegue seguir o sinal arbitrado como era requerido por um dos objetivos deste trabalho.

Uma particularidade observada foi que ao aumentar a amplitude do sinal que se deseja seguir (trajetória) a resposta do sistema se torna mais rápida, porém é verificado o aparecimento de um sobrepasso proporcional a amplitude da trajetória.

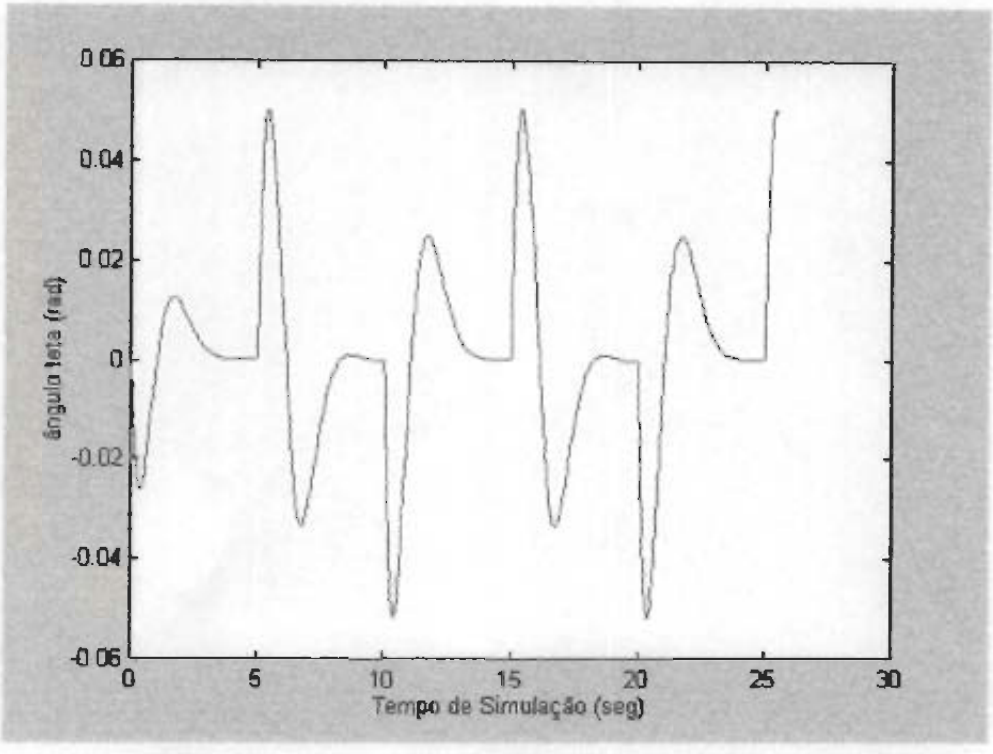


Fig 6.13 – Gráfico da amplitude do ângulo de teta durante a simulação.

Pode ser observado que o ângulo teta fica oscilando em valores muito próximos do zero radiano, como era o desejado, pois a haste deve ter uma posição em torno de seu eixo de equilíbrio instável. Estas oscilações se devem ao próprio movimento do carrinho, e o ajudam no seu deslocamento para cumprir a trajetória requerida e para manutenção do equilíbrio da haste durante a simulação.

Como uma informação extra será mostrado um gráfico do comportamento do sinal de controle.

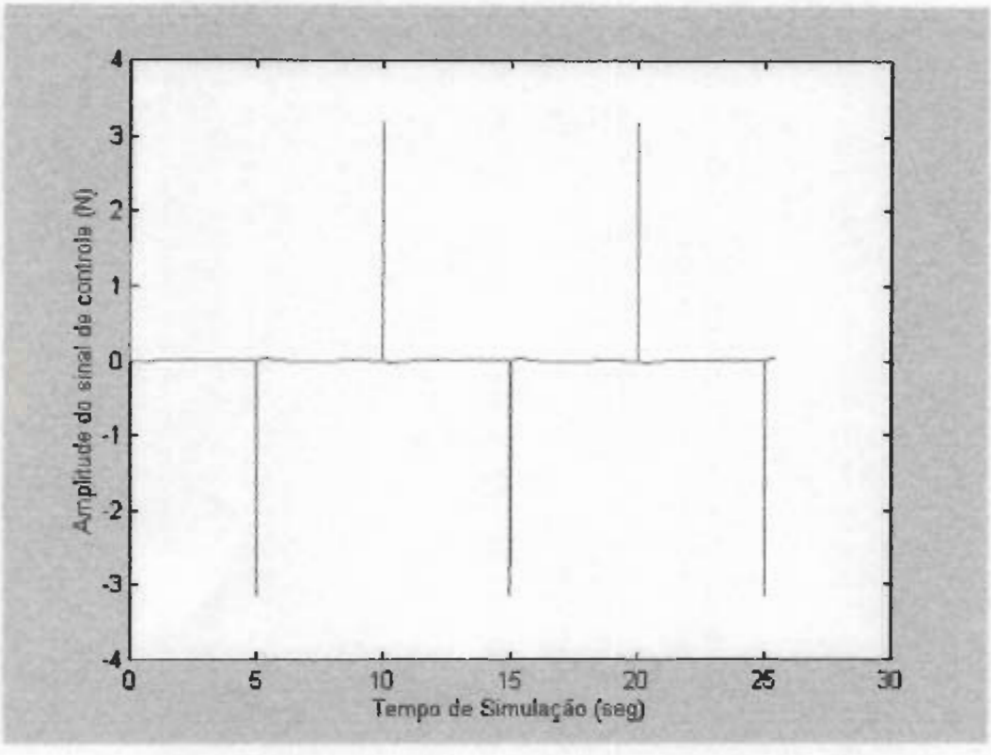


Fig 6.14 – Gráfico da amplitude do sinal de controle durante a simulação

Ao observar o gráfico pode-se notar os instantes de atuação do controlador para impulsionar o carro, para que este fosse capaz de seguir a trajetória pré-determinada durante o teste

Pode-se então notar ao examinar os gráficos que o controlador neural permite que o sistema se torne estável e capaz de seguir uma trajetória y desejada (trajetória do carro). Dessa forma, pode-se concluir que o trabalho teve seus objetivos alcançados, mesmo não seguindo o procedimento sugerido no seu início.

7 – CONCLUSÃO

Quando se observa o ambiente em que o ser humano habita, pode ser notada uma grande quantidade de dispositivos, criados por ele mesmo, que possuem os mais diversos objetivos; desde da fabricação de tecido até a construção de peças altamente sofisticadas, que são utilizadas nos ônibus espaciais. Porém na maioria dos casos, estes sistemas, ao serem construídos, precisam de outros sistemas para realizar o controle de seu funcionamento, ou até mesmo para corrigir certos comportamentos indesejados durante a sua operação. Estes sistemas, que são acoplados a planta original para cumprir estas funções, são denominados *controladores*.

Os controladores se fazem necessários, principalmente, quando os sistemas, aos quais são acoplados, são não-lineares (cap. 02). Como foi discutido, todos os sistemas reais são não-lineares, qualquer característica linear se apresenta apenas em intervalos de operação. Neste trabalho, foi estudado um sistema com características muito peculiares, especialmente quanto a sua estabilidade. O *Pêndulo Inverso com Carro* é um sistema muito complexo, que precisou de um estudo detalhado, para que todo o seu funcionamento fosse realmente compreendido.

O objetivo deste trabalho foi a identificação e o projeto de um controlador utilizando redes neurais artificiais, que mantivesse a haste do pêndulo equilibrada em torno sua posição de equilíbrio instável (haste perpendicular ao plano do carro, com sua extremidade livre voltada para cima), enquanto o sistema pendular inverso seguisse uma trajetória arbitrada (onda quadrada ou senoidal, por exemplo).

Para uma melhor compreensão do sistema do pêndulo inverso sobre carro, foi realizada uma análise das suas características de controlabilidade e da observabilidade (cap.02), além de uma análise de estabilidade. Foi mostrado que apesar de ser um sistema instável, o sistema pendular invertido é observável e controlável.

Para que fosse possível o entendimento do procedimento de utilização das redes neurais artificiais, tanto para a identificação como para o controle, foi feita uma discussão sobre seus conceitos teóricos principais (cap. 03). Foram também apresentados vários métodos de identificação e de projeto de controladores utilizando redes neurais (cap. 04). No caso apresentado neste relatório, foi utilizado o método de identificação não-paramétrica, baseado nos dados de entrada que determinam o comportamento do sistema a ser identificado; enquanto que o projeto do neurocontrolador foi desenvolvido com base na rede neural que identifica o modelo não-linear do processo.

Para a identificação do sistema de pêndulo inverso sobre carro foi utilizada uma rede neural do tipo *Perceptron Multicamadas*, constituída de 15 neurônios na camada oculta e 4 neurônios na camada de saída. Os neurônios da camada de saída tinham como resultados os valores das variáveis de estado do sistema após 10ms de intervalo de operação do mesmo. Para proporcionar um treinamento mais rápido foi utilizado o algoritmo *Backpropagation* baseado no método de Levenberg-Marquardt para a determinação da expressão para o ajuste de pesos da rede neural.

Observando os resultados dos testes da rede neural de identificação foi possível notar seu funcionamento satisfatório, dentro dos limites que foram impostos a mesma durante o seu treinamento, ou seja, os valores proporcionados pela rede neural foram muito próximos daqueles obtidos através das expressões do próprio modelo não-linear. Este objetivo do trabalho foi alcançado com sucesso, a rede neural do tipo *Perceptron* Multicamadas foi capaz de identificar o modelo não-linear do pêndulo inverso com carro. Estes resultados corroboraram a própria teoria de redes neurais, já que estas têm como um de seus objetivos o de identificar processos complexos.

- Para a determinação do neurocontrolador foi utilizado o método citado anteriormente (projeto do neurocontrolador baseado no modelo - cap. 05), no qual o treinamento da rede, que tomaria o lugar do controlador, se faria através da retropropagação do erro entre a saída obtida pela rede e a obtida através de um modelo de referência, sendo utilizado neste

Conclusão

procedimento os pesos da rede de identificação. Este modelo de referência tinha como função representar o comportamento que se desejaria que o sistema pendular inverso tivesse quando em malha fechada com o controlador. Dessa forma, a identificação e o projeto do neurocontrolador se tornariam procedimentos complementares. Porém, não foi possível, no tempo de execução do projeto, obter um modelo de referência que fosse capaz de, nos testes do neurocontrolador, manter o pêndulo inverso equilibrado durante o acompanhamento de um trajetória.

Como procedimento alternativo, para a determinação de um neurocontrolador para o pêndulo inverso com carro, foram utilizados conceitos de *lógica fuzzy* para a montagem de um conjunto de treinamento, que contivesse as informações necessárias para que a rede neural pudesse impor o comportamento acima descrito. Foram utilizadas funções de pertinência (não-lineares) e regras *fuzzy*, para determinar um sinal de controle (força aplicada ao carro) baseado nos valores das variáveis de estado, em um determinado instante. Com este conjunto de treinamento foi possível treinar uma rede neural do tipo *Perceptron* Multicamadas, que foi acoplada ao sistema original. Esta rede neural era composta de 15 neurônios na camada oculta e 1 neurônio na camada de saída, e seu treinamento foi o mesmo utilizado na rede neural de identificação. A resposta da camada de saída era exatamente a força que deveria ser aplicada ao carro do pêndulo inverso, para que este seguisse uma trajetória e mantivesse sua haste equilibrada (posição de equilíbrio instável).

Neste procedimento alternativo, a identificação e o projeto do neurocontrolador se tornaram processos independentes.

Os resultados do funcionamento do neurocontrolador obtido através deste procedimento foram satisfatórios. O sistema pendular invertido com o neurocontrolador, em malha fechada, foi capaz de seguir uma trajetória definida, mesmo apresentando um certo atraso na resposta, mantendo a haste equilibrada (cap. 06). A manutenção do equilíbrio da haste foi confirmada através do monitoramento que mostrava as pequenas amplitudes assumidas pelo deslocamento angular durante os testes. Também foi possível observar a forma do sinal de controle e seus instantes de atuação.

Dessa forma, pode-se notar que os objetivos deste trabalho foram alcançados, a identificação e o controle do pêndulo inverso com carro, mesmo não seguindo os procedimentos que foram propostos no início de sua realização.

Uma provável continuação deste projeto seria a retomada da construção do sistema pendular, e a realização de testes deste dispositivo com o neurocontrolador desenvolvido. Uma outra sugestão para a continuação deste estudo seria a aplicação deste método de neurocontrole no seguimento de trajetórias no lançamento de foguetes no espaço, já que a teoria do pêndulo inverso sobre carro é a base para o estudo deste problema prático.

Referências Bibliográficas

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Ogata, Katsuhiko; *Engenharia de Controle Moderno*; 2ª Edição, Editora Prentice-Hall do Brasil, 1993.

[2] Bauchspiess, Adolfo; *Notas de Aula da Disciplina de Teoria de Sistemas Lineares*, 1998.

[3] Gupta, Madan M. e Sinha, Naresh K.; *Intelligent Control Systems, Theory and Applications*; IEEE Press, 1996.

[4] Chen, Chi-Tsong; *Linear Systems Theory and Design*, CBS College Publishing, 1984.

[5] Russel, Stuart e Norvig, Peter; *Artificial Intelligence, a Modern Approach*; Prentice-Hall, 1995.

[6] Tafner, Malcon A., Xerez, Marcos de e Rodrigues Filho, Ilson W.; *Redes Neurais Artificiais, Introdução e Princípios de Neurocomputação*; Editora EKO e Editora da FURB, 1996.

[7] Kovács, Zsolt L.; *O Cérebro e a sua Mente, uma Introdução à Neurociência*; Edição Acadêmica, 1997.

[8] Kovács, Zsolt L.; *Redes Neurais Artificiais, Fundamentos e Aplicações*; Edição Acadêmica, 1996.

[9] Bauchspiess, Adolfo, Drummond, Adriana de C. e Romariz, Alexandre R.S.; *Servocontrole Não-Linear Auto-Sintonizado por Redes Neurais de Base Radial*; 3° Simpósio Brasileiro de Automação Inteligente, pag 430, Setembro 1997, Vitória, ES.

[10] Kasabov, Nikola K.; *Foundations of Neural Network, Fuzzy Systems and Knowledge Engineering*; MIT Press, 1996.

[11] Loesch, Claudio e Sari, Solange T.; *Redes Neurais Artificiais, Fundamentos e Modelos*; Editora da FURB, 1996.

[12] Drummond, Adriana de C., Bauchspiess, Adolfo e Romariz, Alexandre R.S.; *Comparação de Desempenho de Diversas Arquiteturas de Redes Neurais Artificiais para o Servocontrole Não-Linear Auto-Sintonizado*; XII Congresso Brasileiro de Automática, Setembro 1998, Uberlândia, MG.

[13] Harris, C.J., Moore, C.G. e Brown, M.; *Intelligent Control – Aspects of Fuzzy Logic and Neural Nets*; World Scientific, 1994.

[14] Suykens, Johan A.K., Vanderwalle, Joos P.L. e De Moor, Bart L.R.; *Artificial Neural Networks for Modeling and Control of Non-Linear Systems*; Kluwer Academic Publishers, 1997.

[15] Drummond, Adriana de C.; Bauchspiess, Adolfo; Oliveira, Kleyton C.; *Estudo do Controle do Pêndulo Inverso sobre Carro utilizando Redes Neurais de Base Radial*; IV Congresso Brasileiro de Redes Neurais, Julho de 1999, São José dos Campos – SP.

[16] Demuth, Howard e Beale, Mark; *Neural Network Toolbox- User's Guide, For Use with MATLAB*; The MathWorks, Inc.; Janeiro 1994.

[17] *Help da Toolbox de Lógica Fuzzy do MATLAB, versão 5.0.*

[18] Rosenblat, Frank; *Principles of Neurodynamics*; New York, Spartan Books, 1959.

[19] Minsky, Marvin e Papert, Seymour; *Perceptrons*; MIT Press, Cambridge, 1969.

[20] Sandri, Sandra; *Introdução à Lógica Fuzzy*; Apostila do Minicurso M2 do 3º Simpósio Brasileiro de Automação Inteligente, Setembro de 1997.

APÊNDICE A – INTRODUÇÃO SOBRE LÓGICA FUZZY

Neste apêndice será feita uma análise qualitativa sucinta sobre Lógica *Fuzzy* , apenas para apresentar a ideia desta teoria.

A imprecisão e a incerteza são os dois principais aspectos da imperfeição da informação. Estas duas características são intrinsicamente ligadas e opostas entre si, ou seja, quando a incerteza aumenta a imprecisão diminui e vice-versa. As teorias mais utilizadas para tratas estas dois aspectos são a teoria de conjunto e a teoria das probabilidades. Porém, estas teorias nem sempre conseguem exprimir toda a gama de informação fornecida pelos seres humanos. A teoria de conjunto não consegue tratar o aspecto vago da informação, e a teoria das probabilidades é mais voltada para o tratamento de informações frequentistas (esta teoria é aditiva).

A teoria dos **conjuntos *fuzzy*** foi desenvolvida por Lofti Zadeh, a partir de 1965, para tratar o aspecto vago da informação. Além disso em 1978, Zadeh começou a desenvolver a ***teoria das possibilidade*** , que trata da incerteza da informação (esta teoria não é aditiva). A teoria de conjuntos tradicional pode ser considerada como um caso particular da teoria de conjuntos *fuzzy*.

Estas duas teorias citadas são intimamente ligadas, possibilitando o tratamento da imprecisão e da incerteza da informação dentro de um mesmo ambiente formal. A teoria de conjuntos de conjuntos *fuzzy*, quando utilizada em um contexto lógico (sistemas baseado em conhecimento) é denominada de ***Lógica Fuzzy***.

A teoria de conjuntos *fuzzy* tem como objetivo graduações na pertinência de um elemento a uma dada classe, isto significa, que permite a um elemento pertencer com maior ou menor intensidade àquela classe. Uma definição mais formal é dada abaixo.

- Dado um universo de discurso Ω, um conjunto *fuzzy* A de Ω é definido por uma função de pertinência $\mu_A : \Omega \rightarrow [0,1]$, que associa a cada elemento x de Ω o grau $\mu_A(x)$, com o qual x pertence a A. $\mu_A(x)$ indica o grau de compatibilidade entre x e o conceito expresso por A:

  - $\mu_A(x) = 1$, x é completamente compatível com A;
  - $\mu_A(x) = 0$, x é completamente incompatível com A;
  - $0 < \mu_A(x) < 1$, x é parcialmente compatível com A, com grau $\mu_A(x)$.

Como exemplo, pode ser citado o modelamento do conceito 'alto'. Pode-se arbitrar que uma pessoa com mais de 1,75m é considerada alta, enquanto que uma pessoa com menos de 1,60m é considerada baixa. Dessa forma, uma pessoa que tenha sua altura entre 1,60 e 1,75m será considerada mais alta quanto mais sua altura esteja próxima de 1,75. Devido a essa característica de imprecisão, o conceito 'alto', pode ser modelado por um conjunto *fuzzy* [20]. Portanto, tem-se:

- $\mu_A(x) = 1$, x > 1,75m;

- $\mu_A(x) = 0$, x < 1,60m;
- $\mu_A(x) = (x-1,60)/0,15$; $1,60 \leq x \leq 1,75$.

A seguir está um gráfico para a visualização do conjunto *fuzzy*:

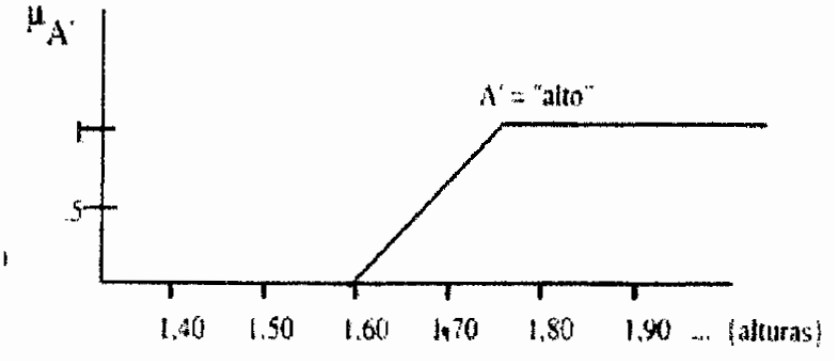


Fig A.1 – Conjunto *fuzzy* para o exemplo do conceito de 'alto' (Sandri, 1997)

Os conjuntos com os quais se trabalha usualmente (teoria de conjuntos) são denominados, no caso *fuzzy*, de 'crisp', e são definidos como conjuntos *fuzzy* onde a função de pertinência assume apenas os valores 0 e 1.

Na teoria, mostrada neste apêndice, o complemento, a intersecção e a união são implementadas por famílias de operações. A intersecção é implementada por uma família de operações denominada ***t-normas***, enquanto que a união é implementada por uma família de operações denominada ***t-conormas***. O conjunto das t-normas e das t-conormas formam as ***normas triangulares*** [20].

A seguir está mostrada uma tabela que lista as principais operações de t-normas e de t-conormas:

Tabela A.1 – Principais t-normas e t-conormas (Sandri, 1997)

t-norma	*t-conorma*	*negação*	*nome*
min(a, b)	max(a, b)	1 - a	Zadeh
a . b	a + b - ab	1 - a	probabilista
max(a + b - 1, 0)	min(a + b, 1)	1 - a	Lukasiewicz
{ a, se b = 1	{ a, se b = 0	1 - a	Weber
{ b, se a = 1	{ b, se a = 0		
{ 0 senão	{ 1 senão		

Como exemplo, foram escolhidas as operações de Zadeh para uma visualização de dois subconjuntos *fuzzy*. Considerando os subconjuntos *fuzzy* A e B em X, o complemento A^C, a intersecção $A \cap B$ e a união $A \cup B$ são determinados por [20]:

$$\begin{aligned} \mu_{A^C}(x) &= 1 - \mu_{A^C}(x) \\ \mu_{A \cap B}(x) &= \min(\mu_A(x), \mu_B(x)) \\ \mu_{A \cup B}(x) &= \max(\mu_A(x), \mu_B(x)) \end{aligned} \qquad \text{(A.1)}$$

Os gráficos que representam estas operações estão mostrados no quadro a seguir, onde os dois subconjuntos *fuzzy* estão representando pessoas em duas diferentes faixas etárias.

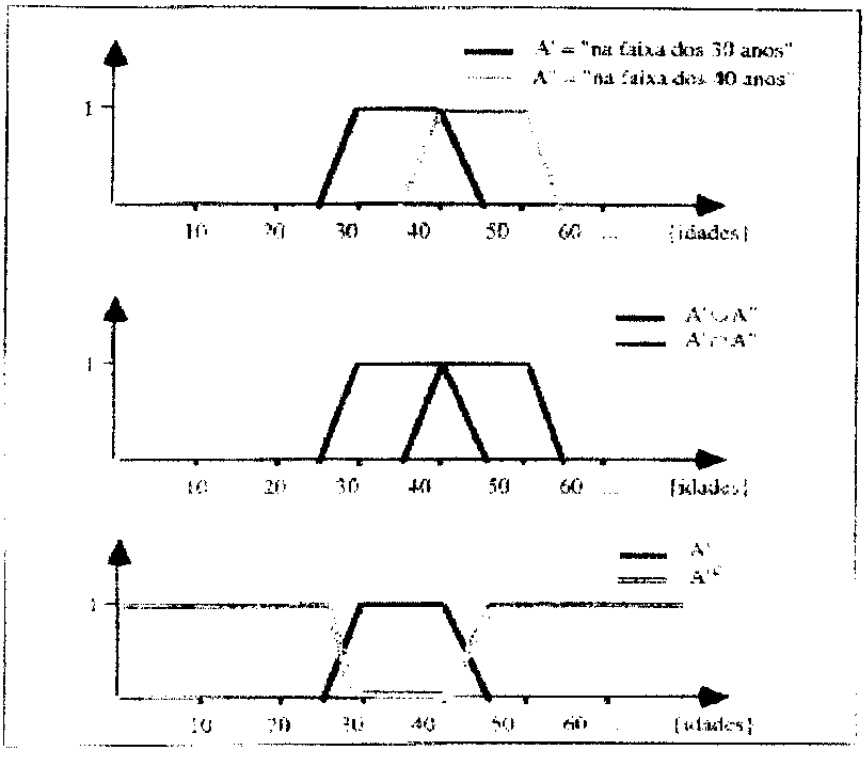


Fig A.2 – Exemplos de intersecção, união e complementação de dois subconjuntos *fuzzy* em um mesmo universo (Sandri, 1997)

As normas triangulares respeitam as propriedades da comutatividade, da associatividade, da monotonicidade e do elemento neutro.

O que se denomina *lógica fuzzy* é a maneira de se implementar o raciocínio aproximado no contexto da teoria de conjuntos *fuzzy*. São criadas regras que relacionam as diversas funções de pertinência que são importantes para um determinado sistema. Estas regras utilizam as operações mostradas anteriormente (união, intersecção e complementação) para relacionar os diversos conjuntos *fuzzy*.

Este texto foi apenas uma pequena amostra da teoria de lógica *fuzzy*. Esta introdução poderia ser extendida por páginas, pois existem diversos conceitos importantes e aplicações a serem discutidas. Todas as informações (gráficos e tabelas)descritas neste apêndice foram retiradas da *Apostila do Minicurso M1 do 3º Simpósio Brasileiro de Automação Inteligente, Setembro de 1997.*

APÊNDICE B – CONJUNTO DE REGRAS FUZZY

As regras de controle *fuzzy* listadas abaixo foram aquelas utilizadas para a determinação do vetor de saída para o treinamento da rede neural do neurocontrolador, onde:

teta1 – deslocamento angular da haste do pêndulo inverso;
vteta2 – velocidade angular da haste do pêndulo inverso;
desloc3 – deslocamento horizontal do carro do pêndulo inverso; e
vdesloc4 – velocidade do deslocamento horizontal do carro do pêndulo inverso.

Os termos tetamf1 e tetamf2 são as duas funções de pertinência que estão relacionadas aos valores das variáveis teta. O mesmo princípio é aplicado para as outras variáveis de estado. Baseado nestas regras, valores *fuzzy* (conjuntos *fuzzy*) são determinados para a saída 'u', que são representados pelos termos: umf1, umf2, ..., umf16.

1. If (teta1 is teta1mf1) and (vteta2 is vteta2mf1) and (desloc3 is desloc3mf1) and (vdesloc4 is vdesloc4mf1) then (u is umf1) (1)

2. If (teta1 is teta1mf1) and (vteta2 is vteta2mf1) and (desloc3 is desloc3mf1) and (vdesloc4 is vdesloc4mf2) then (u is umf2) (1)

3. If (teta1 is teta1mf1) and (vteta2 is vteta2mf1) and (desloc3 is desloc3mf2) and (vdesloc4 is vdesloc4mf1) then (u is umf3) (1)

4. If (teta1 is teta1mf1) and (vteta2 is vteta2mf1) and (desloc3 is desloc3mf2) and (vdesloc4 is vdesloc4mf2) then (u is umf4) (1)

5. If (teta1 is teta1mf1) and (vteta2 is vteta2mf2) and (desloc3 is desloc3mf1) and (vdesloc4 is vdesloc4mf1) then (u is umf5) (1)

6. If (teta1 is teta1mf1) and (vteta2 is vteta2mf2) and (desloc3 is desloc3mf1) and (vdesloc4 is vdesloc4mf2) then (u is umf6) (1)

7. If (teta1 is teta1mf1) and (vteta2 is vteta2mf2) and (desloc3 is desloc3mf2) and (vdesloc4 is vdesloc4mf1) then (u is umf7) (1)

8. If (teta1 is teta1mf1) and (vteta2 is vteta2mf2) and (desloc3 is desloc3mf2) and (vdesloc4 is vdesloc4mf2) then (u is umf8) (1)

9. If (teta1 is teta1mf2) and (vteta2 is vteta2mf1) and (desloc3 is desloc3mf1) and (vdesloc4 is vdesloc4mf1) then (u is umf9) (1)

10. If (teta1 is teta1mf2) and (vteta2 is vteta2mf1) and (desloc3 is desloc3mf1) and (vdesloc4 is vdesloc4mf2) then (u is umf10) (1)

11. If (teta1 is teta1mf2) and (vteta2 is vteta2mf1) and (desloc3 is desloc3mf2) and (vdesloc4 is vdesloc4mf1) then (u is umf11) (1)

12. If (teta1 is teta1mf2) and (vteta2 is vteta2mf1) and (desloc3 is desloc3mf2) and (vdesloc4 is vdesloc4mf2) then (u is umf12) (1)

13. If (teta1 is teta1mf2) and (vteta2 is vteta2mf2) and (desloc3 is desloc3mf1) and (vdesloc4 is vdesloc4mf1) then (u is umf13) (1)

14. If (teta1 is teta1mf2) and (vteta2 is vteta2mf2) and (desloc3 is desloc3mf1) and (vdesloc4 is vdesloc4mf2) then (u is umf14) (1)

15. If (teta1 is teta1mf2) and (vteta2 is vteta2mf2) and (desloc3 is desloc3mf2) and (vdesloc4 is vdesloc4mf1) then (u is umf15) (1)

16. If (teta1 is teta1mf2) and (vteta2 is vteta2mf2) and (desloc3 is desloc3mf2) and (vdesloc4 is vdesloc4mf2) then (u is umf16) (1)

APÊNDICE C – VISUALIZAÇÃO DAS JANELAS DA ROTINA 'FINAL'

Neste apêndice são mostradas algumas das janelas geradas pela rotina ***final.*** Esta rotina foi desenvolvida no ambiente MATLAB, e tem a finalidade de simplificar o uso das diversas funções utilizadas para o treinamento das redes neurais artificiais para o caso de identificação de sistemas. As janelas são mostradas na ordem sequencial de apresentação durante o funcionamento da rotina.

A primeira janela está mostrada a seguir:

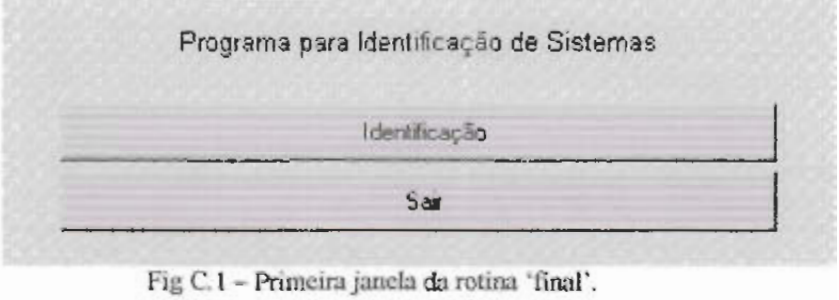


Fig C.1 – Primeira janela da rotina 'final'.

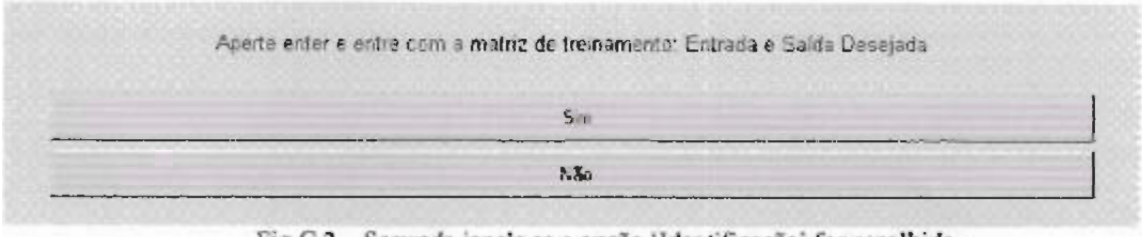


Fig C.2 – Segunda janela se a opção 'Identificação' for escolhida.

Entre com o nome da matriz de entrada e aperte enter

inicial2=[1 2 3 4; 5 6 7 8]

Enter com o nome da matriz de saída desejada e aperte enter

final_nlinear=[1 0 1 0; 0 0 1 2]

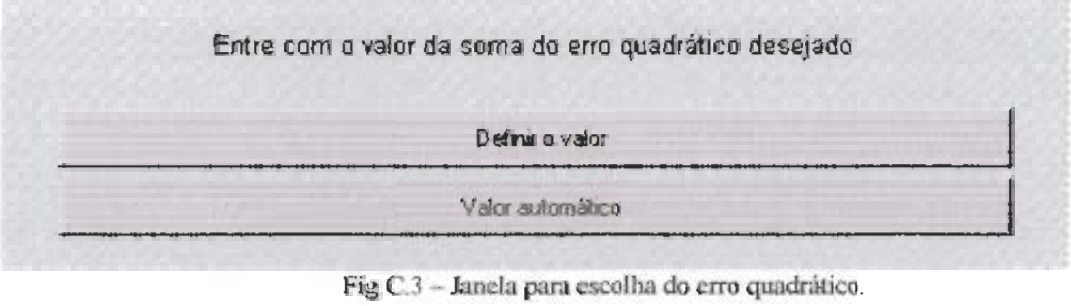


Fig C.3 – Janela para escolha do erro quadrático.

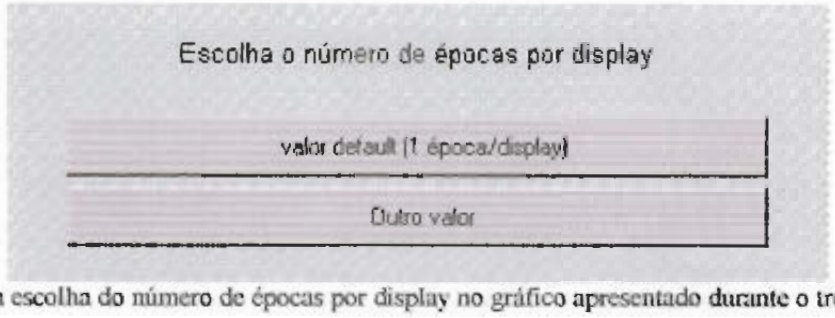


Fig C.4 – Janela para escolha do número de épocas por display no gráfico apresentado durante o treinamento

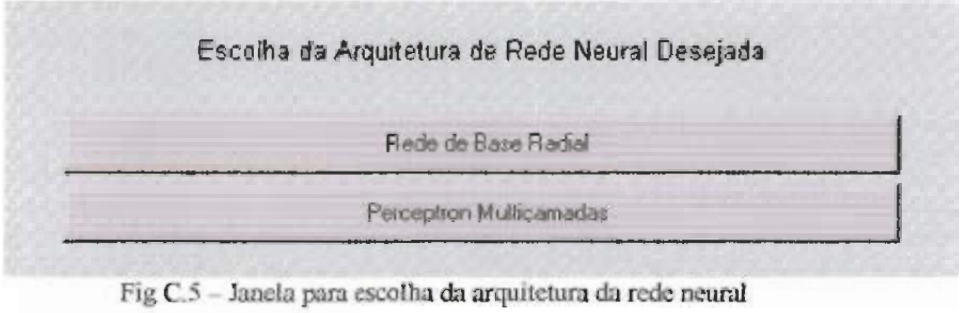


Fig C.5 – Janela para escolha da arquitetura da rede neural

Escolhendo a *Perceptron* Multicamadas, tem-se as seguintes janelas (caso fosse escolhida a rede de base radial as janelas seriam diferentes):

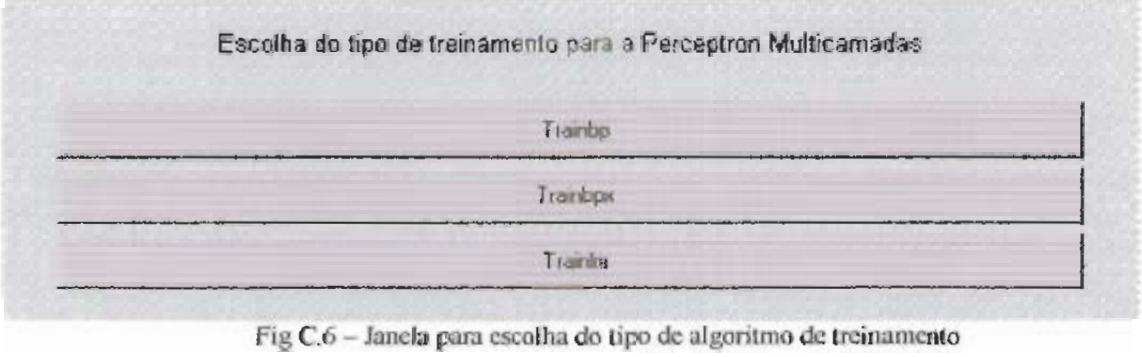


Fig C.6 – Janela para escolha do tipo de algoritmo de treinamento

Escolhendo o algoritmo de treinamento *trainlm* tem-se as seguintes janelas (caso fosse escolhido outro algoritmo as janelas seriam diferentes):

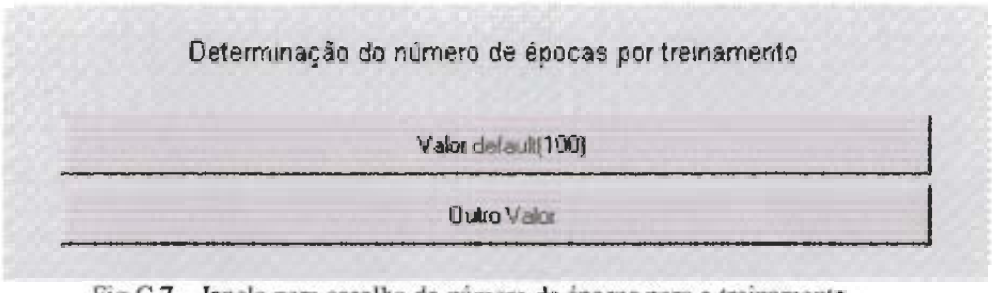


Fig C.7 – Janela para escolha do número de épocas para o treinamento

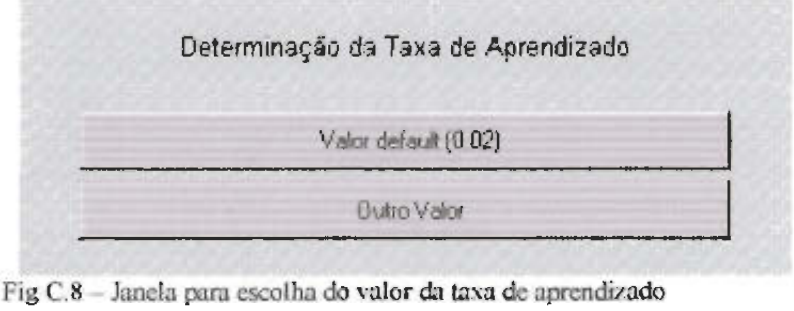


Fig C.8 – Janela para escolha do valor da taxa de aprendizado

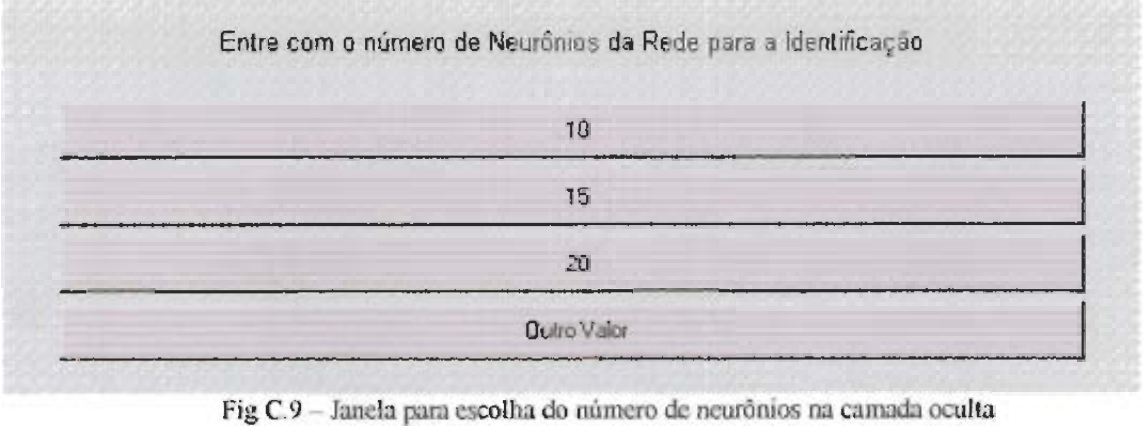


Fig C.9 – Janela para escolha do número de neurônios na camada oculta

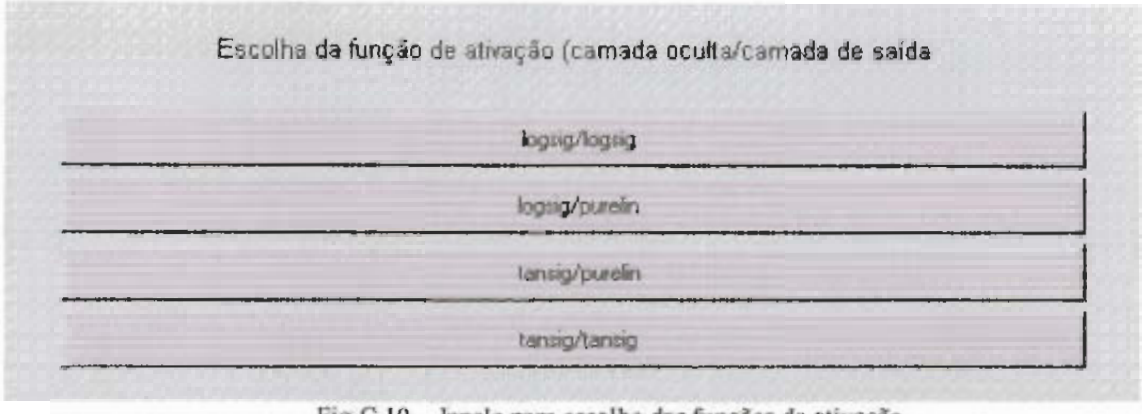


Fig C.10 – Janela para escolha das funções de ativação

Gráfico obtido durante o treinamento da rede utilizada no exemplo deste apêndice:

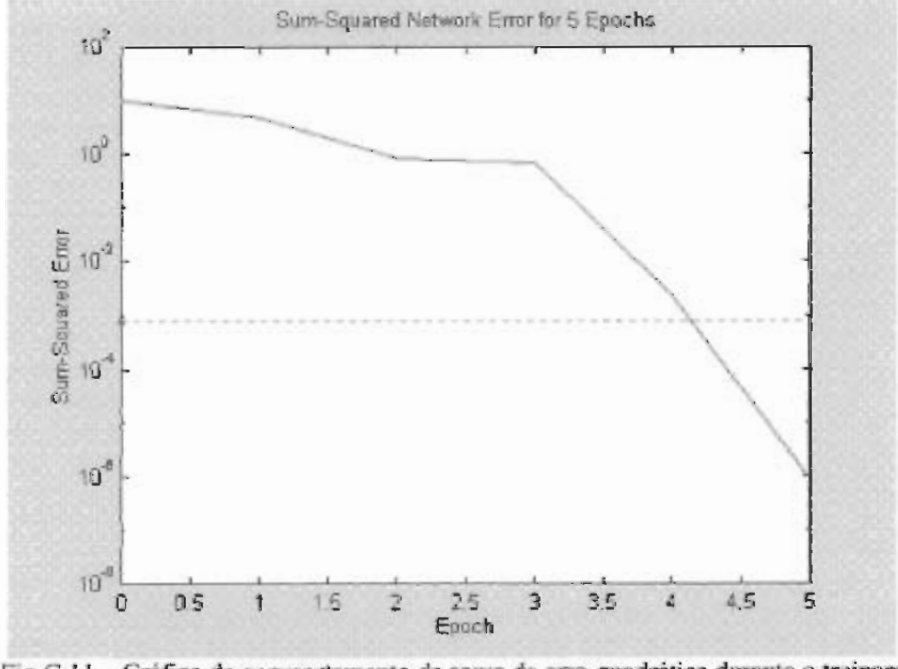


Fig C.11 – Gráfico do comportamento da soma do erro quadrático durante o treinamento

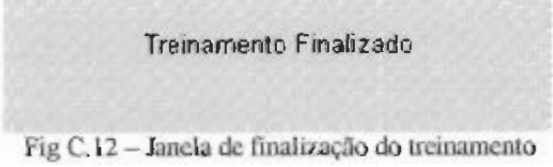


Fig C.12 – Janela de finalização do treinamento

Quando na primeira janela da rotina é escolhida a opção sair, aparece na tela de trabalho:

Fig C.13 – Janela de finalização da rotina

APÊNDICE D – LISTAGEM DAS ROTINAS

Neste apêndice foram listadas todas as rotinas utilizadas neste trabalho. Algumas delas são sub-rotinas, ou seja, são 'chamadas' dentro de outras rotinas.

- ***Observabilidade:***

```
% Verificação de Observabilidade:

k1=exist('A','var');
k2=exist('C','var');

if (k1==0)&(k2==0)
disp('CUIDADO: Defina as matrizes A, B e C do Espaço de Estados')
disp ('  ')
else

% geração da matriz de observabilidade:

disp('Matriz de Observabilidade do sistema')
disp(' ')

CO2=obsv(A,C)


tamanho2=size(CO2);

posto2=rank(CO2);

if tamanho2(1,2)==posto2
   disp('Sistema plenamente observável')
end
```

```
if tamanho2(1,2)~=posto2
   disp('Sistema não é plenamente observável')
   disp(' ')
   s2=tamanho2-posto2;
   disp('ordem do subsistema não observável:  ',s2)
end
end
```

- ***Controlabilidade:***

```
% Verificação de Controlabilidade:

k1=exist('A','var');
k2=exist('B','var');

if (k1==0)&(k2==0)
disp('CUIDADO: Defina as matrizes A, B e C do Espaço de Estados')
disp (' ')
else

% geração da matriz de controlabilidade:

disp('Matriz de Controlabilidade do sistema')
disp(' ')

CO1=ctrb(A,B)


tamanho1=size(CO1);

posto1=rank(CO1);

if tamanho1(1,1)==posto1
   disp('Sistema plenamente controlável')
end

if tamanho1(1,1)~=posto1
   disp('Sistema não é plenamente controlável')
   disp(' ')
   s1=tamanho1-posto1;
   disp('ordem do subsistema não controlável:  ',s1)
end

end
```

- ***Menu1:***

```
%Geração de um menu para a escolha do cálculo da controlabilidade
observabilidade
```

```
k=menu ('Escolha a opção desejada', 'Controlabilidade', 'Observabilidade');

menu2
```

- ***Menu 2:***

```
  geração da rotina de observabilidade e controlabilidade:

if k==1

   controlabilidade

end

if k==2
   observabilidade

end
```

- ***Estabilidade:***

```
% Programa de Análise de Estabilidade

t=menu('Entre com os coeficientes do denominador da função de
transferência','Continuar', 'Sair');


if t==1

coef=input('coef=');

pause
s=menu('Escolha o método de Análise de Estabilidade', 'BIBO Estabilidade',
'Estabilidade no Sentido de Lyapunov','Estabilidade Assintótica','T
Estabilidade','Sair');


if s==1  %  Análise da BIBO Estabilidade

   r=(roots(coef))';
   n=size(r);
   j=0;

   for I=1:n(2)
```

```
if real(r(I))<=0

        j=j+1;
    end
```

```
  if j==n(2)

     menu('Sistema BIBO Estável');
  end

  if j-=n(2)

     menu('Sistema não é BIBO Estável');
  end


  end  % Fim da BIBO Estabilidade


  if s==2 %Inicio da Estabilidade no Sentido de Lyapunov

     m=menu('Entre com os coeficientes da seguinte expressão para a
determinação dos AUTOVALORES do sistema','Continuar','Sair');
```

```
     pause

     if m==1
     coef1=input('coef1=');

     auto=(roots(coef1))';
     n2=size(auto);
     j1=0;
     j2=0;

     for I=1:n2(2)
        if auto(I)==0

           j1=j1+1;

        end

        if real(auto(I))< 0

           j2=j2+1;
        end
     end

        if ((j1+j2)==n2(2))& (j1==1)

           menu ('O Sistema é Estável no Sentido de Lyapunov');

        end
```

```
    ((j1+j2)~=n2(2)) | (j1~=1)

       menu( 'O Sistema não é Estável no Sentido de Lyapunov');
    end


end
if m==2

   menu('PROGRAMA FINALIZADO')


end  % Fim da Estabilidade no Sentido de Lyapunov

if s==3 % Inicio da Estabilidade Assintótica

   m1=menu ('Confirmar se os dados para a análise da Estabilidade
Assintótica já estão disponiveis','Continuar', 'Sair');

   if m1==1

      k=exist('n2','var');
      k1=exist('auto','var');

      j3=0;

      if k==0 | k1==0
      coef2=input('coef2=');

      auto=(roots(coef2))';
      n2=size(auto);
   end


       for I=1:n2(2)

         if real(auto(I))< 0

            j3=j3+1;
         end
      end

      if j3==n2(2)

         menu('O sistema é ASSINTOTICAMENTE ESTÁVEL');

      end

      if j3~=n2(2)

         menu('O Sistema NÃO é ASSINTOTICAMENTE ESTÁVEL');
```

```
   end

   if m1==2

      menu('PROGRAMA FINALIZADO')

   end % Final da Estabilidade Assintótica

end

if s==4 %Inicio da análise da T-Estabilidade

   m2=menu('Confirmar se os dados para a análise da T-Estabilidade já estão
disponiveis','Continuar', 'Sair');

   if m2==1

      k2=exist('j1','var');
      k3=exist('j2','var');
      k4=exist('n2','var');

      if k2==0 | k3==0 |k4==0

         disp('Defina os coeficientes do polinômio para a deteminação dos
AUTOVALORES');
         disp('                              );

      coef3=input('coef3=');

      auto=[roots(coef3))';
      n2=size(auto);
      j1=0;
      j2=0;

      for I=1:n2(2)
         if auto(I)==0

            j1=j1+1;

         end

         if real(auto(I))< 0

            j2=j2+1;
         end
      end
   end

   G=menu['Entre com a matriz B do sistema (vetor
coluna)','Confirmar','Sair'};

   B=input('B=')';

   j4=0;
```

```
for J=1:n2(2)
     if auto(J)~=0 & B(J)~=0
        j4=j4+1;
     end
  end
        if ((j1+j2)==n2(2))& (j1==1)&(j4==0)
           menu ('O Sistema é T-Estável');
        end
        if ((j1+j2)~=n2(2)) | (j1~=1)|(j4~=0)
           menu( 'O Sistema NÃO é T-Estável');
        end
  end
  if m2==2
     menu('PROGRAMA FINALIZADO')
  end
end % Final da T-Estabilidade

end

if t==2
   menu('PROGRAMA FINALIZADO');
end
```

- ***Análise1:***

```
%Geração de um menu para a escolha do cálculo da controlabilidade
observabilidade

t=menu('Determinação da Controlabilidade e
Observabilidade','Continuar','Sair');

if t==1
   m=menu('Definir matrizes A,B,C e D do Sistema','Definir','Sair');
   if m==1
```

```
  A=input('A=');
   B=input('B=');
   C=input('C=');
   D=input('D=');

   pause

    k=menu ('Escolha a opção desejada', 'Controlabilidade',
'Observabilidade');
   menu2

end

if m==2

   menu('PROGRAMA FINALIZADO');

end

end


if t==2
   menu('PROGRAMA FINALIZADO');
end
```

- ***Montagem:***

```
%Geração da Matriz de Treinamento e teste (Entrada da Rede)

deg2rad=pi/180;

% definição dos intervalos das variáveis de estado:

 teta = [-20:5:20]*deg2rad;
 vteta = [-5:2.5:5]*deg2rad;
 force = -1:0.25:1;
 desloc=[0:0.2:1];
 vdesloc=[-0.5:0.2:0.5];
teta2=[-20:10:20];


%Montagem da matriz:



A1=combvec(teta,vteta);
A2=combvec(desloc,vdesloc,force);
A3=zeros(3,36);


inicial2=[[A1 A1 A1 A1 A1 A1 A1 A1;A2 A3] [teta2; zeros(4,length(teta2))]];
```

♦ *Pendlinear:*

```
function dy=pendlinear(t,y)

% Estados:

teta=y(1);
vteta=y(2);
desloc  = y(3);
vdesloc  = y(4);
demand = y(5);

% Cálculo das Derivadas:

dteta=vteta;
dvteta= (67.052*(teta))-(131.5789*(demand));
ddesloc = vdesloc;
dvdesloc = 52.63*(demand-(0.1372*teta));
ddemand = 0;

% Retorno das derivadas:
dy = [dteta; dvteta; ddesloc; dvdesloc;ddemand];
```

♦ *Pendnl:*

```
function dy=pendnl(t,y)

% Estado

teta= y(1);
vteta=y(2);
desloc = y(3);
vdesloc= y(4);
forca=y(5);

% Cálculo das Derivadas

dteta = vteta;
dvteta =
(49.*(sin(teta)./((cos(teta)).^2)))+(sin(2.*teta).*((vteta).^2)./(2.*(cos(t
eta).^2)))-
(131.58.*(forca)./(cos(teta)))+(18.052.*(tan(teta)).*(cos(teta)));
ddesloc= vdesloc;
dvdesloc= 52.63.*(forca-0.1372.*(tan(teta)));
dforca = zeros(size(forca));

% Saida:
dy = [dteta; dvteta;ddesloc; dvdesloc; dforca];
```

- ***Adrinl:***

```
% Simulação do modelo não-linear:


timestep = 0.01;
Q = length(teste);
teste_nlinear = [ ];

for i=1:Q
  [ode_time,ode_state] = ode23('pendnl',[0 timestep],teste(:,i));
  teste_nlinear1 = ode_state(length(ode_state),1:4)';
  teste_nlinear=[teste_nlinear teste_nlinear1];

end
```

Final:

```
% Programa para identificação e Neurocontrole

s=menu('Programa para Identificação de Sistemas','Identificação','Sair');


if s==1

   %Definição da matriz de treinamento:

   L=menu('Aperte enter e entre com a matriz de treinamento: Entrada
Saida Desejada','Sim','Não');


   if L==1

   disp('Entre com o nome da matriz de entrada e aperte enter');

   inicial2=input('inicial2=');

   pause

   disp('Enter com o nome da matriz de saida desejada e aperte enter');

   final_nlinear=input('final_nlinear= ');
end

if L==2
```

```
  menu('Programa Finalizado');
end


   %Definição do erro quadrático:

   M=menu('Entre com o valor da soma do erro quadrático desejado','Definir
o valor','Valor automático');

   if M==1
      disp('Entre com o valor da soma do erro quadrático desejado');
      eg=input('eg= ');
   end

   if M==2

      Q=length(inicial2);
      [S2,S]=size(final_nlinear);
      eg = (0.01^2)*Q*2;


   %Escolha do número de épocas por display:

   k=menu ('Escolha o número de épocas por display','valor default
época/display)','Outro valor');


      k==1

      f=1

   end

   if k==2

      disp('Entre com o número de épocas desejado');

      f=input('f =');


   % Escolha do tipo de arquitetura de rede neural:


   m= menu('Escolha da Arquitetura de Rede Neural Desejada','Rede de Base
Radial','Perceptron Multicamadas');

   if m==1

      % treinamento e simulação para a RBF:
      m1=menu('Escolha do Valor do Espalhamento para a Função
Gaussiana','Valor Default (1.0)','Outro Valor');
```

```
if m1==1

        ep=1;

    end

    if m1==2

        disp('Entre com o valor de espalhamento desejado');
        ep=input('ep =');

    end


    menu('inicio do treinamento')

    norma1
    norma2

    [w1,b1,w2,b2,te,tr]=solverb(inicial_norm2, final_norm,[f,Q,eg,ep]);

    menu('Treinamento Terminado');

    pause

    m2=menu('Simulação da Rede','Sim', 'Não');

    if m2==1

        teste=simurb(inicial_norm2,w1,b1,w2,b2);

        menu('Fim da Simulação / Programa Finalizado')

    end

    if m2==2

        menu('Programa Finalizado');

    end


end

if m==2

    % Treinamento e simulação da perceptron multicamadas:


    %Escolha do algoritmo de treinamento:

    m3=menu('Escolha do tipo de treinamento para a Perceptron
Multicamadas','Trainbp','Trainbpx','Trainlm');
```

```
k1=menu('Determinação do número de épocas por treinamento','Valor
default(100)','Outro Valor');

      if k1==1

         d=100;
      end

      if k1==2

         disp('Entre com o valor de epocas desejado para o treinamento');

         d=input('d= ');

      k2=menu('Determinação da Taxa de Aprendizado','Valor default
(0.02)','Outro Valor');

      if k2==1

         mi=0.02;
      end

      if k2==2

         disp('Entre com o valor de taxa de treinamento desejado')

         mi=input('mi= ');
      end


   % Definição do número de neurônios:

   N=menu('Entre com o número de Neurônios da Rede para
Identificação','10','15','20','Outro Valor');


   if N==1
      S1=10
   end
   if N==2
      S1=15
   end
   if N==3
      S1=20
   end
   if N==4
      disp ('Entre com o número desejado de neurônios');
      S1=input['S1=');
   end
```

```
if m3==1

      % TRAINBP:Escolha da função de ativação:Camada Oculta/Camada de
saida:

      m31=menu('Escolha da função de ativação (camada oculta/camada de
saida','logsig/logsig','logsig/purelin','tansig/purelin','tansig/tansig');

      if m31==1

         menu('As matrizes de entrada e saida serão normalizadas. Tecle
enter');

         pause

         normal
         norma2

      [w1,b1,w2,b2]=initff(inicial_norm2,S1,'logsig',S2,'logsig');

[w1,b1,w2,b2,te,tr]=trainbp(w1,b1,'logsig',w2,b2,'logsig',inicial_norm2,fin
al_norm,[f,d,eq,mi]);

      menu ('Treinamento Terminado');

      % Simulação da Perceptron Multicamadas:

   m4=menu('Simulação da Rede:','Sim', 'Não');


   if m4==1

      teste=simuff(inicial_norm2,w1,b1,'logsig',w2,b2,'logsig');

      menu('Simulação Terminada / Programa Finalizado');

   end

   if m4==2

      menu('Programa Finalizado');


   end


   if m31==2

      [w1,b1,w2,b2]=initff(inicial2,S1,'logsig',S2,'purelin');

[w1,b1,w2,b2,te,tr]=trainbp(w1,b1,'logsig',w2,b2,'purelin',inicial2,final_n
linear,[f,d,eq,mi]);

      menu('Treinamento Finalizado')
```

```
% Simulação da Perceptron Multicamadas:
m4=menu('Simulação da Rede:','Sim', 'Não');

if m4==1
   teste=simuff(inicial2,w1,b1,'logsig',w2,b2,'purelin');
   menu('Simulação Terminada / Programa Finalizado');
end
if m4==2
   menu('Programa Finalizado');

end
   if m31==3
      [w1,b1,w2,b2]=initff(inicial2,S1,'tansig',S2,'purelin');
[w1,b1,w2,b2,te,tr]=trainbp(w1,b1,'tansig',w2,b2,'purelin',inicial2,final_n
linear,[f,d,eg,mi]);

      menu('Treinamento Finalizado')
      % Simulação da Perceptron Multicamadas:
   m4=menu('Simulação da Rede:','Sim', 'Não');

   if m4==1
      teste=simuff(inicial2,w1,b1,'tansig',w2,b2,'purelin');
      menu('Simulação Terminada / Programa Finalizado');

      m4==2
      menu('Programa Finalizado');


      end
      if m31==4
        menu('As matrizes de entrada e saida serão normalizadas. Tecle
enter');
```

```
        pause

        normal
        norma2

    [w1,b1,w2,b2]=initff(inicial_norm2,S1,'tansig',S2,'tansig');

[w1,b1,w2,b2,te,tr]=trainbp(w1,b1,'tansig',w2,b2,'tansig',inicial_norm2,fin
al_norm,[f,d,eg,mi]);

    menu('Treinamento Finalizado')

    % Simulação da Perceptron Multicamadas:

  m4=menu('Simulação da Rede:','Sim', 'Não');


  if m4==1

    teste=simuff(inicial_norm22,w1,b1,'tansig',w2,b2,'tansig');

    menu('Simulação Terminada / Programa Finalizado');

  end

  if m4==2

    menu('Programa Finalizado');

  end

  end


end


  if m3==2

    % TRAINBPx:Escolha da função de ativação:Camada Oculta/Camada de
saida:

    m32=menu('Escolha da função de ativação (camada oculta/camada de
saida','logsig/logsig','logsig/purelin','tansig/purelin','tansig/tansig');

    if m32==1

      menu('As matrizes de entrada e saída serão normalizadas. Tecle
enter');

      pause

      normal
      norma2
```

```
      [w1,b1,w2,b2]=initff(inicial_norm2,S1,'logsig',S2,'logsig');

[w1,b1,w2,b2,te,tr]=trainbpx(w1,b1,'logsig',w2,b2,'logsig',inicial_norm2,fi
nal_norm,[f,d,eg,mi,1.05,0.7,0.9,1.04]);

      menu('Treinamento Finalizado')

      % Simulação da Perceptron Multicamadas:

   m4=menu('Simulação da Rede:','Sim', 'Não');

   if m4==1

      teste=simuff(inicial_norm2,w1,b1,'logsig',w2,b2,'logsig');

      menu('Simulação Terminada / Programa Finalizado');

   end

   if m4==2

      menu('Programa Finalizado');

   end

   end

   if m32==2

      [w1,b1,w2,b2]=initff(inicial2,S1,'logsig',S2,'purelin');

[w1,b1,w2,b2,te,tr]=trainbpx(w1,b1,'logsig',w2,b2,'purelin',inicial2,final_
nlinear,[f,d,eg,mi,1.05,0.7,0.9,1.04]);

      menu('Treinamento Finalizado')

      % Simulação da Perceptron Multicamadas:

   m4=menu('Simulação da Rede:','Sim', 'Não');

   if m4==1

      teste=simuff(inicial2,w1,b1,'logsig',w2,b2,'purelin');

      menu('Simulação Terminada / Programa Finalizado');

   end

   if m4==2

      menu('Programa Finalizado');
```

```
end

   end

   if m32==3

      [w1,b1,w2,b2]=initff(inicial2,S1,'tansig',S2,'purelin');

[w1,b1,w2,b2,te,tr]=trainbpx(w1,b1,'tansig',w2,b2,'purelin',inicial2,final_
nlinear,[f,d,eg,mi,1.05,0.7,0.9,1.04]);

      menu('Treinamento Finalizado')

      % Simulação da Perceptron Multicamadas:

   m4=menu('Simulação da Rede:','Sim', 'Não');

   if m4==1

      teste=simuff(inicial2,w1,b1,'tansig',w2,b2,'purelin');

      menu('Simulação Terminada / Programa Finalizado');


   if m4==2

      menu('Programa Finalizado');

   end


   end

      if m32==4

        menu('As matrizes de entrada e saida serão normalizadas. Tecle
enter');

         pause

         normal
         norma2


      [w1,b1,w2,b2]=initff(inicial_norm2,S1,'tansig',S2,'tansig');

[w1,b1,w2,b2,te,tr]=trainbpx(w1,b1,'tansig',w2,b2,'tansig',inicial_norm2,f
nal_norm,[f,d,eg,mi,1.05,0.7,0.9,1.04]);

      menu('Treinamento Finalizado')

      % Simulação da Perceptron Multicamadas:
```

```
 m4=menu('Simulacão da Rede:','Sim', 'Não');

    if m4==1

       teste=simuff(inicial_norm2,w1,b1,'tansig',w2,b2,'tansi');

       menu('Simulação Terminada / Programa Finalizado');

    end

    if m4==2

       menu('Programa Finalizado');

    end

    end

end


    if m3==3

       % TRAINLM:Escolha da função de ativação:Camada Oculta/Camada de
saida:

       m33=menu('Escolha da função de ativação (camada oculta/camada de
saida','logsig/logsig','logsig/purelin','tansig/purelin','tansig/tansig');

       if m33==1

          menu('As matrizes de entrada e saida serão normalizadas. Tecle
enter');

          pause

          normal
          norma2

       [w1,b1,w2,b2]=initff(inicial_norm2,S1,'logsig',S2,'logsig');

[w1,b1,w2,b2,te,tr]=trainlm(w1,b1,'logsig',w2,b2,'logsig',inicial_norm2,fin
al_norm,[f,d,eg,0.0001,0.001,10,0.1,1e10]);


       menu('Treinamento Finalizado')

       % Simulação da Perceptron Multicamadas:

    m4=menu('Simulação da Rede:','Sim', 'Não');

    if m4==1

       teste=simuff(inicial_norm2,w1,b1,'logsig',w2,b2,'logsig');
```

```
      menu('Simulação Terminada / Programa Finalizado');
   end

      m4==2
      menu('Programa Finalizado');



      m33==2
      [w1,b1,w2,b2]=initff(inicial2,S1,'logsig',S2,'purelin');
[w1,b1,w2,b2,te,tr]=trainlm(w1,b1,'logsig',w2,b2,'purelin',inicial2,final_n
linear,[f,d,eq,0.0001,0.001,10,0.1,1e10]);

      menu('Treinamento Finalizado')
      % Simulação da Perceptron Multicamadas:
   m4=menu('Simulação da Rede:','Sim', 'Não');

   if m4==1
      teste=simuff(inicial2,w1,b1,'logsig',w2,b2,'purelin');
      menu('Simulação Terminada / Programa Finalizado');
   end
   if m4==2
      menu('Programa Finalizado');
   end


   end

   if m33==3
      [w1,b1,w2,b2]=initff(inicial2,S1,'tansig',S2,'purelin');
[w1,b1,w2,b2,te,tr]=trainlm(w1,b1,'tansig',w2,b2,'purelin',inicial2,final_n
linear,[f,d,eq,0.0001,0.001,10,0.1,1e10]);
```

```
menu('Treinamento Finalizado')

    % Simulação da Perceptron Multicamadas:

m4=menu('Simulação da Rede:','Sim', 'Não');

if m4==1

    teste=simuff(inicial2,w1,b1,'tansig',w2,b2,'purelin');

    menu('Simulação Terminada / Programa Finalizado');

end

if m4==2

    menu('Programa Finalizado');

end


end

    if m33==4

      menu('As matrizes de entrada e saida serão normalizadas. Tecle
enter');

       pause

       norma1
       norma2


    [w1,b1,w2,b2]=initff(inicial_norm2,S1,'tansig',S2,'tansig');

[w1,b1,w2,b2,te,tr]=trainlm(w1,b1,'tansig',w2,b2,'tansig',inicial_norm2,fin
al_norm,[f,d,eq,0.0001,0.001,10,0.1,1e10]);


    menu('Treinamento Finalizado')

    % Simulação da Perceptron Multicamadas:

m4=menu('Simulação da Rede:','Sim', 'Não');

if m4==1

    teste=simuff(inicial2,w1,b1,'tansig',w2,b2,'tansig');

    menu('Simulação Terminada / Programa Finalizado');

end

if m4==2
```

```
        menu('Programa Finalizado');
    end
    end

end
end
end

if s==2
    menu('Programa Finalizado );
end
```

- ***Adri2nl:***

```
% montagem do conjunto de treinamento para o neurocontrolador:

deg2rad=pi/180;


teta = [-20:4:20]*deg2rad;
vteta = [-2:0.4:2]*deg2rad;
desloc=[-1:0.2:1];
vdesloc=[-0.5:0.2:0.5];
demand=[-4:1:4]*deg2rad;
teta2 = [-20:2.5:20]*deg2rad;



D=combvec(teta,vteta);
E=combvec(desloc,vdesloc,demand);
F=zeros(3,11);


inicial_neuro1 = [D D D D D;E F];
inicial_neuro=[inicial_neuro1 [teta2;zeros(4,length(teta2))]];


timestep = 0.01;
Q = length(inicial_neuro);
final_neuro = [ ];

for i=1:Q
```

```
   [ode_time,ode_state] = ode23('pendlinear',[0
timestep],inicial_neuro(:,i));
   final_neuro1 = ode_state(length(ode_state),1:4)';
   final_neuro=[final_neuro final_neuro1];

end
```

- ***Adri3nl:***

```
S1 = 15;
[cW1,cb1,cW2,cb2] = initff(inicial_neuro,S1,'tansig',1,'purelin');


Q=611;
df = 1;
me = 800;
eg = (0.04^2)*Q*2;
mu = 1e-5;

tp = [df me eg NaN mu];


[cW1,cb1,cW2,cb2,ep,tr] =
traincon[cW1,cb1,cW2,cb2,mW1,mb1,mW2,mb2,inicial_neuro,final_neuro,tp);
```

- ***Teste1:***

```
% Teste do neurocontrolador:


deg2rad=pi/180;

 teta = [-5:1:5]*deg2rad;
 vteta = [-2:0.6:2]*deg2rad;
 desloc=[0:0.15:0.5];
 vdesloc=[-0.5:0.1:0.5];
 demand=[-0.5:0.2:0.5];
 teta2 = [-5:2:10]*deg2rad;

 D=combvec(teta,vteta);
 E=combvec(desloc,vdesloc,demand);
 F=zeros(3,44);

 teste_neuro = [D D D D;E F ];
```

- ***Traj1:***

```
% Testando o Model
% ----------------

deg2rad=pi/180;

teta = 5 * deg2rad;
vteta = 0*deg2rad;
desloc=0;
vdesloc=0;
force = 0;

init_state = [teta; vteta;desloc;vdesloc];


[p_time,p_states] = ode23('pendnl',[0 0.1],[init_state; force]);
p_states = p_states';


% Resposta do Modelo em Malha Aberta
% ----------------------------------


time = 0:0.01:0.1;
state = init_state;
states = zeros(4,length(time));
states(:,1) = state;
for i=4:length(time)
  state = state + simuff([state;force],mW1,mb1,'tansig',mW2,mb2,'purelin');
  states(:,i) = state;
end
```