SINTONIA AUTOMÁTICA DE CONTROLADORES PID USANDO REDES NEURONAIS COM ADAPTAÇÃO EM TEMPO REAL

Dissertação submetida ao grau de Mestre

Ana Beatriz da Piedade de Azevedo

Universidade do Algarve

Dezembro de 1998

DECLARAÇÕES

Declaro que, esta dissertação é o resultado da minha investigação pessoal e independente, o seu conteúdo é original e todas as fontes consultadas estão devidamente mencionadas no texto, nas notas e na bibliografia.

Declaro ainda que, esta dissertação não foi aceite em nenhuma outra instituição para qualquer grau nem está a ser apresentada para obtenção de um outro grau para além daquele a que diz respeito.

A candidata,

Declaro que, tanto quanto me foi possível verificar, esta dissertação é o resultado da investigação pessoal e independente da candidata.

O orientador,

(Prof. Doutor António E. B. Ruano)

Em memória do meu caro professor, mentor e colega

Prof. José António Fernandes Silvestre (1953-98)

AGRADECIMENTOS

Ao meu orientador, Prof. Doutor António E. B. Ruano, pela colaboração, críticas, sugestões, apoio e em especial pela motivação e incentivo prestados e pela cuidada revisão desta dissertação.

À minha família, amigos e colegas, pela paciência, apoio e compreensão demonstrados.

Aos Prof. Doutor Carlos M. da Fonseca e Lic. Pedro Frazão Ferreira pela ajuda prestada dos últimos e mais críticos momentos.

Ao meu companheiro Rui Manuel Sousa Dores, pela paciência e compreensão demonstrados e por me ter suportado nos bons e maus momentos.

Nota: Esta dissertação foi elaborada no âmbito do Programa de Formação Avançada para o Ensino Superior – PRODEP – Acção 5.2

SUMÁRIO

Os controladores Proporcional, Integral e Derivativo (PID) são dos controladores mais usados no controlo automático industrial. A sua utilização em larga escala advém da sua simplicidade de funcionamento e desempenho robusto.

Estes controladores necessitam de ser sintonizados frequentemente, porque o processo a controlar varia no tempo ou por os seus componentes envelhecerem. Dado que, uma instalação industrial pode possuir centenas destes reguladores, e dado que uma sintonia correcta é uma actividade dispendiosa em termos de tempo, a sintonia automática de controladores PID é uma questão importante em termos económicos.

Desde o trabalho de Ziegler e Nichols [1], em 1942, foram propostos vários métodos de sintonia automática de controladores PID. Recentemente, foi introduzida a utilização de redes neuronais artificiais e é sobre o trabalho desenvolvido nesse âmbito, como por exemplo em [2] e [3], que esta dissertação se alicerça.

A aplicação desenvolvida, utiliza como método de sintonia o critério Integral do Tempo multiplicado pelo Erro Absoluto (ITAE), que possui uma resposta com boa selectividade e amortecimento. Este critério é raramente utilizado em aplicações em tempo real, pois como não se conhece uma forma analítica para o seu cálculo é necessário utilizar uma simulação do sistema completo (processo a controlar e controlador PID).

A aplicação é composta por dois módulos, o sistema completo (processo a controlar e controlador PID) e o módulo de sintonia automática. O módulo de sintonia automática consiste num bloco que identifica o processo a controlar e em dois conjuntos de redes neuronais, um utilizado na minimização do ITAE do processo e outro para fornecer os valores dos parâmetros do controlador.

As redes neuronais usadas são do tipo *Basis(B)-spline* [4], que possuem a vantagem de "aprender" preservando o "conhecimento" anteriormente adquirido. Estas redes são construídas e treinadas *off-line*, utilizando o algoritmo ASMOD (*Adaptive Spline Modelling of Observation Data*) [5], usando um conjunto restrito de treino, sendo depois adaptadas em tempo real para valores de treino relativos ao processo específico a controlar.

ABSTRACT

The Proportional, Integral and Derivative (PID) controllers are the controllers more often used in industrial automation. Their use, to a large extent, results from its functional simplicity and its robust performance.

These controllers often need to be retuned, because the plant to be controlled is time varying or because of components ageing. As an industrial plant can have hundreds of these regulators, and as an accurate tuning is a time-consuming operation, PID controller automatic tuning is an important issue in economic terms.

Since the work of Ziegler and Nichols [1], in 1942, several methods have been proposed for PID automatic tuning. Recently, the use of artificial neural networks has been introduced in this subject and the present work is based upon contributions in this area, as for example in [2] and [3].

The application developed, uses as tuning method the Integral of Time multiplied by the Absolute Error (ITAE) criterion, which offers good selectivity and provides well-damped responses. This criterion is seldom used in real-time control because, as it is not known an analytical expression for its calculation, a simulation of the whole system (plant and PID controller) must be performed.

Two modules compose the application: the whole system (plant and PID controller) and the automatic tuning module. The automatic tuning module consists in a block that identifies the plant and in two neural networks sets, one employed in the ITAE minimisation and the other for providing the parameter values to the controller.

The neural networks used are “Basis(B)-spline neural networks” [4], which offer the advantage of “learning” while preserving the previous “knowledge”. These networks are built and trained off-line, using the ASMOD (Adaptive Spline Modelling of Observation Data) algorithm [5], within a restrict training set, and afterwards are adapted in real-time using data specific to the plant.

ÍNDICE

ÍNDICE DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

AMN - Associative Memory Networks

ARMA – Auto-Regressive Moving Average

ASMOD - Adaptive Spline Modelling of Observation Data

BA - Bloco de Adaptação

BIBO – Bounded Input Bounded Output

BIS – Bloco de Identificação e Sintonia

CIB – Critério de Informação Baisiana

CMAC - Cerebellar Model Articulation Controller

ITAE - Integral of Time multiplied by the Absolute Error

KSDMM - Kaverna's Sparse Distributed Memory Model

MSA - Módulo de Sintonia Automática

MSE - Mean Square Error

PID - Proporcional, Integral e Derivativo

RBF – Radial Basis Function

NOMENCLATURA

$u(t)$ - sinal de controlo em malha fechada ou entrada dum sistema em malha aberta

$U(s)$ - Transformada de Laplace de $u(t)$

$e(t)$ - sinal de erro dum sistema em malha fechada

$E(s)$ - Transformada de Laplace de $e(t)$

$y(t)$ - saída dum sistema ou duma rede neuronal

$Y(s)$ - Transformada de Laplace da saída dum sistema $y(t)$

$r(t)$ - entrada de referência de um sistema em malha fechada

k_c, T_i e T_d - ganho proporcional, constante de tempo integral e constante de tempo derivativa, dum controlador PID, respectivamente

$G(s)$ - Função de Transferência dum processo

k_p - ganho DC dum processo

l - atraso dum processo

n_z e n_p - quantidade de zeros e de pólos, dum processo, respectivamente

T_{z_i} e T_{p_j} - constantes de tempo do i-ésimo zero e j-ésimo pólo, respectivamente

α e σ - parâmetro para determinação das medidas de identificação e seu valor escalado, respectivamente

$F(\alpha)$ e $F(\sigma)$ - medidas de identificação dum processo

T_T - constante de tempo total dum sistema, usada como factor de escala

y_{ss} - valor em regime estacionário da saída dum sistema

B - amplitude duma entrada em degrau

$S(\sigma)$ - medida integral dum sistema em malha aberta

u_{ss} - valor em regime estacionário do sinal de controlo dum sistema

$S_u(\sigma)$ - medida integral dum sistema em malha fechada

L e T - atraso e constante de tempo aparentes dum sistema, respectivamente

R - inclinação máxima da curva da resposta ao degrau, dum processo

$\mathbf{x}(t)$ e x - conjunto de entradas e uma entrada duma rede neuronal, respectivamente

$\mathbf{a}(t)$ e a_i - vector de saídas e í-ésima saída das funções-base duma rede *B-spline,* respectivamente

$\mathbf{w}(t)$ e ω_i - vector de pesos e í-ésimo peso duma rede *B-spline,* respectivamente

p - quantidade de funções-base ou pesos duma *lattice-based* AMN

$\lambda_{i,j}$ - j-ésimo nó do i-ésimo eixo de entrada

x_i^{min} e x_i^{max} - valores mínimo e máximo da í-ésima entrada, respectivamente

K e k_i - vector das ordens e ordem das funções-base do i-ésimo eixo de entrada, duma rede *B-spline*, respectivamente

ρ - parâmetro de generalização duma rede *B-spline*

N_k^j e N_K^j - j-ésima função-base univariável de ordem k ou multi-variável de ordens K, respectivamente

I_j - j-ésimo intervalo dum eixo de entrada

$\hat{y}(t)$ - valor desejado da saída duma rede neuronal

$\varepsilon_y(t)$ - erro na saída duma rede neuronal

δ - taxa de aprendizagem da NLMS

Bayk - medida de desempenho dum modelo, pelo CIB

m - quantidade de padrões de treino

J - MSE dum modelo

$\overline{k_{inv}}$ e $\overline{ITAE}$ - valores escalados e invertidos de k_c e ITAE, respectivamente

$\overline{t_i}$ e $\overline{t_d}$ - valores escalados de T_i e T_d

Capítulo 1

INTRODUÇÃO

O controlador PID é largamente utilizado em processos industriais, pelo seu desempenho robusto sobre uma ampla gama de processos e condições operacionais, e ainda, pela sua simplicidade. A sua sintonia pode ser realizada manualmente mas, essa solução é desapropriada, já que estes reguladores necessitam de ser sintonizados, frequentemente, devido a alterações nas condições de funcionamento ou por outros factores como o mero envelhecimento dos componentes. Pelo exposto e dado que uma instalação industrial pode possuir centenas destes controladores, é evidente que a sintonia automática dos mesmos é um assunto de grande interesse económico.

Várias técnicas e métodos de sintonia automática de controladores PID tem sido propostos nos últimos anos, que podem ser classificados, grosso modo, como:

Métodos de resposta em frequência – dos quais o mais conhecido é o de Ziegler-Nichols [1], em que a identificação é feita a partir do chamado ponto crítico, que se obtém aumentando o ganho do sistema até que este comece a oscilar, sendo a sintonia feita usando regras empíricas. Como não é muito aconselhável realizar esta operação, num sistema em funcionamento normal, Åström e Hagglünd [6] propuseram a utilização de realimentação por relé para a determinação do ponto crítico. Vários melhoramentos foram realizados posteriormente, por vários autores. Um exemplo dum controlador comercial que usa realimentação por relé é o controlador com sintonia automática da Satt Control Instruments and Fisher Controls [7].

Métodos de resposta ao degrau – dos quais o mais conhecido é também o de Ziegler- Nichols [1], que utiliza a determinação de um modelo de 1ª ordem, com atraso, usando a resposta ao degrau em malha aberta, processo este dificil de automatizar.

Nishikawa et al. [8] propõem um método melhorado, baseado na obtenção de medidas integrais da resposta ao degrau em malha aberta ou do sinal de controlo, em malha fechada, já com um controlador PID. Vários critérios de sintonia podem ser usados, sendo o mais popular o das regras empíricas de Ziegler- Nichols [1].

Estimação de parâmetros *on-line* – são métodos que assumem um modelo *Auto_Regressive Moving Average* (ARMA) para o processo, e onde os parâmetros do modelo são estimados *on-line*, usando um algoritmo de mínimos quadráticos [9]. Vários critérios podem ser usados com o modelo estimado, originando diferentes controladores PID auto-sintonizáveis, entre os quais um exemplo dum controlador comercial é o Electromax V da Leeds and Northrup [10].

Redes Neuronais, Sistemas *Fuzzy* e Algorítmos Genéticos – entre os quais, Ruano et al. [2] propôs uma técnica, na qual redes neuronais são usada para fazer o mapeamento de medidas de identificação, similares às propostas por Nishikawa et al. [8], nos parâmetros do controlador, obtidos por minimização do ITAE. Uma implementação dum controlador PID com sintonia autimática usando algorítmos genéticos é descrita em Wang e Kwok [11] e uma implementação de sintonia automática multi-objectivos, com algorítmos genéticos é descrita por Schroder et al. [12].

Assentando este trabalho directamente nos desenvolvimentos apresentados em [2], partilha com esse o método de identificação do processo a controlar, o critério de sintonia dos controladores (ITAE) e a utilização de redes neuronais, mas difere no que é o grande objectivo que motiva este trabalho. Enquanto anteriormente as redes neuronais eram treinadas *off-line* e depois permaneciam inalteradas, aqui pretende-se que as redes "aprendam" com o processo a controlar durante o seu funcionamento normal.

O sistema a implementar é constituído por dois módulos, o sistema completo (processo a controlar e controlador PID) e o Módulo de Sintonia Automática (MSA). O MSA é composto por dois blocos, o Bloco de Identificação e Sintonia (BIS) e o Bloco de Adaptação (BA). O BIS é responsável por identificar o sistema e a partir daí, usando um conjunto de redes neuronais, fornecer os parâmetros ao controlador, de acordo com o critério de sintonia empregue. O BA é responsável por determinar os valores óptimos dos parâmetros do controlador, usando uma rede neuronal que modela o critério de sintonia, usar informações recebidas sobre o processo para adaptar (refinar o conhecimento) desta e se necessário usar os valores óptimos para treinar as redes do BIS.

Todos os componentes são construídos e simulados usando o ambiente de simulação MATLAB® [13] e a sua extensão para simulação de sistemas dinâmicos SIMULINK™ [14].

As redes neuronais, do tipo *B-splin*e [4], são construídas/treinadas *off-line* usando o algoritmo ASMOD [5] e posteriormente inseridas no MSA e adaptadas *on-line* e em tempo real.

1.1 - O controlador PID

Uma versão elementar do controlador PID, usando notação de Transformada de Laplace é:

$$U(s) = k_c\left(1 + \frac{1}{T_i s} + T_d s\right)E(s) \tag{1}$$

onde $U(s)$ é a Transformada de Laplace do sinal de controlo, $u(t)$, e $E(s)$ é a Transformada de Laplace do erro, $e(t)$, definido por:

$$e(t) = r(t) - y(t) \tag{2}$$

Na equação (2), $r(t)$ é a entrada de referência e $y(t)$ é a saída. Os termos k_c, T_i e T_d são o ganho proporcional, a constante de tempo integral e a constante de tempo derivativa, respectivamente.

O algoritmo usado na prática contém algumas modificações [2]. É prática comum permitir que a acção derivativa actue apenas sobre a saída do processo, e substituí-la por uma aproximação que reduz o ganho a altas frequências:

$$T_d s \rightarrow \frac{1 + T_d s}{1 + T_f s} \tag{3}$$

onde $T_f = \frac{T_d}{N}$ e geralmente $N = 10$.

Portanto, o diagrama de blocos do sistema usado é:

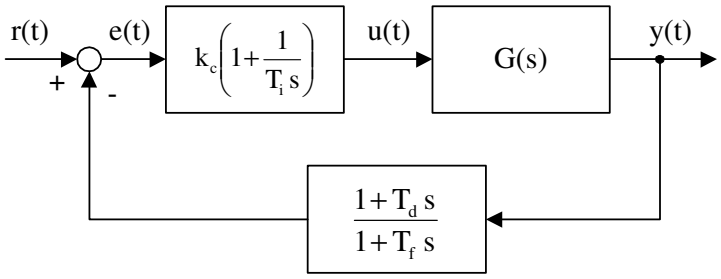


Fig. 1 - Sistema em Malha Fechada

1.2 - As Redes Neuronais

A utilização de redes neuronais em aplicações de controlo, introduzida nos anos sessenta [4], tem visto o seu potencial de aplicação aumentado com os recentes desenvolvimentos ao nível das arquitecturas de redes, de algoritmos de treino e das suas aplicações.

As redes neuronais apresentam grandes vantagens em relação às técnicas algorítmicas tradicionais, tais como, aprendizagem por interacção com o ambiente, em vez de por programação explícita; capacidade de generalizar (interpolar e/ou extrapolar) a partir dum conjunto de treino restrito; estrutura inerentemente paralela, podendo a carga computacional ser facilmente distribuída. Estas características genéricas traduzem-se em relação aos sistemas de controlo, na possibilidade de:

- diminuição da necessidade de intervenção humana;
- aumento da flexibilidade dos sistemas;
- melhoramento do desempenho dos sistemas; e
- redução do tempo de execução e custo do projecto inicial.

Para aplicações de controlo adaptativo em tempo real, as redes neuronais e algoritmos de treino necessitam ainda de garantir uma aprendizagem em tempo real, não redundante, convergente e estável.

A maioria das redes neuronais mais conhecidas não possuem estas últimas características, podendo apenas ser usadas para aplicações *off-line*.

A classe de redes denominada *Associative Memory Networks* (AMN), possuem as propriedades necessárias para serem usadas em controlo adaptativo em tempo real. Nesta classe os tipos de redes mais comuns são as *Cerebellar Model Articulation Controller* (CMAC), *B-spline*, *Radial Basis Function* (RBF) e *Kaverna's Sparse Distributed Memory Model* (KSDMM) [4].

As redes CMAC e *B-spline* pertencem a uma categoria denominada *lattice-based AMNs*, devido ao seu espaço de entrada ser particionado por uma grade (*lattice*) sobre a qual são definidas funções-base.

Destas redes, a escolhida foi a rede *B-spline*, pois para além das propriedades anteriormente referidas, possui uma saída suave, podendo-se incluir conhecimentos

previamente adquiridos e, principalmente, porque a aprendizagem é localizada, ou seja, o conhecimento sobre uma parte do espaço de entrada minimamente afecta o restante.

O principal problema das redes *B-spline*, comum a todas as *lattice-based AMNs*, é conhecido como a "maldição dimensional" e relaciona-se com o facto de a carga computacional para a sua implementação/funcionamento aumentar exponencialmente com a dimensão do espaço de entrada.

1.3 - O algoritmo ASMOD

O algoritmo ASMOD [5] é utilizado para construir e treinar as redes *B-spline*, por um lado por proporcionar a melhor estrutura para uma posterior aprendizagem em tempo real, e por outro, por suplantar a referida "maldição dimensional".

O algoritmo de construção baseia-se em fazer evoluir a rede refinando iterativamente um modelo inicial muito simples. Começa a partir de um modelo inicial vazio ou contendo um número pequeno de sub-redes relevantes e gradualmente novas entradas são incluídas, dependências são identificadas sendo assim formada uma melhor representação de cada entrada. A cada iteração, as formas possíveis da rede evoluir são identificadas e o desempenho de cada é medido, sendo incluído o melhor passo de refinamento no modelo corrente. O modelo obtido é mais complexo mas reproduz fielmente o anterior.

Durante a construção intercalam-se com os passos de refinamento, passos de redução por forma a eliminar possíveis redundâncias.

1. 4 - O ambiente de simulação MATLAB® e SIMULINK™

O MATLAB é um poderoso ambiente de simulação, fácil de utilizar e com grandes capacidades a nível de cálculo, introdução e manipulação de dados e representação gráfica. A sua aplicação tem vindo a alargar-se a variados campos desde a Álgebra Linear, Cálculo Vectorial, Análise Numérica até ao Controlo Automático, Processamento de Sinal, etc. e a diversos níveis desde o ensino até à investigação aplicada.

Para além de imensas funções básicas, de cálculo, aquisição e apresentação de dados e outras, ainda existem extensões que permitem a resolução de problemas específicos, denominadas *toolboxes*.

Estas *toolboxes* são conjuntos de funções escritas em MATLAB que permitem ampliar a aplicação deste, e existem para vários campos específicos, das quais serão utilizadas as seguintes:

- Control System Toolbox [15]; e
- Optimization Toolbox [16].

O SIMULINK é uma extensão do MATLAB destinada à simulação de sistemas dinâmicos. Para além de dispor de todas as funcionalidades deste, possui ainda alguns módulos mais adequados a simulação propriamente dita. A sua utilização é muito simples e agradável, sendo os modelos construídos em diagrama de blocos juntando elementos e blocos pré-definidos ou construídos especificamente para o modelo em causa. A análise e simulação do modelo pode ser iniciada e dirigida quer do SIMULINK, quer do MATLAB; quaisquer dados podem transitar dum sistema para o outro facilmente; permite ainda visualizar a simulação enquanto esta decorre.

1.5 – Estrutura da Dissertação

Os assuntos que são comuns ao trabalho apresentado em [2], isto é, as características do processo a controlar e do controlador PID, bem com as medidas de identificação e critério de sintonia, são expostos no capítulo 2.

No capítulo 3, descrevem-se as propriedades, estrutura e vantagens das redes neuronais *B-spline,* bem como a regra de aprendizagem e algoritmo de construção (ASMOD), utilizados para a sua implementação e treino.

Os passos necessários para iniciar o desenvolvimento das redes neuronais, a sua implementação, treino e teste de funcionamento são expostos no capítulo 4. Neste capítulo também se apresentam e comentam alguns resultados intermédios mal sucedidos mas que são relevantes por a bibliografia consultada ser omissa ou pouco clara sobre o assunto.

Os componentes do sistema final, sua interligação e funcionamento são descritos no capítulo 5, onde também se apresentam resultados obtidos da simulação do seu funcionamento normal e em tempo real.

Nos Apêndices, apresentam-se os modelos desenvolvidos e programas elaborados, bem como resultados intermédios relevantes.

Capítulo 2

IDENTIFICAÇÃO DO PROCESSO A CONTROLAR E SINTONIA DO CONTROLADOR

As características do processo a controlar e do controlador PID, o método de identificação e critério de sintonia empregues são comuns ao trabalho ao qual este dá seguimento (para mais informações consultar [2]).

O processo a controlar e o controlador PID são inicialmente simulados por forma a obter as entradas e valores desejados para o treino/adaptação das redes neuronais.

As entradas das redes neuronais são medidas de identificação do sistema em malha aberta e/ou malha fechada, como se mostra na figura 2.

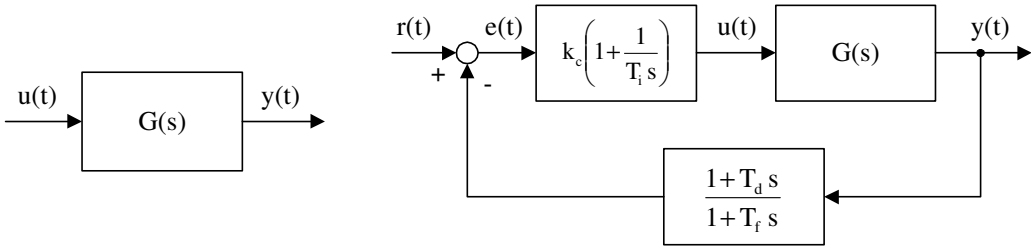


Fig. 2 - Sistema em (a) malha aberta e em (b) malha fechada

Os valores desejados como saída das redes neuronais são os valores óptimos de k_c, T_i e T_d; denominados óptimos porque minimizam o critério de sintonia utilizado.

Como se pode observar na figura 2, o sistema é encarado numa perspectiva em que não se considera a existência de ruído ou de perturbações, já que a tónica deste trabalho centra em verificar as possibilidades de conseguir realizar o processo de sintonia com

adaptação em tempo real, para um caso simples, e só então, numa fase posterior, estudar a sua expansão a processos mais complicados e reais.

2.1 – Medidas de identificação

As entradas das redes neuronais são as medidas de identificação propostas em [2]. Estas medidas de identificação apresentam uma vantagem muito importante, que consiste em ser possível obtê-las quer em malha aberta quer em malha fechada.

Considerando processos a controlar com função de transferência:

$$G(s) = \frac{Y(s)}{U(s)} = k_p e^{-ls} \frac{\prod_{i=1}^{n_z} (1 + T_{z_i} s)}{\prod_{j=1}^{n_p} (1 + T_{p_j} s)} \tag{4}$$

onde, k_p é o ganho do processo, l é o atraso ($l \geq 0$) e T_{z_i} e T_{p_j} são as constantes de tempo dos zeros e dos pólos, respectivamente. Assume-se o processo estável no sentido *Bounded Input Bounded Output* (BIBO).

As medidas

$$F(\alpha) = \left. \frac{G(s)}{k_p} \right|_{s=\alpha} = e^{-l\alpha} \frac{\prod_{i=1}^{n_z} (1 + T_{z_i} \alpha)}{\prod_{j=1}^{n_p} (1 + T_{p_j} \alpha)} \tag{5}$$

para uma certa gama de valores reais de α, podem ser utilizadas para caracterizar o processo. Uma soma das constantes de tempo e do tempo de atraso do processo

$$T_T = l + \sum_{i=1}^{n_p} T_{p_i} - \sum_{j=1}^{n_z} T_{z_j} \tag{6}$$

é usada como factor de escala para normalizar α:

$$\alpha(\sigma) = \frac{\sigma}{T_T} \tag{7}$$

onde σ pertence ao intervalo $[0.1,10]$.

Portanto, as medidas de identificação serão dadas por

$$F(\sigma)=\frac{G(s)}{k_p}\bigg|_{s=\frac{\sigma}{T_T}}=e^{-\frac{l\sigma}{T_T}}\frac{\prod_{i=1}^{n_z}\left(1+T_{z_i}\frac{\sigma}{T_T}\right)}{\prod_{j=1}^{n_p}\left(1+T_{p_j}\frac{\sigma}{T_T}\right)} \tag{8}$$

As entradas das redes neuronais serão as medidas $F(\sigma)$, para alguma(s) concretização(ões) de σ.

Para calcular as medidas de identificação em malha aberta, um degrau de amplitude B é aplicado ao processo. O ganho DC pode ser obtido a partir do valor em regime estacionário da saída:

$$k_p=\frac{y_{ss}}{B} \tag{9}$$

Introduzindo as medidas integrais $S(\sigma)$ definidas por:

$$S(\sigma)=\int_0^\infty e^{-\frac{\tau\sigma}{T_T}}[y_{ss}-y(\tau)]d\tau \tag{10}$$

o factor de escala pode ser calculado como:

$$T_T=\frac{S(0)}{y_{ss}} \tag{11}$$

As medidas de identificação obtêém-se de:

$$F(\sigma)=1-\frac{\sigma S(\sigma)}{T_T y_{ss}} \tag{12}$$

Para calcular as medidas de identificação em malha fechada, utiliza-se o sistema da figura 2b).

Aplicando um degrau de amplitude B ao sistema, o ganho DC pode ser obtido a partir do valor em regime estacionário do sinal de controlo:

$$k_p = \frac{B}{u_{ss}} \tag{13}$$

Calculando as medidas integrais do sinal de controlo de:

$$S_u(\sigma) = \int_0^{\infty} e^{-\frac{\tau\sigma}{T_T}} [u_{ss} - u(\tau)] d\tau \tag{14}$$

o factor de escala obtém-se de:

$$T_T = \frac{T_i}{k_p k_c} + T_d\left(1 - \frac{1}{N}\right) - \frac{S_u(0)}{u_{ss}} \tag{15}$$

e as medidas de identificação de:

$$F(\sigma) = \frac{1 + \frac{\sigma T_d}{N T_T}}{1 + \frac{\sigma T_d}{T_T}} \left(\frac{B}{B - \frac{\sigma k_p S_u(\sigma)}{T_T}} - \frac{\sigma T_i}{k_p k_c (T_T + \sigma T_i)} \right) \tag{16}$$

2.2 – O critério de sintonia ITAE

Existem inúmeros critérios de sintonia de controladores PID, baseados na resposta ao degrau, na resposta em frequência, em medidas integrais de desempenho, etc.

Um dos mais conhecidos e simples é o método da resposta do sistema, ao degrau, em malha aberta, de Ziegler-Nichols [1]. Partindo duma resposta, ao degrau, em malha aberta, como se mostra na seguinte figura:

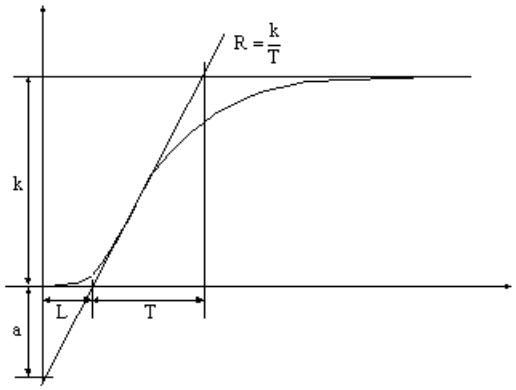


Fig. 3 - Resposta, ao degrau, dum sistema em malha aberta

onde, L e T são o atraso e a constante de tempo aparentes do sistema, respectivamente, e R é a inclinação máxima da curva.

Ziegler-Nichols propuseram, as seguintes fórmulas, para determinação dos parâmetros do controlador:

$$\begin{cases} k_c = 1.2/RL \\ T_i = 2L \\ T_d = L/2 \end{cases} \tag{17}$$

Este método apresenta algumas desvantagens das quais se destaca o facto de produzir uma resposta em malha fechada pouco amortecida.

Um grupo de métodos conhecidos como Critério de Desempenho Integral apresentam uma resposta com um bom amortecimento. Destes, vamos usar o critério do ITAE, por apresentar um melhor amortecimento que os restantes e por também ser bastante selectivo nos parâmetros do controlador [2].

Os critérios de Desempenho Integral baseiam-se em obter os valores de k_c, T_i e T_d que minimizam um integral, que no caso do ITAE (*Integral of Time multiplied by the Absolute Error*) é:

$$\text{ITAE} = \int_0^\infty t\,|e(t)|\,dt \tag{18}$$

Existe uma grande desvantagem neste método pois, não sendo conhecida forma analítica para o seu cálculo, torna-se necessário calculá-lo por integração rectangular ou trapezoidal, usando amostras de $e(t)$, obtidas a partir duma simulação do sistema em malha fechada.

Capítulo 3

AS REDES NEURONAIS

As redes neuronais que serão usadas são *B-spline neural networks* [4]. Estas redes são do tipo AMN e diferem das restantes do mesmo tipo, porque guardam a informação localmente, o que é fundamental numa utilização de aprendizagem em tempo real.

Estas redes serão treinadas usando a regra *Normalised Least Mean Squares* (NLMS) [4], que para além de ter um bom desempenho em treino *off-line* é a mais apropriada para adaptação em tempo real.

A construção e treino *off-line* das redes será realizado usando o algoritmo ASMOD, que garante a estrutura mais racional para cada conjunto de treino, minimizando desta forma o problema conhecido como "maldição dimensional" e conseguindo uma boa rede inicial para a posterior adaptação em tempo real.

3.1 - As redes *B-spline*

A estrutura básica duma *lattice-based AMN* comporta três níveis, como se mostra na figura 4:

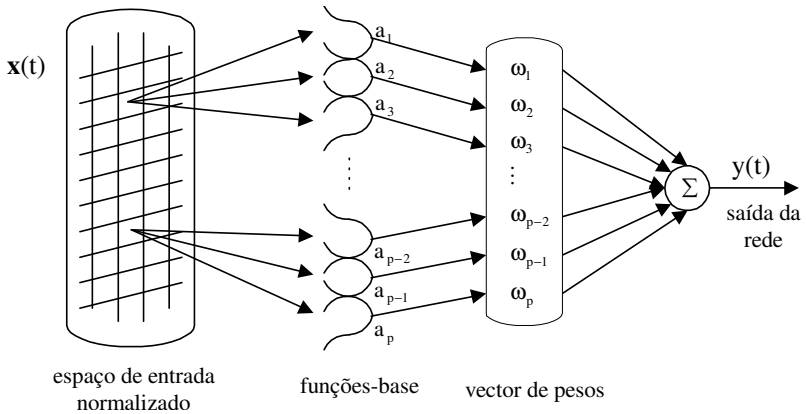


Fig. 4 - Estrutura básica duma *lattice-based AMN*

No primeiro nível, uma grade é definida para normalizar o espaço de entrada original; o segundo nível é constituído por funções-base definidas sobre essa grade e no terceiro nível, é definido um vector de pesos, estando cada peso associado à saída duma função-base. A saída da rede será uma combinação linear das saídas das funções-base cujos coeficientes são os pesos.

Para gerar uma grade sobre o espaço de entrada n-dimensional, é necessário definir um conjunto de n vectores, um para cada eixo da entrada. Os valores dos nós, em cada vector, vão definir n-1 hiperplanos paralelos aos restantes eixos e é o conjunto de todos os hiperplanos que vai formar a referida grade.

Sobre o espaço de entrada, definem-se $\sum_{i=1}^{n} r_i$ nós interiores, $\lambda_{i,j}$, que devem respeitar a seguinte relação:

$$x_i^{min} < \lambda_{i,1} \leq \lambda_{i,2} \leq \cdots \leq \lambda_{i,ri} < x_i^{max} \tag{19}$$

onde, x_i^{min} e x_i^{max} são os valores mínimo e máximo da i-ésima entrada, respectivamente. Esses valores têm, portanto, de ser conhecidos ou definidos antes de se iniciar o treino.

Por forma a permitir uma aproximação correcta nas regiões perto dos limites, um conjunto de nós exteriores tem também de ser definido, que verifique as seguintes relações:

$$\lambda_{i,-(k_i-1)} \leq \ldots \leq \lambda_{i,0} = x_i^{min} \tag{20.1}$$

$$x_i^{max} = \lambda_{i,r_i+1} \leq \ldots \leq \lambda_{i,r_i+k_i} \tag{20.2}$$

onde k_i é a ordem das funções-base a definir sobre esse eixo de entrada.

Sobre este primeiro nível, são definidas funções-base, cujo tamanho, forma e sobreposição determinam como a rede generaliza num espaço n-dimensional e também a sua complexidade.

É o tipo de funções-base utilizadas que distingue o tipo de *lattice-based AMN*. Nas redes *B-spline*, as funções-base são funções polinomiais, como se mostra na figura 5.

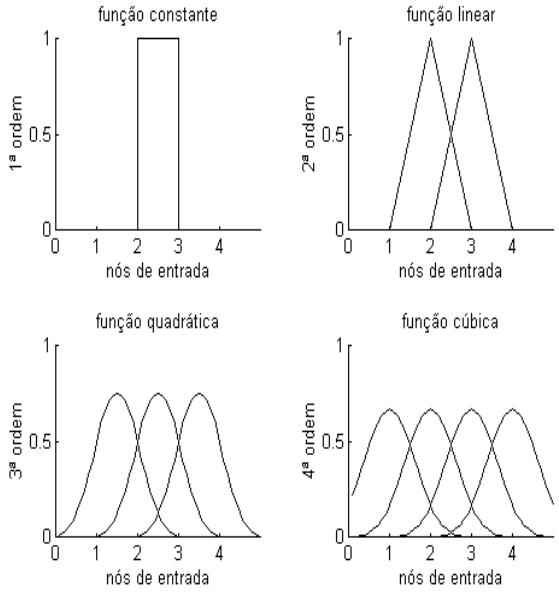


Fig. 5 – Funções-base univariáveis duma rede *B-spline*, de ordens 1 a 4, activadas por uma entrada pertencendo ao intervalo $[\lambda_2, \lambda_3[$

Observa-se facilmente que, a ordem da função-base determina quer o seu tipo quer a largura e número de funções-base univariáveis activadas para uma entrada num certo intervalo. A designação função-base univariável deve-se a que esta é construída apenas sobre um eixo da entrada, existindo ainda funções-base multivariáveis que resultam do produto tensorial de n funções-base univariável, cada uma e só uma definida sobre cada um dos eixos de entrada.

As saídas das funções-base, a_i, geralmente pertencem ao intervalo $[0,1]$, e quando a saída é nula, a função-base e o peso correspondente não contribuem para a saída da rede. Para cada entrada, o número de funções-base não nulas é uma constante que se denomina parâmetro de generalização, ρ.

Quando se inicia o desenho duma rede *B-spline*, é necessário definir a forma (ordem) de cada uma das funções-base. Se as funções-base são todas de largura k, $\rho = k^n$ funções-base contribuem para a saída da rede.

A saída duma rede *B-spline* resulta duma combinação linear dum conjunto de funções-base, onde os coeficientes são os pesos, ω_i. Assim, no instante t, temos:

$$y(t) = \sum_{i=1}^{p} a_i(t) * \omega_i(t-1) \tag{21.1}$$

ou de preferência:

$$y(t) = \sum_{i=1}^{\rho} a_{ad(i)}(t) * \omega_{ad(i)}(t-1) \tag{21.2}$$

Na equação (21.2), um algoritmo de cálculo de endereço garante que seja necessário calcular apenas a saída de ρ funções-base, em vez de p (total de funções-base), o que resulta numa grande redução de custo computacional.

Denotando a j-ésima função-base univariável de ordem k por $N_k^j(\ )$ (omite-se o índice i referente à entrada para simplificar a notação), as funções-base são definidas como:

$$N_k^j(x) = \left(\frac{x - \lambda_{j-k}}{\lambda_{j-1} - \lambda_{j-k}}\right) N_{k-1}^{j-1}(x) + \left(\frac{\lambda_j - x}{\lambda_j - \lambda_{j-k+1}}\right) N_{k-1}^{j}(x) \tag{22}$$

$$N_1^j(x) = \begin{cases} 1 \Leftarrow x \in I_j \\ 0 \Leftarrow x \notin I_j \end{cases} \tag{23}$$

onde λ_j é o j-ésimo nó e $I_j = [\lambda_{j-1}, \lambda_j[$ é o j-ésimo intervalo.

As funções-base obtidas, usando a relação recursiva anterior, possuem algumas propriedades desejáveis, como seja:

- As funções-base são definidas sobre um domínio limitado e a sua saída é positiva em todo o seu domínio;

$$\begin{cases} N_k^j(x) = 0 & \Leftarrow \quad x \notin [\lambda_{j-k}, \lambda_j] \\ N_k^j(x) > 0 & \Leftarrow \quad x \in]\lambda_{j-k}, \lambda_j[\end{cases} \tag{24}$$

- As funções-base formam uma partição da unidade, ou seja, para qualquer entrada, a soma das saídas de todas as funções-base é sempre igual a um.
- A saída da rede é limitada superior e inferiormente pelos valores dos pesos das funções-base não-nulas, ou seja,

$$\min\{\omega_j, \omega_{j+1}, \ldots, \omega_{j+k-1}\} \le y(x) \le \text{máx}\{\omega_j, \omega_{j+1}, \ldots, \omega_{j+k-1}\} \tag{25}$$

Funções-base multivariáveis são formadas pelo produto tensorial de n funções-base univariáveis, cada uma definida sobre cada um dos eixos de entrada. A j-ésima função-base multivariável $N_K^j(\ )$ é dada por:

$$N_K^j(x) = \prod_{i=1}^{n} N_{k_i,i}^j(x_i) \tag{26}$$

onde K é um vector n-dimensional composto pelas ordens das funções-base univariáveis, k_i. Estas funções partilham as propriedades das funções univariáveis, acima descritas.

O número de funções-base multivariáveis, de ordem k_i definidas num eixo com r_i nós interiores é $k_i + r_i$, portanto o total das necessidades de memória é:

$$p = \prod_{i=1}^{n} (k_i + r_i) \tag{27}$$

o que depende exponencialmente da dimensão do espaço de entrada. Este problema torna as redes *B-spline* desapropriadas para resolver problemas nos quais o espaço de entrada tem cinco ou mais variáveis de entrada, e é geralmente denominado como a "maldição dimensional". O algoritmo ASMOD [5] é um método que principalmente tenta ultrapassar este problema.

3.2 - Regras de Aprendizagem

Os pesos duma rede *B-spline* podem ser treinados iterativamente usando uma variedade de regras do tipo *Least Mean Square* (LMS), das quais vamos usar a regra NLMS, também conhecida como Regra de Correcção de Erro [4].

A regra NLMS actualiza o vector de pesos, por forma a reduzir o erro na saída, após cada padrão de treino ter sido apresentado à rede; razão pela qual é particularmente apropriada para adaptação em tempo real.

A saída duma rede *B-spline*, no instante t, é dada por:

$$y(t) = \sum_{i=1}^{p} a_i(t) * \omega_i(t-1) = \mathbf{a}^T(t)\mathbf{w}(t-1) \tag{28}$$

e sendo $\tilde{y}(t)$ o valor desejado da saída da rede temos que o erro na saída da rede, no instante t, é dado por:

$$\varepsilon_y(t) = \tilde{y}(t) - y(t) = \tilde{y}(t) - \mathbf{a}^T(t)\mathbf{w}(t-1) \tag{29}$$

O acréscimo do peso, $\Delta\mathbf{w}(t-1)$, é calculado segundo uma direcção paralela ao vector de entrada e o passo é calculado de maneira a que a informação correcta da relação entrada/saída, no instante t, seja guardada pela rede. Esta condição é satisfeita, se o vector de pesos actualizado verificar a condição:

$$\tilde{y}(t) = \mathbf{a}^T(t)\mathbf{w}(t) \tag{30}$$

e a regra de aprendizagem, no instante t, pode ser escrita como:

$$\Delta\mathbf{w}(t-1) = c(t)\mathbf{a}(t) \tag{31}$$

para o passo de tamanho $c(t)$.

Das três equações anteriores obtém-se que:

$$c(t) = \frac{\varepsilon_y(t)}{\mathbf{a}^T(t)\mathbf{a}(t)} \tag{32}$$

resultando numa regra de aprendizagem da forma:

$$\Delta \mathbf{w}(t-1) = \frac{\delta \varepsilon_y(t)\mathbf{a}(t)}{\mathbf{a}^T(t)\mathbf{a}(t)} \tag{33}$$

onde δ é a taxa de aprendizagem.

Se alguma das funções-base não tiver sido inicializada, a equação (33) torna-se em:

$$\omega_i(t) = \begin{cases} \hat{y}(t) & \Leftarrow \omega_i \text{ não inic.} \wedge a_i(t) > 0 \\ \omega_i(t-1) + \dfrac{\delta \varepsilon_y(t)}{\mathbf{a}^T(t)\mathbf{a}(t)} a_i(t) & \Leftarrow \text{outros casos} \end{cases} \tag{34}$$

Obtém-se uma aprendizagem estável para taxas de aprendizagem no intervalo $]0,2[$ e se $\delta = 1$, o exemplo de treino é guardado ao fim de apenas uma iteração [4].

3.3 – O Algoritmo ASMOD

O algoritmo ASMOD [5] baseia-se em fazer evoluir a rede refinando, iterativamente, um modelo inicial muito simples. Começa a partir de um modelo inicial vazio ou contendo um número pequeno de sub-redes relevantes e gradualmente novas entradas são incluídas, dependências são identificadas sendo assim formada uma melhor representação de cada entrada. A cada iteração, as formas possíveis da rede evoluir são identificadas e o desempenho de cada rede é medido, sendo incluído o melhor passo de refinamento no modelo corrente. O modelo obtido é mais complexo mas reproduz fielmente o anterior mas, para tal ser possível, o algoritmo não pode alterar a ordem das funções-base, donde que esta tem que ser inicialmente definida para cada entrada.

As formas elementares em que a rede pode evoluir, em cada iteração, são:

- introdução duma nova variável, representada como um sub-modelo univariável de ordem k, geralmente com k funções-base definidas sobre o eixo de entrada, e que a rede tem a capacidade de modelar aditivamente;

- dependências inter-variáveis, sub-modelos univariável e multivariável são combinados para formar novos modelos multivariável;
- adição de novas funções-base, o que corresponde a introduzir um nó no eixo de entrada, nó este que só pode ser introduzido a meia distância entre dois nós adjacentes.

Durante a construção, intercalam-se com os passos de refinamento, passos de redução, por forma a eliminar possíveis redundâncias. Estes passos consistem em separar modelos multivariável em sub-modelos univariável, modelando as entradas aditivamente, ou retirar nós, o que possibilita que existam nós a ¼ ou ¾ de distância entre nós adjacentes.

Por forma a determinar o novo modelo a adoptar ou a paragem da construção, são usadas medidas de desempenho do modelo. Estas medidas devem fazer o balanço entre a complexidade da rede, a quantidade de dados de treino e a redução do erro quadrático médio (MSE).

Existem algumas formas conhecidas, para cálculo destas medidas de desempenho, das quais vamos usar o Critério de Informação Baisiana (CIB) [4], que é dada por:

$$\mathrm{Bayk} = m\ln(J) + p\ln(m) \tag{35}$$

onde, Bayk é a medida de desempenho, m é o número de padrões de treino, J é o MSE do modelo, e p é o tamanho (número de pesos) do modelo.

Geralmente, toma-se por novo modelo aquele que apresentar o menor valor de Bayk, ou termina-se a construção se em passos sucessivos de construção/redução, não se conseguir reduzir o valor de Bayk.

Capítulo 4

IMPLEMENTAÇÃO E RESULTADOS INTERMÉDIOS

O processo de implementação foi desenvolvido em várias etapas.

Desenvolveram-se vários modelos em SIMULINK e programas em MATLAB para:

- cálculo do ITAE em função dos parâmetros do controlador e do processo;
- minimização do ITAE em função dos parâmetros do controlador;
- cálculo das medidas de identificação em malha aberta e fechada;
- implementação e treino das redes *B-spline*;
- implementação do algoritmo ASMOD;
- ligação dos vários componentes para funcionamento em tempo real.

Para alguns casos teste, procedeu-se a:

- obtenção das medidas de identificação e comparação com os valores analíticos correspondentes;
- obtenção dos valores dos parâmetros do controlador que minimizam o ITAE;
- construção e treino *off-line* das redes neuronais responsáveis pelos parâmetros do controlador;
- simulação de treino *on-line* das redes anteriores;
- construção e treino *off-line* da rede neuronal de modelização do ITAE;
- simulação de treino *on-line* da rede anterior e teste da sua utilização para minimização do ITAE;
- simulação do sistema final em funcionamento em tempo real.

4.1- Processo a controlar e controlador PID

Neste ponto apresenta-se e descreve-se o funcionamento de programas e modelos criados para simulação do processo a controlar e controlador PID, cálculo do ITAE, minimização do ITAE e determinação das medidas de identificação. Também se apresentam e comentam alguns resultados obtidos, e são tecidas considerações relativamente à escolha das medidas de identificação (malha aberta ou fechada) a utilizar para o treino das redes.

O conjunto processo a controlar e controlador é constituído como se mostra, na figura 6:

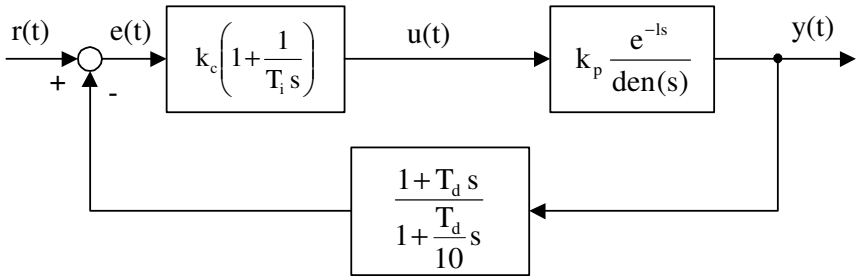


Fig. 6 - Esquema do processo a controlar e controlador PID

onde, o atraso l toma valores entre 0.1 e 3, a intervalos de 0.1, $\text{den}(s) = s+1$ e $k_p = 1$.

Por uma questão de simplicidade e transparência, vários modelos foram implementados usando o SIMULINK, consoante a aplicação a que se destinam (Apêndice A).

O algoritmo de simulação utilizado é o denominado Runge-Kutta de 5ª ordem, aconselhado para sistemas com elementos não lineares [14] e que, foi preferido em relação ao Runge-Kutta de 3ª ordem, por ser mais rápido que este último e por gerar menos amostras, com uma melhor exactidão.

Para obter os valores do controlador que minimizam o ITAE, foram utilizados o modelo em SIMULINK *sistema* (Apêndice A.1) e as funções do MATLAB *itaef*, *initaezn* (Apêndices B.1 e B.2) e *fminuit* .

O modelo *sistema* implementa o processo a controlar e o controlador PID e retorna os valores do tempo de simulação e do erro e(t), contendo ainda duas condições de paragem da simulação, uma para quando o sistema se torna instável e outra para quando o erro e(t) toma valores muito próximos da tolerância permitida. O critério de paragem da simulação quando o erro e(t) se aproxima da tolerância permitida é motivado por, na tentativa de só se parar a simulação quando o erro se anulasse, realizaram-se várias simulações, observando-se que, o erro nunca se anula e oscila indefinidamente em torno do valor final com valores próximos da tolerância permitida. Usando tolerâncias máximas $10^{-6}, 10^{-9}, 10^{-12}, 10^{-15}$ e 10^{-18} e vários limites de aceitação de resultados, observou-se que, rejeitando os valores do erro e(t) inferiores a cem vezes a tolerância máxima permitida, se obtinha exactamente o mesmo resultado, para tolerâncias de $10^{-12}, 10^{-15}$ e 10^{-18}. Em face ao descrito acima e dado que a quantidade de amostras fornecidas pela simulação é inversamente proporcional à tolerância, optou-se por usar o maior valor aceitável entre os testados. Sendo assim, utilizou-se uma tolerância máxima de 10^{-12} e a simulação foi terminada quando o módulo do erro do sistema se tornasse inferior a 10^{-10}.

A função *itaef* calcula o valor do ITAE, para cada conjunto de valores $\{1, k_c, T_i, T_d\}$, retornando também uma mensagem de aviso se o ITAE for infinito, útil durante a observação da minimização. O cálculo do ITAE foi realizado por integração numérica trapezoidal, usando todas as amostras do tempo, t, e erro, e(t), resultantes da simulação.

A função *initaezn* determina os valores óptimos de k_c, T_i e T_d segundo o método da resposta do sistema ao degrau, em malha aberta, de Ziegler-Nichols, descrito no capítulo 2.2. Estes valores serão usados para iniciar a minimização.

A função *fminuit* realiza a minimização do ITAE, em função dos parâmetros k_c, T_i e T_d, a partir dum ponto inicial fornecido. Esta função foi adaptada da função *fminu* da Optimization Toolbox [16]. As adaptações realizadas foram inseridas porque a função original tende a provocar grandes variações em relação ao ponto inicial fornecido, nos primeiros passos de minimização, provocando muitas vezes resultados infinitos e não conseguindo depois evoluir.

O processo de minimização iniciou-se, tendo como ponto de partida os valores óptimos segundo o método de Ziegler-Nichols, para um caso e depois usando os novos valores óptimos, como ponto de partida para os casos adjacentes. Este processo revelou-se moroso e cansativo porque, o algoritmo de minimização por vezes ficava bloqueado, em mínimos locais do ITAE ou em resultados infinitos consecutivos, tendo de ser permanente observado e assistido.

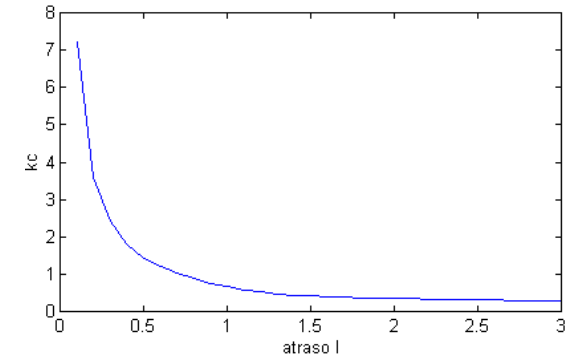
Obtidos os primeiros resultados da minimização, observou-se ainda que havia algumas discrepâncias entre valores de casos adjacentes, razão pela qual se procedeu a várias outras minimizações, tomando novos pontos de partida, até não se conseguir obter melhores resultados.

Devido a todos os problemas descritos, o processo de minimização do ITAE, para os trinta casos estudados ($1 = 0.1,0.2,...3$), resultou num gasto de várias centenas de horas de trabalho.

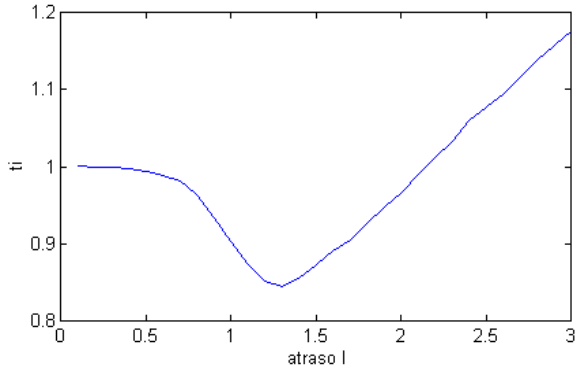
Os valores finalmente obtidos, apresentam-se na figura 7, onde se pode observar que a relação de proporcionalidade entre T_i e T_d e a relação directa entre T_i e o atraso obtidas pela aplicação das regras de Ziegler-Nichols (17) não foram verificadas para este caso, dado que as regras empíricas de Ziegler-Nichols tiveram como base a minimização do integral do erro absoluto (IAE) [17] que é dado por:

$$IAE = \int_0^{\infty} |e(t)| \, dt \qquad (36)$$

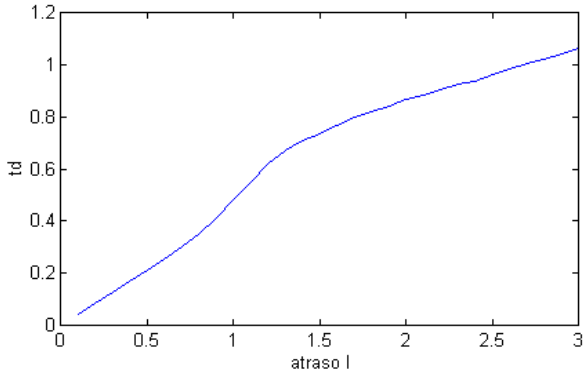
enquanto que o critério aqui utilizado é a minimização do ITAE.



a)



b)



c)

Fig.7 - Valores que minimizam o ITAE , (a) k_c óptimo, em função de l, (b) T_i óptimo, em função de l , (c) T_d óptimo, em função de l.

Para obter as medidas de identificação em malha aberta, foram utilizados, o modelo em SIMULINK *sinid* (Apêndice A.2) e a função em MATLAB *fsigmaop* (Apêndice B.3).

O cálculo dos integrais e o critério de paragem da simulação são semelhantes aos descritos para o cálculo do ITAE.

As medidas foram obtidas para $\sigma = 1$ porque, para este valor consegue-se conciliar um bom espaçamento entre medidas adjacentes com um valor não muito pequeno das mesmas, ou seja, com valores de σ menores, as medidas de identificação têm um valor maior mas o espaçamento relativo é menor e com valores de σ maiores, o espaçamento relativo é maior mas o valor das medidas de identificação torna-se muito pequeno.

Obtidas as medidas $F(\sigma)$ em malha aberta, observou-se que, estas diferem dos valores que se obtém analiticamente, e que o erro é tanto maior quanto maior for o atraso l.

Os valores de $F(\sigma)$ e de T_T medidos são superiores aos valores analíticos em, aproximadamente 0.0007 % a 0.16 % para $F(\sigma)$ e aproximadamente 0.0002 % a 0.0008 % para T_T. Os valores de k_p são inferiores, aos valores reais, em cerca de 4×10^{-9} % a 5×10^{-9} %. Este problema não se torna preocupante porque não são os valores analíticos de $F(\sigma)$ que serão utilizados para treino das redes, e em relação a T_T e k_p os erros são muito pequenos.

Para obter as medidas de identificação em malha fechada, foram utilizados, o modelo em SIMULINK *csist* (Apêndice A.3) e a função em MATLAB *fsigmacp* (Apêndice B.4).

O cálculo dos integrais e o critério de paragem da simulação são semelhantes aos descritos para o cálculo do ITAE.

As medidas foram obtidas, para $\sigma = 1$, usando os valores óptimos de k_c, T_i e T_d, previamente obtidos.

Obtidas as medidas, para todos os casos, observou-se que, conforme o atraso l vai aumentando, os valores das medidas de identificação vão-se distanciando dos valores correspondentes em malha aberta. A relação entre o valor analítico e as medidas em malha aberta e fechada é ilustrada na figura 8.

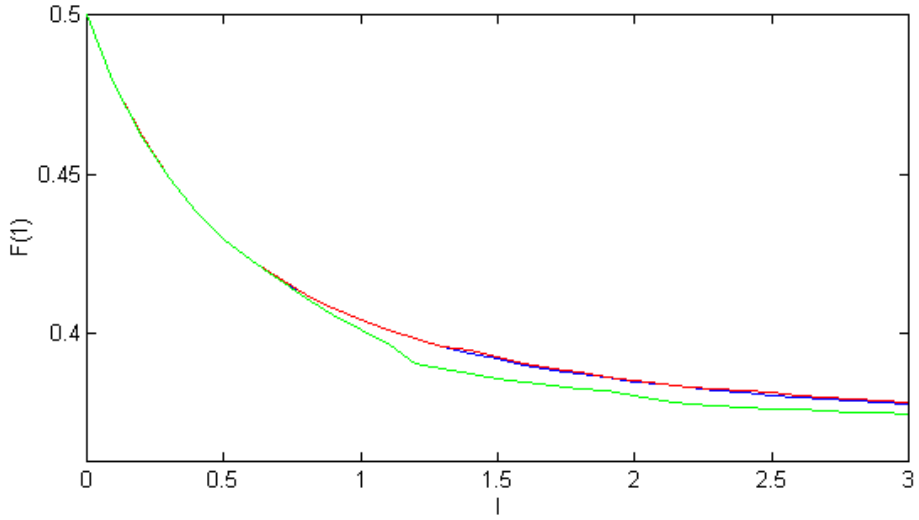


Fig.8 - Medidas de identificação F(1) em função do atraso l , valor analítico __ , medição em malha aberta __ , medição em malha fechada __ .

Procedeu-se também à determinação das medidas de identificação usando valores óptimos de k_c, T_i e T_d de casos próximos, observando-se que, usando os valores óptimos de casos seguintes (atraso l maior), não ocorrem quaisquer problemas, tornando-se os valores de T_T e k_p mais próximos dos analíticos e os valores de $F(\sigma)$ afastando-se mais dos obtidos em malha aberta. No entanto, quando se usam os valores óptimos de casos anteriores, podem surgir problemas. Para processos com um valor baixo de atraso, usar os valores óptimos de dois ou três casos anteriores pode torná-los instáveis.

As medidas $F(\sigma)$ em malha aberta tem valores superiores aos obtidos em malha fechada. Por exemplo, o valor que identifica um processo com atraso $l = 3$, em malha aberta, vai corresponder, em malha fechada, a um sistema com um atraso entre $l = 2.1$ e $l = 2.2$. Assim, se o treino das redes for feito com os valores em malha aberta, a rede vai desconhecer uma certa gama de valores de $F(\sigma)$, mas se a rede for treinada com os valores em malha fechada, quando se faz a identificação em malha aberta, esta vai indicar-nos um sistema com um atraso menor, podendo eventualmente provocar instabilidade.

Para determinar a verdadeira dimensão deste problema procedeu-se a uma simulação, da seguinte forma:

- primeiro, colocaram-se os valores das medidas de identificação, e os valores óptimos dos parâmetros do controlador em vectores de dados, distintos e ordenados da mesma maneira (cinco vectores: F(1) em malha aberta, F(1) em malha fechada, k_c, T_i e T_d) por forma a que uma dada posição dentro de cada vector correspondesse ao mesmo processo;
- seguidamente, conforme a situação a simular, realizou-se uma interpolação polinomial, utilizando as funções de Matlab, para determinação dos parâmetros do controlador PID correspondentes a uma dada medida, considerando que a função a interpolar correspondia ao mapeamento entre a outra medida de identificação e os parâmetros do controlador. Desta maneira pretendeu-se simular a situação em que as redes seriam treinadas com medidas de identificação obtidas em malha aberta, e utilizadas com medidas de identificação obtidas em malha fechada, e vice-versa.

Estes problemas são ilustrados na tabela 1, onde na 1ª coluna representam-se alguns valores do atraso, l, e na 2ª e 3ª colunas os valores da medida de identificação em malha

aberta e fechada (com k_c, T_i e T_d óptimos), respectivamente. Na 4ª coluna indica-se o valor óptimo do ITAE do processo e nas 5ª e 6ª colunas da tabela, o valor do ITAE e sua variação em relação ao óptimo, obtido através de simulação dinâmica com os parâmetros do controlador indicados pela função de interpolação obtida com as medidas em malha aberta, e utilizada com as medidas de identificação em malha fechada (5ª coluna) ou a situação oposta (6ª coluna).

l	F(1) M.A.	F(1) M.F.	ITAE	Treino - M.A Identificação - M.F.	Treino - M.F. Identificação - M.A.
0.1	0.4783	0.47829	0.011819	0.011821 (+0.017 %)	0.011826 (+0.059 %)
0.2	0.46173	0.46161	0.047281	0.047322 (+0.087 %)	0.047393 (+0.237 %)
0.5	0.42998	0.42968	0.294682	0.295235 (+0.188 %)	0.295855 (+0.398 %)
1.0	0.40454	0.40115	1.14996	1.388052 (+20.7 %)	1.27195 (+8.13 %)
1.5	0.39264	0.38601	2.6013	5.24882 (+102 %)	4.43823 (+70.6 %)
2.1	0.3844	0.3792	5.2119	11.6184 (+123 %)	8.8423 (+69.7 %)
2.2 a 3	-	-	-	F(1) desconhecido	de + 70% a + 100 %

Tabela 1 – Comparação dos possíveis efeitos do treino *off-line,* das redes de parâmetros do controlador, com os valores de F(1) obtidos em malha aberta ou fechada, para alguns valores do atraso l

Pelo que se observa, parece não haver problema em treinar a rede com os valores das medidas de identificação em malha fechada, mas como quando se faz a identificação em malha aberta, obtêm-se valores de k_c, T_i e T_d diferentes dos óptimos, as subsequentes medidas de identificação em malha fechada podem ser alteradas. Para verificar se ocorrem problemas, verificaram-se os valores obtidos quando o sistema é primeiramente identificado em malha aberta, e seguidamente em malha fechada, considerando uma interpolação em malha fechada, como se mostra na Tabela 2. Na 1ª coluna apresentam-se alguns valores do atraso, l, na 2ª, os valores da medida de identificação em malha aberta que resultam em valores de k_c, T_i e T_d usados para determinar, através de simulação dinâmica, as medidas em malha fechada, que se

indicam na 3ª coluna, conjuntamente com a variação em relação às óptimas, e que por sua vez vão produzir o ITAE que se indica na 4ª coluna, conjuntamente com a variação em relação ao óptimo.

l	F(1) M.A.	F(1) M.F.	ITAE
0.1	0.4783	0.47829 (aprox. =)	0.011822 (+0.025%)
0.5	0.42998	0.42955 (–0.03%)	0.294864 (+0.062%)
1.0	0.40454	0.402467 (+0.33%)	1.18244 (+2.82%)
1.5	0.39264	0.38707 (+0.27%)	2.77074 (+6.5%)
2.0	0.38549	0.38094 (+0.16%)	4.781575 (+1.22%)
2.5	0.38159	0.37697 (+0.14%)	7.83796 (+6.5%)
3.0	0.378495	0.37518 (+0.096%)	10.97315 (+4.84%)

Tabela 2 - Estudo dos possíveis efeitos de treinar *off-line* as redes com os valores de F(1) obtidos em malha fechada, para alguns valores do atraso l

Pelos resultados obtidos e representados na Tabela 2, torna-se claro que podemos treinar a rede com os valores de $F(\sigma)$ obtidos em malha fechada, sem correr o risco de qualquer problema Para além disso, os erros causados aquando da primeira identificação feita em malha aberta são bastante inferiores aos que se podiam esperar, por análise da Tabela 1, rondando valores que não atingem os 10%, que não são preocupantes já que a identificação em malha aberta se realiza com pouca frequência (assunto mais desenvolvido no próximo capítulo).

4.2 - Redes de parâmetros do controlador

Um dos elementos mais importantes do BIS é o conjunto de três redes neuronais, do tipo *B-spline*, responsáveis pelos parâmetros do controlador. Neste ponto, descreve-se o processo de implementação e treino dessas redes e apresentam-se e comentam-se os resultados obtidos após o treino *off-line* e simulação do treino *on-line*.

Estas redes são de estrutura semelhante, só com uma entrada, F(1), sendo portanto constituídas apenas por funções-base univariáveis, e as suas saídas são correspondentes aos parâmetros k_c, T_i e T_d.

Para que as saídas sejam mais genéricas e não dependam do valor absoluto das constantes de tempo e do atraso do processo a controlar, as saídas das redes responsáveis pelas constantes de tempo serão normalizadas usando o factor de escala T_T, definido em (6). Pela mesma razão, a saída da rede responsável pelo ganho proporcional será normalizada usando o ganho DC do processo, k_p. Como $k_p * k_c$ é, geralmente maior do que um, e para manter a gama de valores de saída pequena, utilizou-se o inverso desta quantidade. Sendo assim, as três redes produzem os seguintes mapeamentos:

$$F(1) \Rightarrow \overline{k_{inv}} = \frac{1}{k_c * k_p} \tag{37.1}$$

$$F(1) \Rightarrow \overline{t_i} = \frac{T_i}{T_T} \tag{37.2}$$

$$F(1) \Rightarrow \overline{t_d} = \frac{T_d}{T_T} \tag{37.3}$$

As redes foram treinadas *off-line* para seis pares de treino, correspondentes a $l = 0.1, 0.4, 0.8, 1.3, 1.9, 2.6$, para se poder analisar as suas capacidades de interpolação e extrapolação. Os pares de treino não estão igualmente espaçados para reflectir, minimamente, o espaçamento existente entre as medidas de identificação correspondentes pois verificou-se que, usando um espaçamento constante em relação a l, as redes não conseguem ser treinadas até se obter um valor de MSE nulo.

4.2.1 - Implementação e treino *off-line*

O modelo inicial é constituído por uma rede com um nó interior, colocado no ponto médio do eixo de entrada, e os exteriores necessários. O algoritmo ASMOD foi então empregue para construir e treinar as redes e como estas são univariáveis, o processo de construção consistiu apenas em introduzir e eliminar funções-base (nós).

Durante a construção das redes, alternaram-se vários passos de refinamento com vários passos de redução, pois observou-se que, se se alternasse um passo de refinamento com um passo de redução, a rede parava de evoluir antes de atingir um MSE nulo, enquanto se se optasse por reduzir apenas quando a rede não melhorasse mais ou quando se atingisse um MSE nulo, a rede por vezes ficava maior do que necessário. Observou-se ainda que por vezes, embora aparentemente não consigamos melhorar a rede, se refinarmos o segundo melhor modelo encontrado, vamos obter um modelo melhor que o inicial. Sendo assim, quando não se consegue melhorar o modelo actual e ainda não se atingiu um MSE nulo, realiza-se um passo suplementar de refinamento e se se conseguir obter um modelo melhor, este é incorporado como modelo actual, prosseguindo-se normalmente com a construção da rede.

Para determinar a ordem das redes a usar, foram construídas redes de 2ª, 3ª e 4ª ordem, observando-se que as redes de 2ª ordem generalizam mal e que as de 4ª ordem generalizam pior e têm um pior desempenho no treino *on-line* que as de 3ª ordem, pelo que foram estas últimas as escolhidas.

O valor da taxa de aprendizagem δ, (eq. 33) varia ao longo do tempo, tomando inicialmente o valor unitário, para que a "aprendizagem" seja rápida e consoante esta se vai tornando mais lenta, o seu valor vai sendo gradualmente reduzido, até que se determine que a rede não evolui mais.

Para construir e treinar as redes de parâmetros do controlador, utilizou-se o programa, em MATLAB, *bspasmod* (Apêndice B.5) cujo diagrama lógico se apresenta na figura 9. Neste diagrama, o termo CIB denota o Critério de Informação Baisiana, definido em (35).

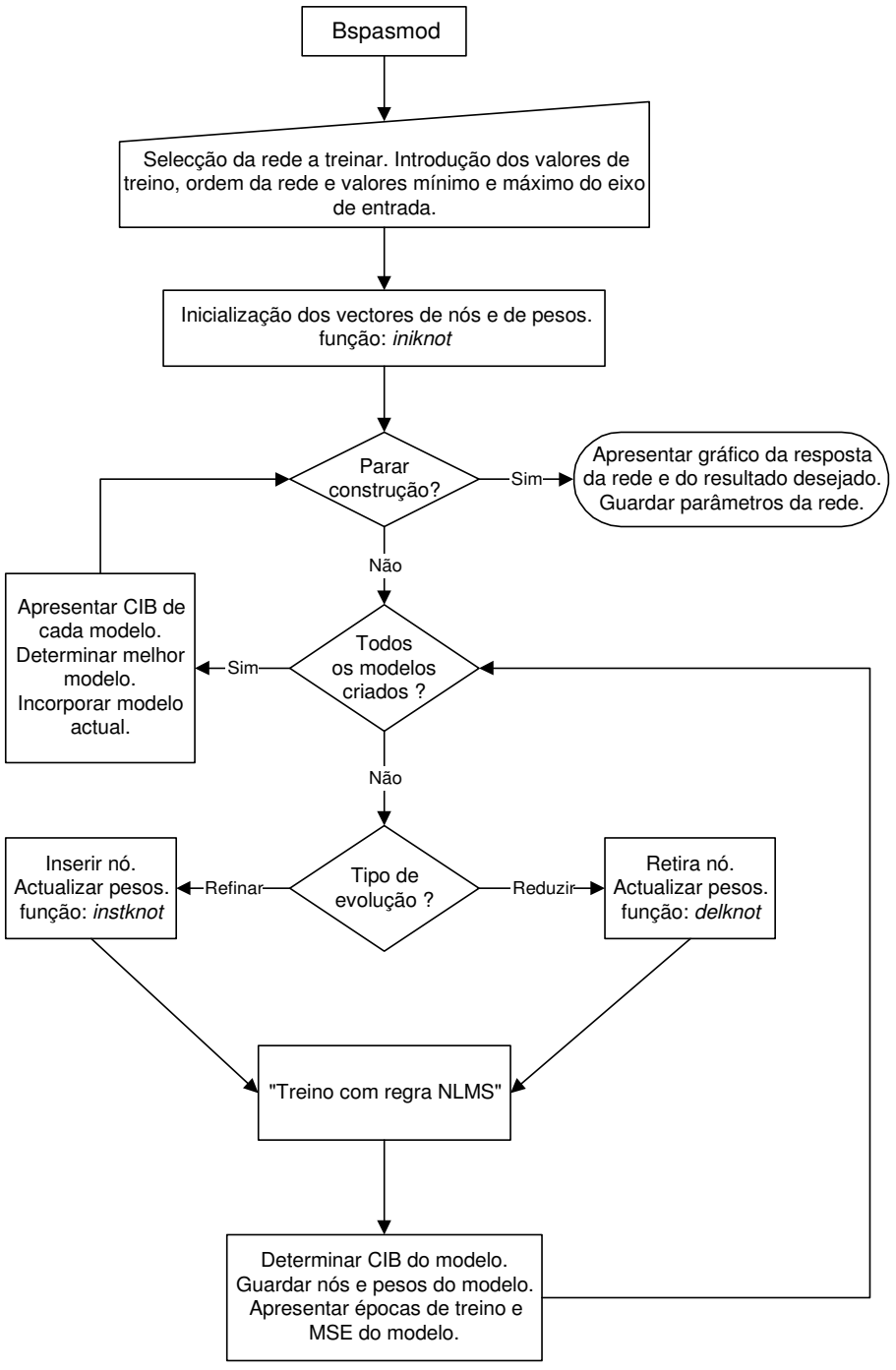


Fig. 9 - Esquema lógico do programa *bspasmod.m*

Em relação às funções de MATLAB referidas na figura 9, temos que:

- A função *iniknot* (Apêndice B.6) inicializa os vectores de nós (cap. 3.1 - eq. 19, 20.1 e 20.2), colocando nós nos valores mínimo e máximo do eixo de entrada, um nó a meio destes valores e nós exteriores (quantos forem necessários) com uma separação inversamente proporcional à quantidade de valores considerados significativos nesse eixo;
- A função *instknot* (Apêndice B.7) introduz um nó num vector de nós, a meio de dois nós preexistentes e acrescenta um peso ao vector dos pesos, peso esse que é a repetição do que estava relacionado com esse intervalo antes de ser seccionado, ou seja, o resultado será passar a existirem dois pesos idênticos consecutivos;
- A função *delknot* (Apêndice B.8) retira um nó num vector de nós e substitui os dois pesos correspondentes aos intervalos que foram fundidos por um peso de valor igual à média desses dois.

O bloco designado "Treino usando a regra NLMS" apresenta-se, em separado, na figura 10, por uma questão de clareza. Em relação às funções referidas, temos que:

- A função *ambf* (Apêndice B.9), determina os valores das várias funções-base univariáveis activadas por um certo valor de entrada x, utiliza ainda as seguintes funções: *aad* (Apêndice B.10) que determina o intervalo do eixo de entrada a que x pertence, *ubf1* (Apêndice B.11) que determina o valor da função-base univariável de 1ªordem (eq. 23) e *ubf2* (Apêndice B.12) que determina o valor das funções-base univariáveis de 2ª ordem e superior activadas por x (eq. 22);
- A função *aprendn* (Apêndice B.13) inicializa e actualiza o vector de pesos usando a regra NLMS (eq. 34); e
- A função *condi* (Apêndice B.14) determina como o erro de aprendizagem está a evoluir, ou seja, se está a decrescer, a crescer, parado ou a decrescer muito

pouco, e faz diminuir o valor da taxa de aprendizagem δ, se se verificar qualquer uma das três últimas situações referidas.

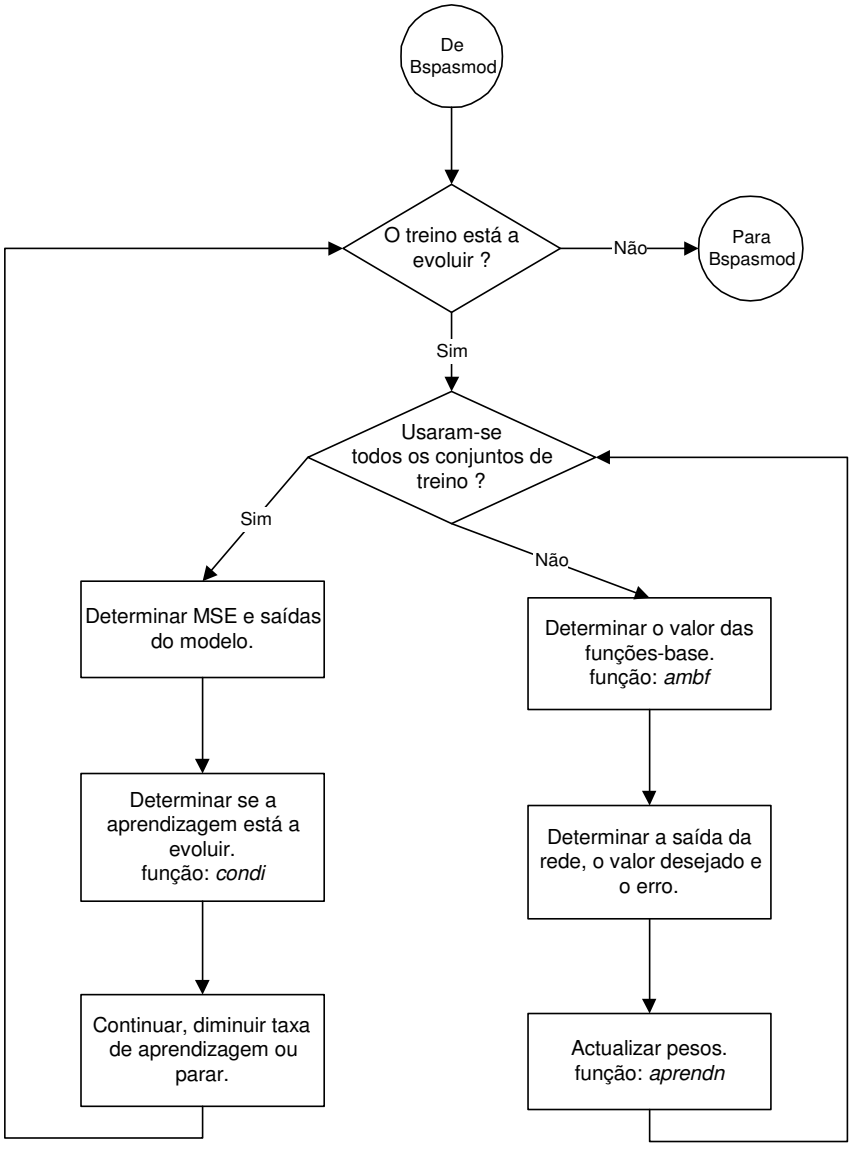
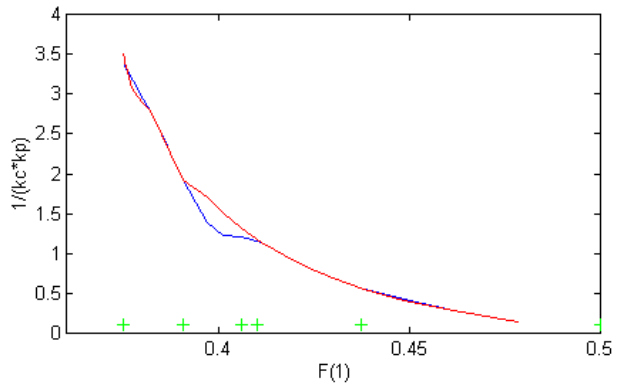


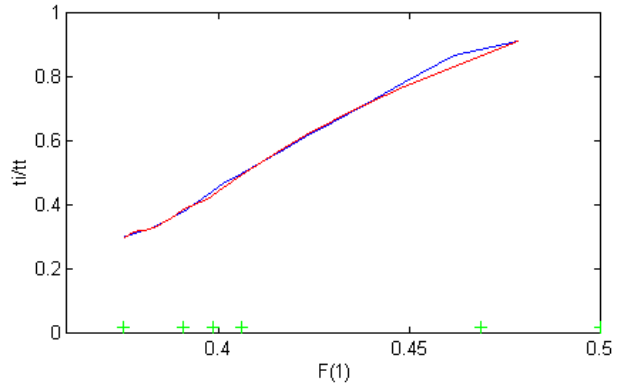
Fig. 10 – Esquema lógico do módulo "Treino com regra NLMS" do programa *bspasmod.m*

Terminado o treino *off-line* obtiveram-se os resultados que se apresentam na figura 11, onde se comparam os resultados obtidos pelas redes com os resultados desejados e também se indica a localização dos nós das redes.

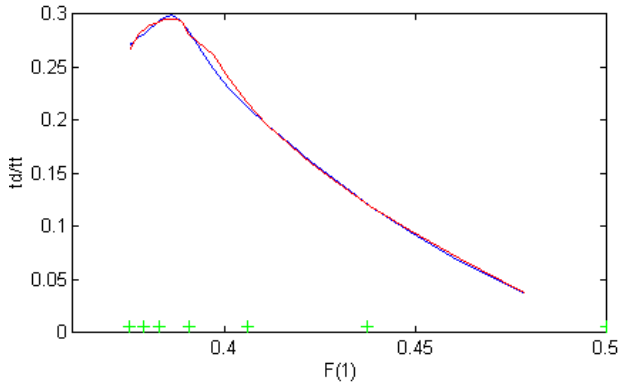
Verificou-se que, as redes "aprenderam" completamente o conjunto de treino fornecido, ou seja, o MSE final sobre o conjunto de treino foi exactamente nulo, mas em relação ao conjunto total (trinta casos), as redes apresentaram um MSE final de 2,729 na rede $\overline{k_{inv}}$, de 0,05026 na rede $\overline{t_i}$ e de 0.00661, na rede $\overline{t_d}$.



a)



b)



c)

Fig. 11 - Mapeamento das redes após o treino *off-line*, (__ m. desejado, __ m. obtido, + nós)

(a) $F(1) \Rightarrow \overline{k_{inv}} = \frac{1}{k_c * k_p}$, **(b)** $F(1) \Rightarrow \overline{t_i} = \frac{T_i}{T_T}$, **(c)** $F(1) \Rightarrow \overline{t_d} = \frac{T_d}{T_T}$

4.2.2 - Simulação de treino *on-line*

Terminado o treino *off-line*, foi realizada uma simulação de treino *on-line*, para verificar o seu comportamento. A rede obtida com o treino *off-line* foi treinada para todo o conjunto de treino, referente aos trinta casos estudados, durante várias épocas. Foram experimentadas as diferentes taxas de aprendizagem $\delta = 0.5, 0.2$ e 0.1, observando-se que os melhores resultados foram obtidos com o valor mais pequeno das mesmas. Para além disso, com o valor $\delta = 0.5$, as redes tendiam a evoluir rapidamente nas primeiras épocas e depois começavam a oscilar sem evoluir mais.

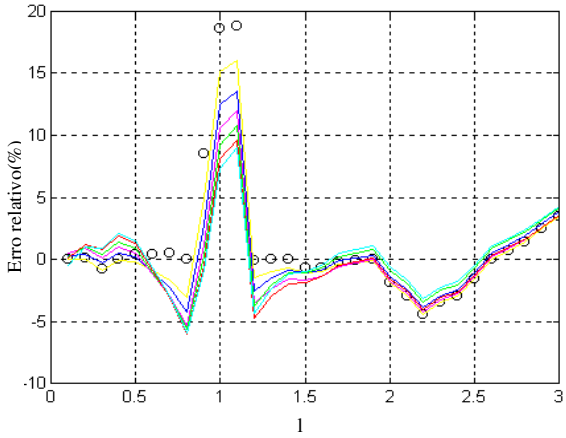
Na Tabela 3, apresentam-se os resultados obtidos após o treino *on-line* das redes, durante seis épocas e usando $\delta = 0.1$. Para cada rede, apresentam-se três medidas de qualidade da aproximação, a soma dos quadrados dos erros (s.q.e), esta medida escalada pela soma dos quadrados dos valores desejados dos parâmetros e a soma dos quadrados dos erros relativos (s.q.e.r.). A unidade usada na tabela é 10^{-3}.

	$\overline{k_{inv}}$			$\overline{t_i}$			$\overline{t_d}$		
Época	s.q.e	$\frac{\text{s.q.e}}{\sum \overline{k_{inv}}^2}$	s.q.e.r.	s.q.e	$\frac{\text{s.q.e}}{\sum \overline{t_i}^2}$	s.q.e.r	s.q.e	$\frac{\text{s.q.e.}}{\sum \overline{t_d}^2}$	s.q.e.r
Inicial	272.9	3.276	85.148	2.0937	0.0714	7.0907	0.4571	0.0290	8.5325
1	213.0	2.557	60.792	1.8287	0.0623	6.3464	0.3450	0.0219	6.5648
2	170.3	2.044	45.324	1.6961	0.0578	6.0072	0.2924	0.0185	5.7646
3	157.0	1.884	39.700	1.6510	0.0563	5.8877	0.2442	0.0155	5.1010
4	136.1	1.633	33.178	1.6048	0.0547	5.8032	0.2219	0.0141	4.8344
5	134.6	1.616	32.055	1.5880	0.0541	5.7224	0.1952	0.0124	4.5429
6	121.1	1.453	27.898	1.5633	0.0533	5.6902	0.1848	0.0117	4.5125

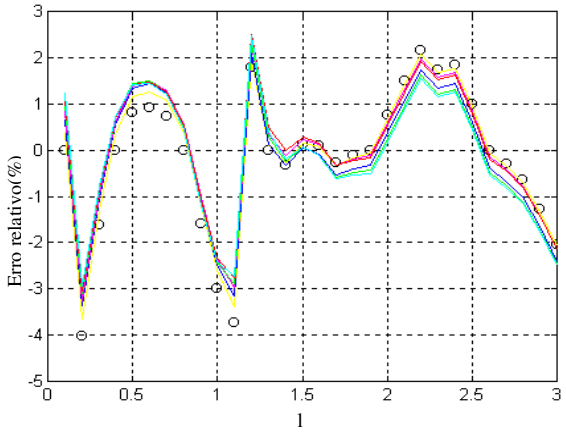
***Tabela 3 - Evolução dos resultados com o treino* on-line**

Observa-se facilmente que, a partir do estado apenas com treino *off-line* (linha inicial), verifica-se um decréscimo permanente em todas as medidas usadas.

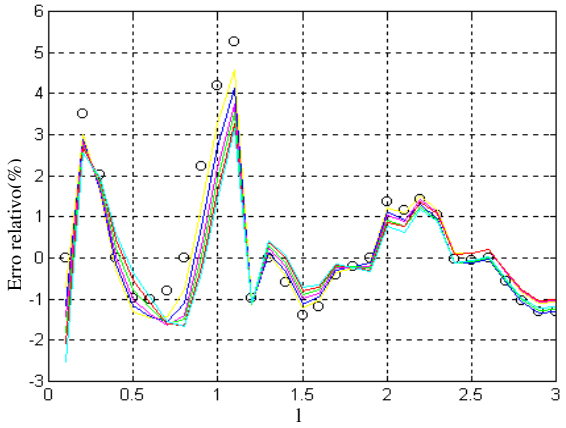
A evolução dos erros relativos para cada elemento do conjunto de treino é também ilustrada na figura 12.



a)



b)



c)

Fig. 12 - Evolução dos erros relativos, após treino *on-line* nas redes a) $\overline{k_{inv}}$, b) $\overline{t_i}$ e c) $\overline{t_d}$.
o – inicial; épocas: 1ª, 2ª, 3ª, 4ª, 5ª e 6ª

É de salientar que a taxa de aprendizagem mostrou ser um factor de grande importância, no treino *on-line*. Dado que só se utilizaram valores óptimos, é ainda necessário, antes de colocar as redes no sistema final, determinar o valor a usar nesse caso, já que nos primeiros tempos estas redes estarão a ser adaptadas com valores talvez, bastante diferentes dos óptimos (assunto a desenvolver no próximo capítulo).

4.3 – Rede de modelização do ITAE

Para construção e treino *off-line* da rede de minimização do ITAE foram gerados 810 padrões de treino, sendo cada um composto por valores correspondentes a $F(1)$, k_c, T_i, T_d e ITAE. Para cada um dos trinta valores da medida de identificação, foram calculados os valores do ITAE correspondentes às vinte e sete (3^3) combinações possíveis dos parâmetros k_c, T_i e T_d , assumindo cada um deles o seu valor óptimo, 75% e 125% desse valor.

Os valores k_c, T_i, T_d foram escalados da forma e pelas razões descritas em 4.2. Os valores do ITAE também foram escalados pelo quadrado do factor de escala T_T, sendo o quadrado devido ao ITAE estar relacionada quadraticamente com o tempo t. Para além disto, sendo o objectivo desta rede modelar o mais fielmente possível os valores mínimos do ITAE, os erros cometidos, na aprendizagem, próximo destes valores podem ser pequenos mas relativamente grandes comparados com os valores desejados, pelo que para valorizar estes erros vai-se usar o inverso do valor escalado do ITAE.

Pelo que se expôs, o valor correspondente ao ITAE, no conjunto de treino, será:

$$\overline{\text{ITAE}} = \frac{(T_T)^2}{\text{ITAE}} \tag{38}$$

4.3.1 - Implementação e treino *off-line*

A implementação e treino *off-line* desta rede, revelou-se muito mais complicada, e morosa do que seria de esperar a priori.

Inicialmente, implementaram-se algumas redes, usando a regra NLMS, descrita em 3.2, e o algoritmo ASMOD, descrito em 3.3.

Mais concretamente, desenvolveu-se uma aplicação que começava por:

- treinar cada variável de entrada para o seu valor médio;
- juntar um par de entradas e treiná-las, primeiro, aditivamente e depois como uma sub-rede multi-variável, escolhendo a melhor, e;
- juntar uma terceira variável, primeiro aditivamente com a rede anterior e depois, formando sub-redes multi-variável, escolhendo a melhor, e;
- da mesma forma juntar a última variável e proceder da forma atrás descrita, resultando finalmente numa das seguintes estruturas: quatro redes aditivas univariáveis ou, duas redes univariáveis e uma multi-variável, etc., ou na pior das hipóteses, uma só rede multi-variável com quatro entradas.

Sobre cada eixo de entrada definiram-se funções-base de 3ª ordem, pelas razões indicadas em 4.2.1., e realizaram-se algumas experiências com diferentes quantidades de nós interiores, para comparar resultados.

O primeiro problema que surgiu foi que, esta forma de implementação, levava várias centenas de horas, num computador normal, para construir uma rede, e para além disso com resultados muito maus.

Para obviar o problema da morosidade, resolveu-se substituir a regra de aprendizagem inicial (33), pela solução, em termos de mínimos quadráticos, da equação (30), estendida a todo o conjunto de treino, ou seja, determinar os pesos a partir de:

$$\mathbf{w} = [\mathbf{A}]^{+}\,\hat{\mathbf{y}} \tag{39}$$

em que o sobrescrito "+" denota uma pseudo inversa, e

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}^{\mathrm{T}}(1) \\ \mathbf{a}^{\mathrm{T}}(2) \\ \cdots \\ \mathbf{a}^{\mathrm{T}}(\mathrm{m}) \end{bmatrix} \tag{40}$$

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{\mathbf{y}}(1) \\ \hat{\mathbf{y}}(2) \\ \cdots \\ \hat{\mathbf{y}}(\mathrm{m}) \end{bmatrix} \tag{41}$$

Esta solução não causa problemas no ponto de vista da aprendizagem, pois como é explicado em 3.2., é a equação (30) que tem que ser obrigatoriamente satisfeita, e as seguintes são apenas uma hipótese de solução, para além de que no processo de aprendizagem *on-line* pode-se continuar a usar a regra NLMS (que aí sim não deve ser substituída). No entanto, a nível do algoritmo ASMOD, vai-se transgredir a questão de cada modelo reproduzir exactamente o anterior, embora neste caso, os fins justifiquem os meios, já que para além de se ter conseguido reduzir o tempo de construção duma rede em mais do que cinco vezes, também se conseguiram resultados de aproximação muito melhores.

Seguidamente, descobriu-se que a ordem pela qual se vão juntando as variáveis de entrada afecta consideravelmente o resultado final, pelo que se tentou uma abordagem em que se experimentavam todas as combinações, desde quatro sub-redes univariáveis, duas univariáveis e uma multi-variável, etc.

Para além disso experimentou-se treinar redes com funções-base de 2ª ordem, obtendo em alguns casos resultados melhores do que usando funções-base de 3ª ordem. Alguns destes primeiros resultados encontram-se representados na Tabela 4, onde nas 1as quatro linhas representam-se redes construídas juntando duas entradas, seleccionando a melhor combinação (univariável ou multi-variável) e depois

adicionando a 3ª e 4ª entradas. As duas últimas representam redes que foram iniciadas como quatro redes univariáveis e depois experimentaram-se todas as combinações de duas redes univariáveis e uma multi-variável, duas multi-variável, etc. Os resultados apresentados para a 1ª rede foram obtidos usando a regra de aprendizagem NLMS, enquanto as restantes foram treinadas com a regra definida pela equação (39). A melhor estrutura obtida em cada caso é representada usando o sinal "*" para denotar uma sub-rede multi-variável e o sinal "+" para denotar a adição de duas sub-redes. As medidas usadas para comparar as redes são o MSE e os valores máximo e médio do valor absoluto dos erros relativos sobre todo o conjunto de treino.

Ordem de int. das entradas	Regra de Aprendiz.	Ordem	Melhor estrutura	M.S.E	Máx. $\lvert$erro rel. %$\rvert$	Média $\lvert$erro rel. %$\rvert$
$F(1), \overline{k_{inv}}, \overline{T_i}, \overline{T_d}$	N.L.M.S	3ª	$F(1) + \overline{k_{inv}} + \overline{T_i} + \overline{T_d}$	2.49	282	34.6
$F(1), \overline{k_{inv}}, \overline{T_i}, \overline{T_d}$	Eq. (39)	3ª	$F(1) * \overline{T_d} + \overline{k_{inv}} * \overline{T_i}$	0.2676	274	23
$\overline{T_d}, \overline{T_i}, \overline{k_{inv}}, F(1)$	Eq. (39)	3ª	$F(1) * \overline{T_d} + \overline{k_{inv}} * \overline{T_i}$	0.2251	218	18.4
$\overline{T_d}, \overline{T_i}, \overline{k_{inv}}, F(1)$	Eq. (39)	2ª	$F(1) * \overline{k_{inv}} + \overline{T_i} * \overline{T_d}$	1.286	114	29.8
Todas comb.	Eq. (39)	3ª	$F(1) * \overline{k_{inv}} + \overline{T_i} * \overline{T_d}$	1.29	537	30.8
Todas comb.	Eq. (39)	2ª	$F(1) * \overline{T_d} + \overline{k_{inv}} * \overline{T_i}$	0.3547	312	20.1

Tabela 4 - Melhores resultados obtidos usando diferentes estratégias de treino da rede de modelização do ITAE

Como se pode observar, em termos de MSE, a melhor rede obtida é a terceira, na qual se utilizam funções-base de 3ª ordem e a melhor estrutura é composta pela adição de duas sub-redes, $F(1) * \overline{T_d}$ e $\overline{k_{inv}} * \overline{T_i}$. Para além disso, pode-se observar que embora para a maioria dos casos exista uma relação notória entre o MSE e os erros relativos,

nem sempre assim acontece (estes desenvolvimentos encontram-se sumariados em [18]).

A observação desta não correspondência entre o MSE e o erro relativo, bem como a observação do processo de crescimento das redes (Apêndice C.1), onde nem sempre é a rede com melhores erros relativos que é seleccionada, levou a que se experimentasse substituir no critério de desempenho, eq. (35), o MSE pelo valor médio quadrático do erro relativo. Esta abordagem revelou-se desastrosa, já que para além de os resultados obtidos serem muito piores, a maioria das redes terminava o seu treino num valor médio quadrático do erro relativo de 0.5, pois ao passarem perto deste valor, as redes começavam a fazer a sua saída tender para zero, mantendo o erro relativo permanentemente a 1.

Outra abordagem que se tentou foi a de, para a estrutura que anteriormente deu melhores resultados ($F(l) * \overline{T_d} + \overline{k_{inv}} * \overline{T_i}$), desenvolver redes para todas as combinações possíveis de funções-base de 2ª, 3ª e 4ª ordem, obtendo-se os resultados que se apresentam no Apêndice C.2, dos quais as melhores cinco hipóteses são representadas na Tabela 5. Na 1ª coluna apresenta-se a ordem das funções-base de cada eixo de entrada, na 2ª o valor da medida de desempenho Bayk (eq. 35), o MSE é apresentado na 3ª coluna e os valores médio e máximo do valor absoluto do erro relativo, em percentagem, são apresentados nas 4ª e 5ª colunas. Para se ter uma certeza mais objectiva do desempenho das redes, foi-se ainda determinar, para cada um dos trinta casos (concretizações do atraso l), se a rede, independentemente do valor da saída, indicava o mínimo correspondente aos k_c, T_i e T_d ditos óptimos. Para tal utilizaram-se os 810 padrões de treino, e para cada grupo de saídas, correspondente a uma certa concretização do atraso l, determinou-se onde se encontrava o mínimo, e se era dado para a correspondência (ITAE - k_c, T_i e T_d) correcta. Sendo assim, na 6ª coluna da

Tabela 5, indica-se a quantidade de mínimos correctamente mapeados (q.m.c.m). Para escolher as ditas cinco melhores hipóteses , seleccionou-se a melhor, segundo a medida indicada em cada uma das colunas, (em negrito na Tabela 5 e no Apêndice C. 2).

Como se pode observar da Tabela 5, é difícil conseguir descobrir uma correspondência entre qualquer uma das medidas usadas, tornando bastante difícil determinar a qualidade duma certa rede, em absoluto ou comparativamente com as restantes.

Ordem ($F(1), \overline{k_{inv}}, \overline{T_i}, \overline{T_d}$)	Bayk	MSE	Méd. erro Relativo (%)	Máx. erro relativo (%)	q.m.c.m
2,2,3,4	**-680.8**	0.236	21.8	384	5
2,3,2,3	-600.9	0.2282	22.1	585.4	**19**
2,3,4,2	-572.8	0.2503	16.07	**67.1**	0
2,4,2,3	-589.9	**0.2077**	24.7	376.5	16
4,3,4,2	-538.2	0.2425	**15.3**	72.87	1

Tabela 5 - Melhores hipóteses encontradas para a estrutura $F(1) * \overline{T_d} + \overline{k_{inv}} * \overline{T_i}$ ***, treinada para todas as combinações de ordens das redes***

Embora estes resultados, já sejam aparentemente melhores que os anteriores, ainda se tentou mais uma nova abordagem, criando uma rede onde as quatro variáveis de entrada pudessem aparecer mais do que uma vez, experimentando juntar todas as hipóteses de redes univariável e multi-variável de duas entradas, com todas as combinações de ordens das funções-base. Mais concretamente, começou-se por treinar cada entrada sozinha para os seus valores mínimos, experimentando 2ª, 3ª e 4ª ordens das funções-base, e escolhendo a melhor para cada entrada. Seguidamente, treinou-se cada combinação possível de duas entradas, também para as mesmas ordens de funções-base, e escolheu-se a melhor de cada combinação. Tomando como modelo inicial, a melhor sub-rede de entre as melhores anteriormente determinadas, começou-se a juntar

aditivamente cada uma das restantes, escolhendo a melhor hipótese, juntando a 3ª sub-rede, etc., até não se conseguir obter um modelo melhor. A melhor rede assim obtida tinha a estrutura $\overline{k_{inv}} + F(1) * \overline{T_i} + \overline{k_{inv}} * \overline{T_i} + \overline{k_{inv}} * \overline{T_d}$, com as ordens 2, 2, 2, 3, 4, 3 e 2, respectivamente.

Experimentou-se, ainda forçar uma rede, com todas as funções-base de 2ª ordem (não se conseguiu desenvolver funções de ordem superior nos computadores disponíveis) a crescer até atingir a forma multi-variável de quatro entradas. Os resultados obtidos, bem como os da rede descrita no parágrafo anterior, são apresentados na Tabela 6, onde se utilizam as mesmas medidas que na Tabela 5.

Rede	Bayk	MSE	Méd. erro Relativo (%)	Máx. erro relativo (%)	q.m.c.m
$\overline{k_{inv}} + F(1) * \overline{T_i} + \overline{k_{inv}} * \overline{T_i} + \overline{k_{inv}} * \overline{T_d}$	-278.9	0.1826	14.26	58.07	1
$F(1) * \overline{k_{inv}} * \overline{T_i} * \overline{T_d}$	-141.2	0.39915	18.86	96.07	0

Tabela 6 - Outras abordagens de treino da rede de modelização do ITAE

Nesta tabela pode-se observar que a 1ª rede é melhor do que qualquer uma das da Tabela 5 em três das medidas utilizadas, e que a rede multi-variável de quatro entradas é muito pior que as restantes em relação ao Bayk e ao MSE e nas outras medidas, enquadra-se vagamente no mesmo nível.

Importa aqui salientar que o trabalho descrito até aqui, neste sub-capítulo, demorou alguns milhares de horas de trabalho e de computação a ser realizado.

Dada a dificuldade em decidir qual ou quais medidas a usar para determinar a melhor rede, decidiu-se que se empregaria a que funcionasse melhor no seu enquadramento, isto é, no sistema final. Sendo assim, foi-se utilizar cada uma das redes acima (à excepção da última, por questões de esforço computacional) para minimização, com a função *fmins* da *Optimization Toolbox* [16]. Utilizou-se esta função em vez da

anteriormente utilizada, porque, como se descreve em 4.1, não se consegue utilizar essa função com segurança sem uma constante observação e assistência, sendo portanto imprópria para ser introduzida num sistema automático.

Os resultados obtidos, para várias concretizações do atraso l, revelaram que a medida mais relacionada com a qualidade efectiva da rede é a q.m.c.m (quantidade de mínimos correctamente mapeados), obtendo-se com todas as redes (à excepção das 2ª e 4ª redes da Tabela 5) resultados verdadeiramente absurdos e disparatados, a maioria dos quais iriam certamente provocar uma instabilização do sistema. Pelo acima exposto, escolheu-se como rede a integrar no sistema final a 2ª rede da Tabela 5.

A construção e treino desta rede foram implementadas usando o algoritmo ASMOD, pelo programa em MATLAB *bspitasu* (Apêndice B.15), cujo funcionamento já se explicou anteriormente. Este programa utiliza as funções *treino4* (Apêndice B.16), e *form_a* (Apêndice B.17) e as funções *iniknot* e *ambf* (comuns ao programa *bspasmod*), e ainda as funções associadas a *ambf*, já descritas em 4.2.1. O programa *bspitasu* define a estrutura de cada modelo, e implementa o processo em geral (de forma semelhante à do programa *bspasmod*, descrito em 4.2.1). A função *treino4* implementa o processo de aprendizagem, usando a eq. (39), e utiliza a função *ambf,* para determinar o valor das saídas das funções-base para um dado padrão de entrada e a função *form_a,* para formar a matriz **A** (eq. 40) com todos os valores das saídas das funções-base correspondentes a todo o conjunto de treino.

4.3.2 - Simulação de treino *on-line*

Para verificar o desempenho da adaptação *on-line*, foram experimentadas várias estratégias, tendo no entanto todas em comum o facto de que os resultados não são observados directamente da rede, mas sim através da função de minimização, já que será desta forma que esta rede será utilizada no sistema final.

A adaptação é realizada pela função em MATLAB *t_o_itae* (Apêndice B.18), que emprega a regra de aprendizagem NLMS, implementada pela função *aprendn* (já descrita em 4.2.1). Foram também experimentadas várias taxas de aprendizagem $\delta = 0.05, 0.01, 0.005$, verificando-se que para a maioria dos casos o valor $\delta = 0.01$ é o mais indicado, já que com um valor superior, para muitos casos não há convergência e com um valor inferior, a adaptação é muito demorada.

A estratégia encontrada que permite uma melhor análise dos resultados foi fornecer, para cada caso (concretização do atraso l), os valores óptimos dum caso contíguo como valor inicial para realizar a optimização, efectuar a optimização, e seguidamente adaptar a rede com os valores óptimos correspondentes ao caso de estudo, e repetir todo o processo, começando sempre a minimização a partir do mesmo valor inicial.

Na figura 13, mostram-se os resultados obtidos ao longo de 80 épocas, da adaptação realizada para um processo com $l = 1.7$, e com a minimização iniciada sempre com os valores óptimos de $l = 1.6$. Em cada gráfico da figura, assinala-se a tracejado o valor óptimo do parâmetro ou da medida correspondente. Os valores do ITAE (fig. 13 d)), são determinados por simulação usando a função *itaef* (descrita em 4.1), com os valores dos parâmetros obtidos da minimização. Nos gráficos dos parâmetros (fig. 13 a),b) e c)), a linha a traço-ponto indica o valor fornecido como ponto inicial.

Como nos gráficos dos parâmetros $\overline{T_i}$ e $\overline{T_d}$ parece não existir evolução, na figura 13, apresentam-se os valores dos erros relativos dos parâmetros e ITAE. Nesta figura, pode-se observar que existe evolução, embora na generalidade seja muito lenta e para o parâmetro $\overline{T_i}$ seja quase nula.

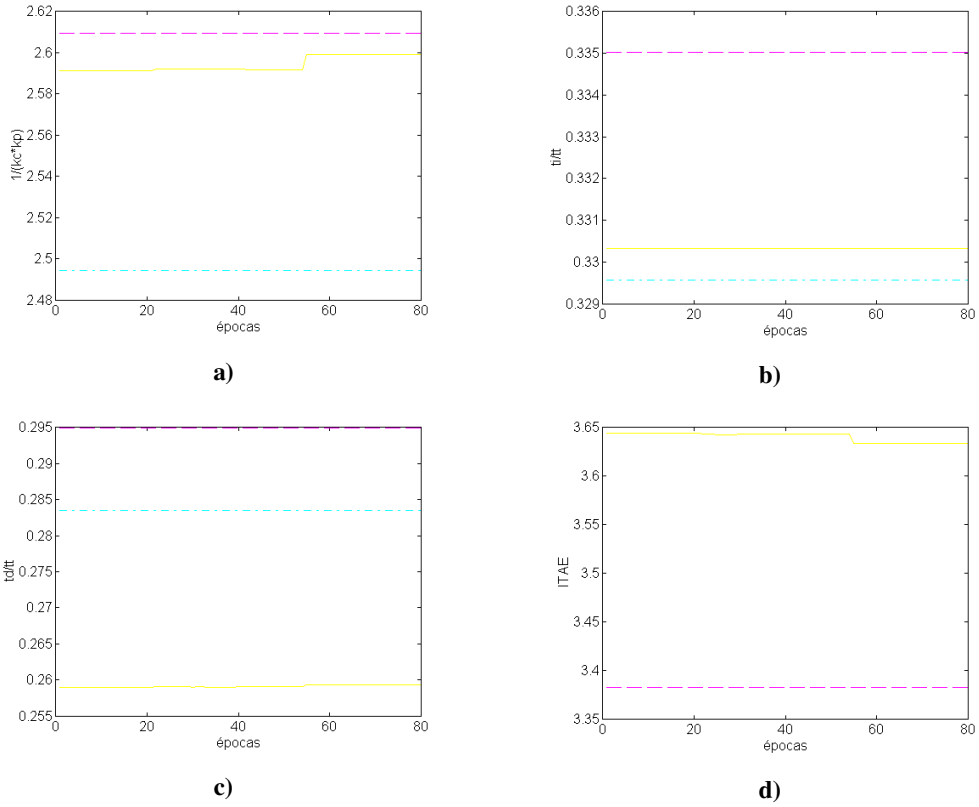


Fig. 13 - Evolução dos parâmetros e ITAE, após adaptação *on-line,* da rede de modelização do ITAE para $l = 1.7$. a) $\overline{k_{inv}}$, b) $\overline{t_i}$, c) $\overline{t_d}$ e d) ITAE

Nas figuras 15 e 16, reproduzem-se os resultados da adaptação realizada para um processo com $l = 0.9$, e com a minimização iniciada sempre com os valores óptimos de $l = 0.8$, obtidos ao longo de 100 iterações.

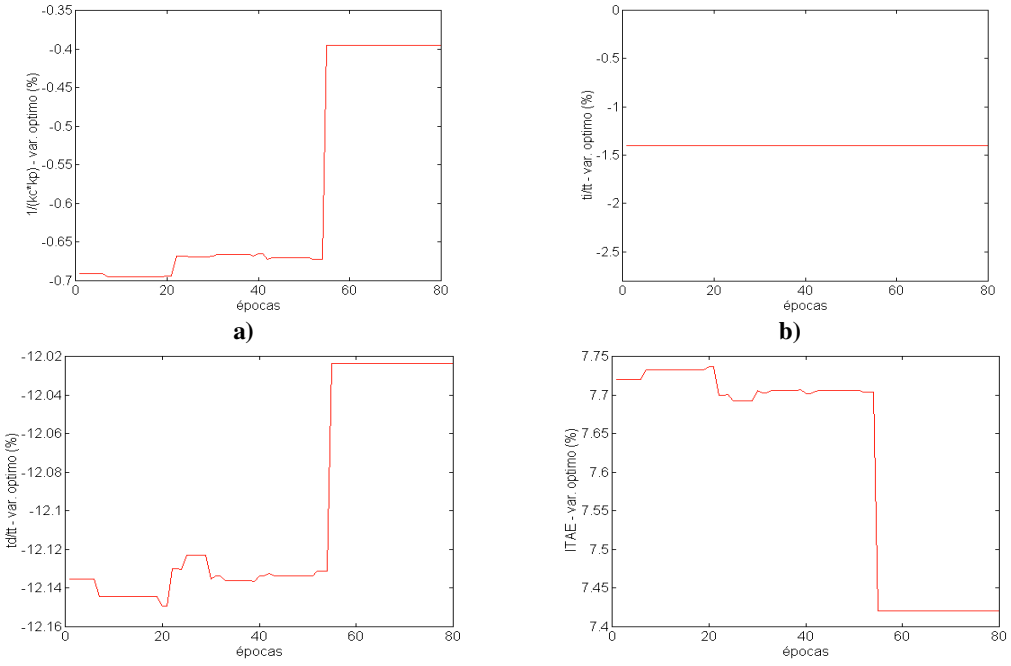


Fig. 14 - Evolução dos erros relativos dos parâmetros e ITAE, após adaptação *on-line,* da rede de modelização do ITAE, para $l = 1.7$ . . a) $\overline{k_{inv}}$, b) $\overline{t_i}$, c) $\overline{t_d}$ e d) ITAE

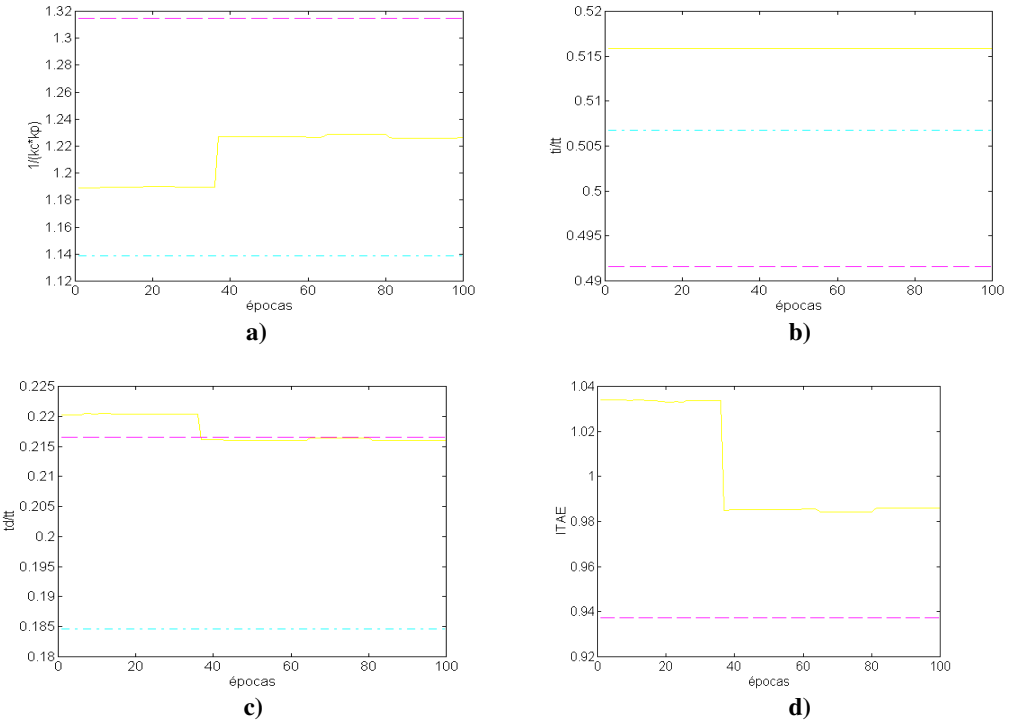


Fig. 15 - Evolução dos parâmetros e ITAE, após adaptação *on-line,* da rede de modelização do ITAE para $l = 0.9$. a) $\overline{k_{inv}}$, b) $\overline{t_i}$, c) $\overline{t_d}$ e d) ITAE

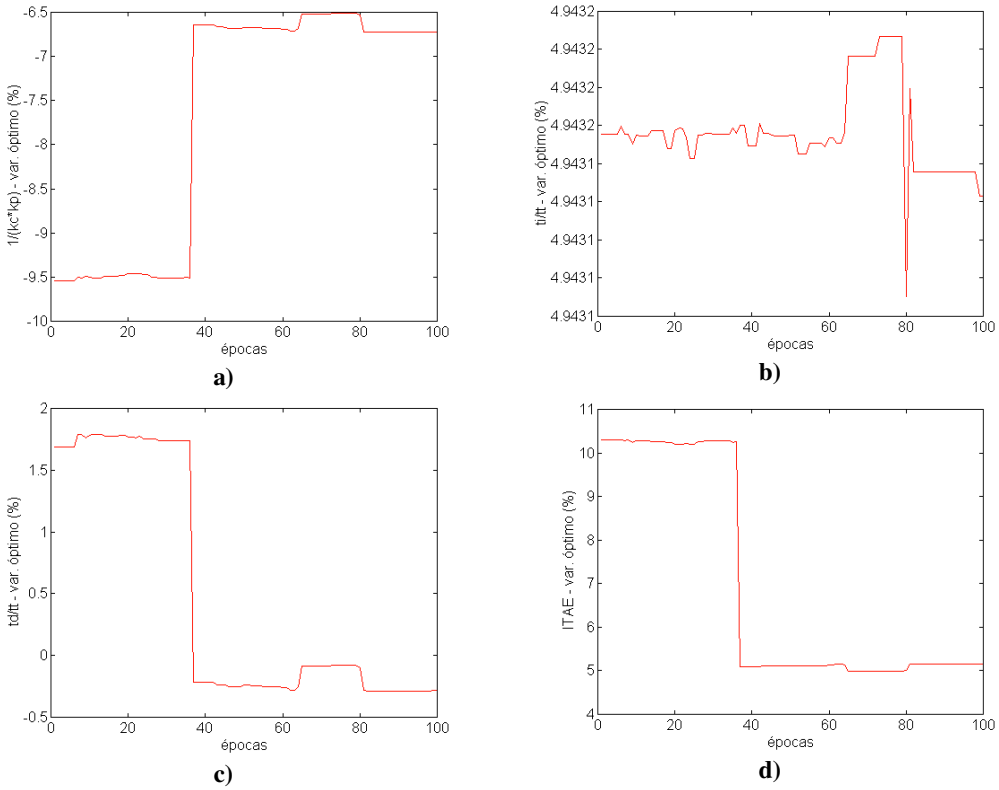


Fig. 16 - Evolução dos erros relativos dos parâmetros e ITAE, após adaptação *on-line,* da rede de modelização do ITAE, para $1 = 0.9$ **. . a)** $\overline{k_{inv}}$ **, b)** $\overline{t_i}$ **, c)** $\overline{t_d}$ **e d) ITAE**

Analisando as figuras 13 a 16, pode-se observar que para todas as variáveis de interesse (os parâmetros do controlador e o ITAE), à excepção do tempo integral, é conseguida convergência, embora lenta. Procurando explicar os resultados da não evolução deste último parâmetro, analisaram-se os pesos da rede utilizada, tendo-se chegado à conclusão de que a última camada é mal condicionada numericamente, isto é, apresenta valores muito grandes e dentro de uma larga gama para os seus elementos. Infelizmente, este facto é uma consequência da utilização da regra de aprendizagem (39), que, quando não supervisionada a sua execução relativamente à característica da matriz **A**, pode conduzir a resultados como este. O processo de aprendizagem *off-line* utilizando a solução de mínimos quadrados para os pesos lineares deverá, assim, ser tema para um estudo mais aprofundado, de modo a minorar este problema.

Capítulo 5

SISTEMA FINAL

Concluído o treino das redes, todos os componentes são ligados de modo a implementar o sistema final.

Este sistema deverá funcionar da seguinte forma:

1º passo – dado um processo desconhecido, é feita a sua identificação em malha aberta;

2º passo – a partir dessa identificação, determinam-se os parâmetros do controlador, usando as redes neuronais correspondentes; seguidamente a malha é fechada;

3º passo – aguarda-se pela próxima alternância do sinal de entrada e é então realizada uma identificação em malha fechada e medido o valor do ITAE obtido. Este valor conjuntamente com os parâmetros do controlador, é utilizado para adaptar a rede responsável pela modelização do ITAE;

4º passo – determinam-se os valores óptimos do controlador PID usando a rede de modelização do ITAE e, se forem diferentes dos fornecidos ao controlador, são usados para treinar as redes dos parâmetros do controlador.

A cada alternância do sinal de entrada repetem-se os passos de 2 a 4 e se por algum acaso o sistema se torna instável, a malha é aberta e volta-se ao 1º passo. Se no intervalo entre transições do sinal de entrada se detecta uma alteração do processo a controlar, esse intervalo não é usado para efeitos de identificação e aguarda-se a próxima transição.

5.1 - Modelo do sistema final

O sistema final é implementado pelo modelo em SIMULINK *sistfim* (Apêndice A.4) e que se representa nas figuras 17.1 e 17.2 e que foi adaptado a partir do utilizado em [19].

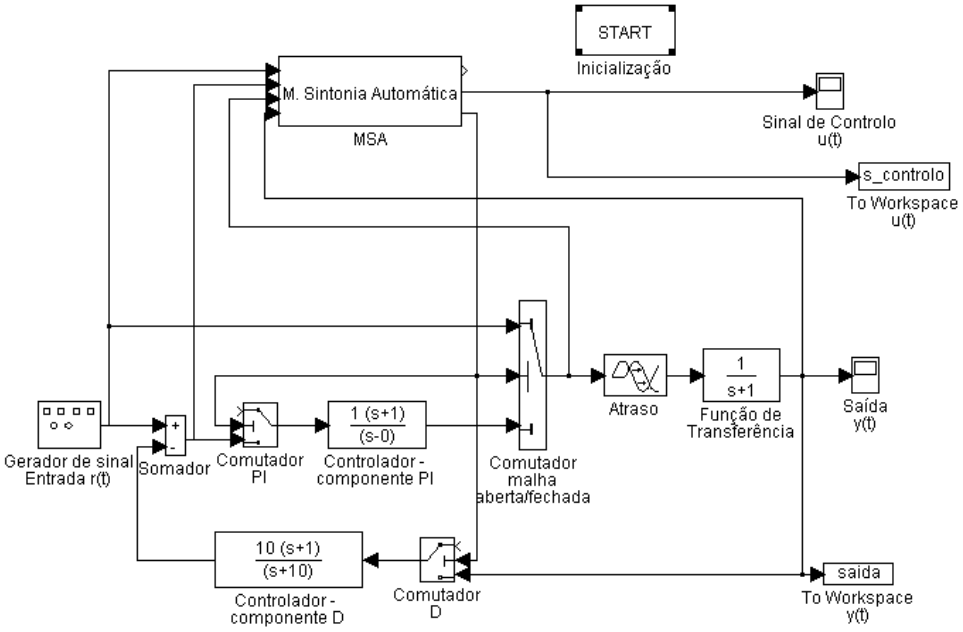


Fig. 17.1 - Modelo em SIMULINK que implementa o sistema final. (*sistfim*)

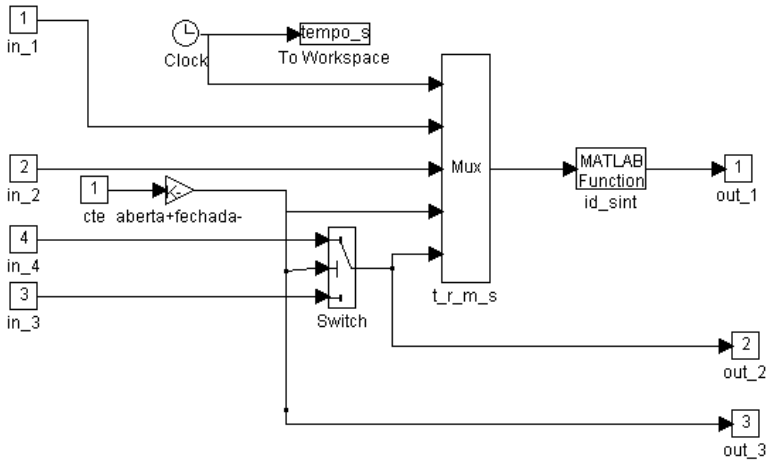


Fig. 17.2 - Modelo em SIMULINK que implementa o sistema final
- Bloco Módulo de Sintonia Automática

Este modelo é composto por dois módulos:

- Processo a controlar e controlador PID
- Inicialização, Sintonia e Adaptação

5.1.1 - **Módulo processo a controlar e controlador PID**

Esta parte do modelo encontra-se totalmente representada na figura 17.1 e inclui todos os blocos à excepção dos intitulados *Inicialização* e *MSA*.

Os blocos constituintes desta parte do modelo são:

- *Gerador de Sinal* , *Entrada r(t)*– gerador de onda quadrada de entrada.
- *Atraso* e *Função de Transferência* – permitem a definição do processo a controlar em termos de atraso, ganho multiplicativo e pólos.
- *Controlador – Componente PI* e *Controlador – Componente D* – implementam o controlador de acordo com as funções de transferência representadas nos blocos.
- *To Workspace u(t)* e *To Workspace y(t)* - permitem a obtenção dos sinais de controlo e de saída, na forma vectorial, depois de terminada a simulação.
- *Sinal de Controlo u(t)* e *Saída y(t)* – permitem a visualização das formas de onda dos sinais de controlo e de saída, no decorrer da simulação.
- *Comutador malha aberta/fechada, Comutador PI* e *Comutador D* – comandados pela parte de adaptação do modelo para implementar o sistema em malha aberta ou fechada. Se o comando for no sentido de a simulação se realizar em malha aberta, a posição de cada comutador é a representada na figura, ou seja, a entrada é fornecida ao bloco *Atraso* e os blocos do controlador estão desligados. Se o comando for no sentido de a simulação se realizar em malha fechada, o bloco *Controlador – componente PI* recebe o sinal do *Somador*, a sua saída é passada ao bloco *Atraso* e o bloco *Controlador – componente D* recebe o sinal de saída.

5.1.2 - **Módulo Inicialização, Sintonia e Adaptação**

Esta parte do modelo é composta pelos seguintes blocos da figura 17.1:

- *Inicialização* – define alguns parâmetros necessários à inicialização da simulação e é implementado pela função em MATLAB *inicia* (Apêndice B.19).
- *MSA* – responsável pela supervisão, sintonia e adaptação. É um sub-sistema que quando aberto é como se representa na figura 17.2.

Na figura 17.2, as entradas representadas por *in_1*, *in_2, in_3* e *in_4* recebem os sinais correspondentes às entradas do bloco *MSA* na figura 17.1. Com as saídas o processo é idêntico à excepção da saída *out_1* que não está ligada já que a função de MATLAB não devolve nenhum valor directamente.

O sub-sistema *MSA* é composto pelos seguintes blocos:

- *Malha aberta(+1) fechada(-1)* – bloco do tipo ganho que toma os valores +1 ou –1, correspondentes a malha aberta e fechada, respectivamente. Este bloco controla o *Comutador Sinal* e através da saída *out_3* vai também controlar o *Comutador malha aberta/fechada,* o *Comutador PI* e o *Comutador D,* do modelo da figura 17.1.
- *Comutador Sinal* – funciona de forma semelhante aos restantes comutadores, fazendo passar o sinal de saída, quando a simulação é em malha aberta, e o sinal de controlo, quando a simulação é em malha fechada.
- *To Workspace t* – permite a obtenção do tempo da simulação, na forma vectorial, depois de a simulação ter terminado.
- *t_r_m_s* – multiplexador que, a cada instante t_i da simulação, compõe um vector de cinco componentes, tempo, valor da entrada, erro, indicador de malha

aberta/fechada e sinal de controlo ou de saída, ou seja, forma o vector $(t_i, r(t_i), e(t_i), 1, y(t_i))$ ou $(t_i, r(t_i), e(t_i), -1, u(t_i))$, conforme o sistema esteja em malha aberta ou fechada, respectivamente.

- *id_sint* – função do MATLAB (Apêndice B. 20) que, em cada instante da simulação recebe um vector do bloco *t_r_m_s*, a partir do qual identifica o processo a controlar, sintoniza o controlador e produz algumas acções de comando e/ ou adaptação.

A função *id_sint* é representada esquematicamente na figura 18, onde em detalhe temos:

- *Validação do instante de simulação* – necessário porque o algoritmo de simulação do SIMULINK por vezes necessita de retroceder no tempo para garantir a tolerância permitida, fazendo com que a sequência de pontos que define a resposta corresponda a instantes no tempo que se sucedem duma forma monotonamente crescente. Sendo assim, só são considerados válidos os pontos gerados no instante t_q tais que, o ponto anterior é gerado para o instante t_p e $t_p < t_q$ e o ponto seguinte é gerado para o instante t_r e $t_r > t_q$.

- *Determinação da situação de instabilidade* –identifica, a cada instante de simulação, se o ponto da resposta corresponde a um máximo relativo e, para cada conjunto de três máximos relativos determina se estes são crescentes. Se tal acontecer, a malha é imediatamente aberta e aguarda-se pela próxima transição para iniciar uma identificação em malha aberta. Se a malha foi fechada na última alternância do sinal de entrada, a situação de instabilidade só é averiguada após o sinal de saída ultrapassar o dobro da amplitude do degrau. Esta tarefa é implementada pela função em MATLAB *maximor* (Apêndice B.21).

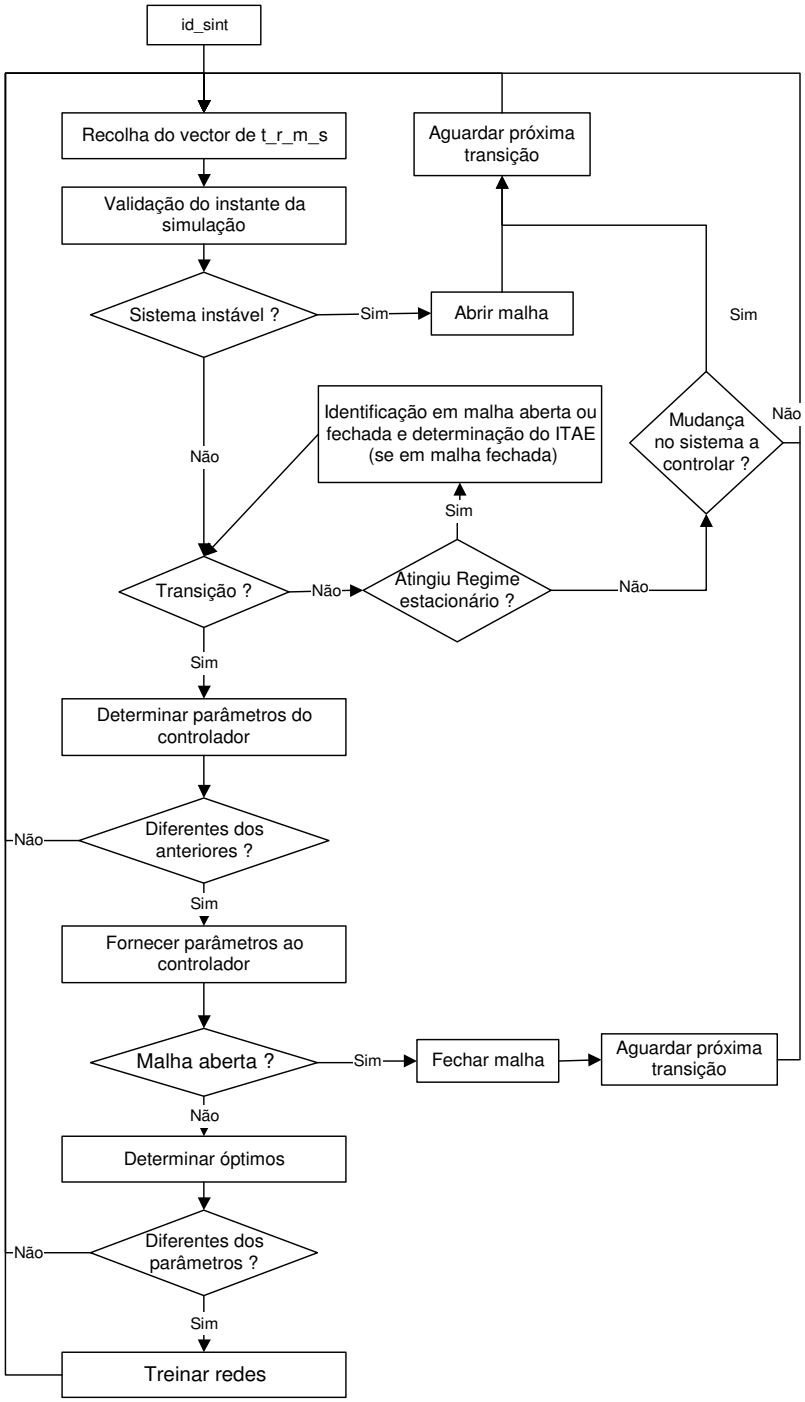


Fig.18 - Esquema lógico da função *id_sint*

- *Determinação da situação de regime estacionário* – para esta tarefa utiliza-se o critério denominado Regra de Avaliação Numérica do Valor Final [19], que consiste em considerar que uma resposta atingiu o seu valor final, num determinado intervalo de tempo, se não exibir variações percentuais superiores a

um certo valor α (neste caso $\alpha = 0.001$), nos últimos 10% da sua evolução nesse intervalo. Esta tarefa é implementada pela função em MATLAB *estacio* (Apêndice B.22).

- *Identificação em malha aberta ou fechada e determinação do ITAE* - após cada transição verificada na entrada, se se atingir o regime estacionário, calcula as medidas de identificação e o ITAE, caso esteja em malha fechada, usando a função em MATLAB *identif* (Apêndice B.23).
- *Detecção de mudança no processo a controlar* – realizada por observação da resposta do sistema após uma transição da entrada até à transição na entrada seguinte. Se durante este período de tempo houver uma variação na resposta superior a 20 % do valor médio das últimas três amostras da resposta, este intervalo não é considerado para identificação e aguarda-se a próxima transição para iniciar um novo processo de identificação. Esta tarefa é implementada pela função em MATLAB *mudac* (Apêndice B.24).
- *Determinar parâmetros do controlador* – determina os parâmetros do controlador, usando as redes de parâmetros do controlador, implementadas pela função em MATLAB *redes* (Apêndice B.25).
- *Determinar óptimos* – calcula os valores óptimos dos parâmetros do controlador PID, através da minimização do ITAE, usando a rede de modelização do ITAE, implementada pela função *rmitaem* (Apêndice B.26) e a função *fmins* da *Optimization Toolbox* [16]. Esta última função foi preferida à anteriormente utilizada devido ao facto, como se expõe em 4.1, de ela não poder ser usada sem uma observação e intervenção frequentes, sendo portanto inaceitável numa aplicação automática.

- *Treinar redes* - usando a medida de identificação, os valores óptimos obtidos, e o ITAE medido, durante uma só época de aprendizagem e com uma taxa de aprendizagem pequena, $\delta = 0.005$, para a rede de modelização e $\delta = 0.0005$ para as redes de parâmetros dos controladores, para não correr o risco de o MSE aumentar. O treino é realizado pela funções em MATLAB *aprendo* (Apêndice B.27) para as redes de parâmetros do controlador, e pela função *t_o_itae* (Apêndice B.18), para a rede de modelização do ITAE.

5.2 - **Simulação do sistema final**

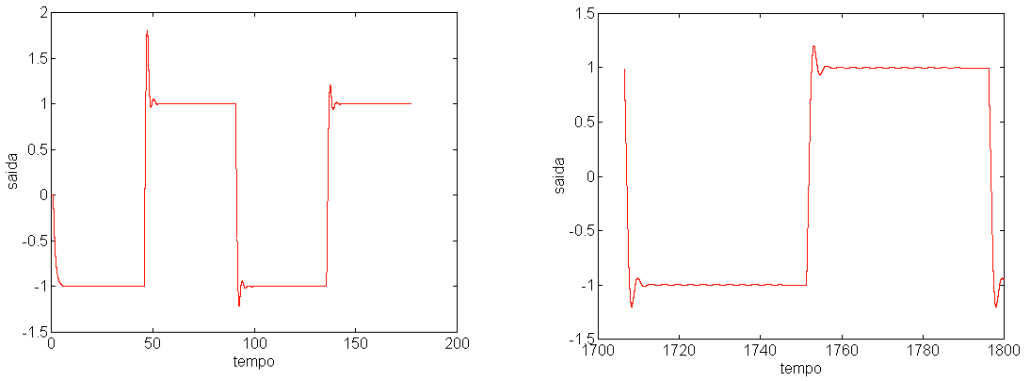
Para ilustrar o funcionamento do sistema final, seleccionou-se uma zona onde as redes de parâmetros dos controladores apresentavam, inicialmente, erros mais acentuados que, como se pode observar da figura 12, corresponde à zona de atrasos de $l = 0.9$ a $l = 1.1$. O objectivo não é mostrar melhoramentos imediatos, pois como se constactou no capítulo 4.3.2, qualquer melhoramento só se torna notório ao fim de algumas dezenas de épocas de treino. Escolheu-se esta zona, apenas para mostrar que o sistema não instabiliza facilmente.

Assim utilizou-se um processo com um atraso de $l = 1.0$, e simulou-se o sistema para vários ciclos de transição da entrada. A figura 19a) ilustra a saída para as 4 primeiras transições da referência, onde podem ser verificados:

- a resposta em malha aberta (1ª transição);
- o fecho da malha com os valores dos parâmetros PID obtidos por identificação em malha aberta (2ª transição);
- a resposta em malha fechada com os parâmetros do controlador ainda obtidos por identificação em malha aberta (3ª transição);

- a resposta em malha fechada com os parâmetros do controlador obtidos por identificação em malha fechada, resultado da análise da 3ª transição (4ª transição).

A figura 19b) representa a saída do sistema no final de 40 iterações. O valor do ITAE inicial era de 1.531, enquanto o valor do ITAE final é de 1.5281, o que corresponde a um erro, em relação ao valor óptimo, inicial de 33.1 % e um erro final de 32.88 %.

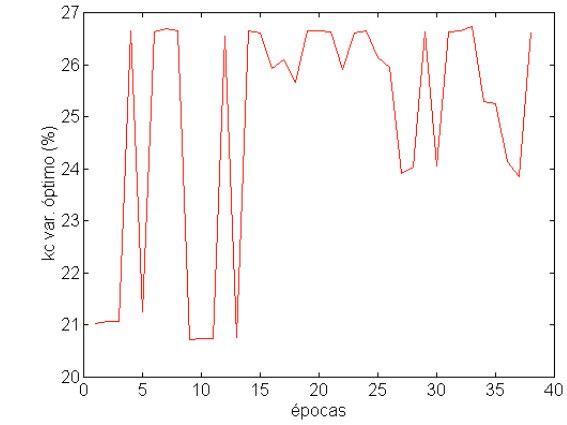


a) **b)**

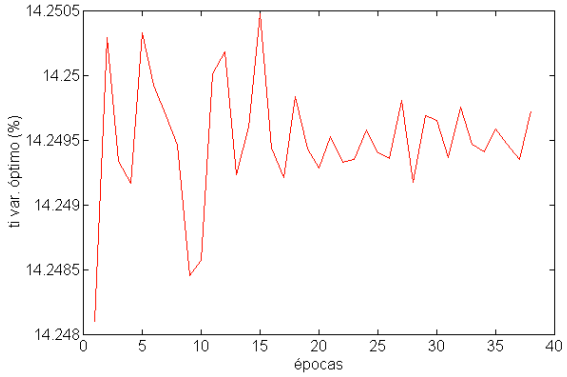
Fig.19 – Saída do sistema final, a) 1as quatro transições, b) ultimas duas transições

A fim de se poder verificar o resultado da adaptação *on-line*, nas figuras 20 a), b) e c) são representados os valores dos erros relativos dos parâmetros, obtidos por optimização utilizando a rede de ITAE.

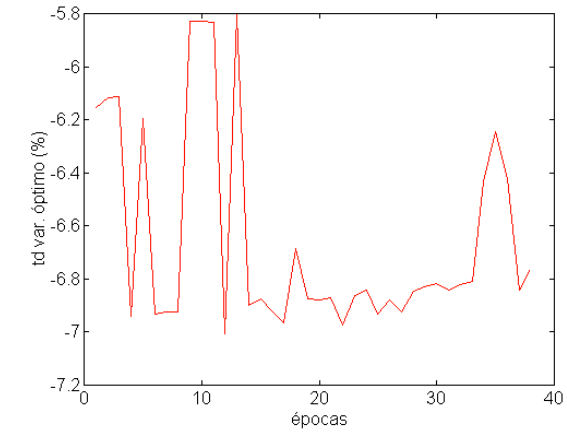
As figuras 21a), b) e c) ilustram a evolução dos erros relativos dos parâmetros do controlador, obtidos pelas redes dos parâmetros, e utilizados efectivamente no controlador.



a)



b)

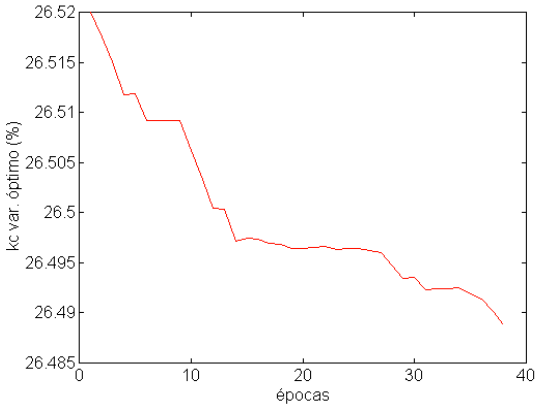


c)

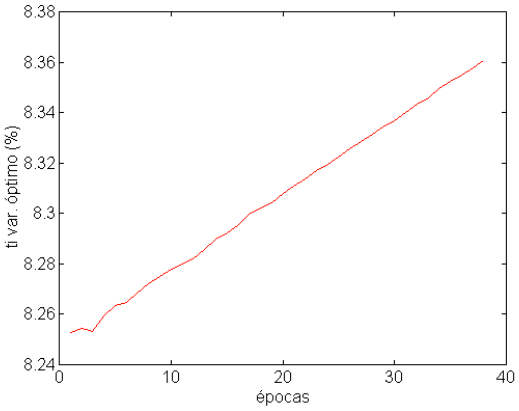
Fig. 20 - Sistema final -Evolução dos erros relativos dos parâmetros, obtidos por optimização utilizando a rede de ITAE, a) k_c, b) T_i e c) T_d

Comparando as duas figuras, observa-se que, embora a rede de modelização esteja a produzir resultados que oscilam, consegue efectivamente melhorar os resultados

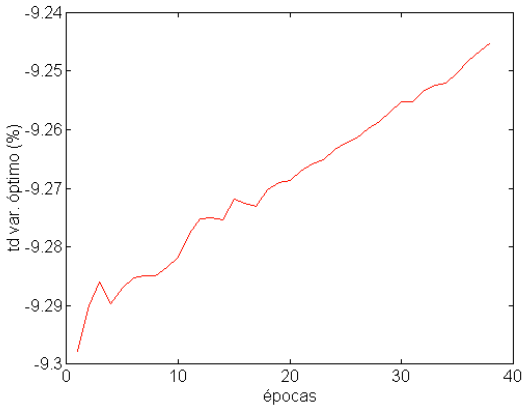
obtidos pelas redes dos parâmetros kc e Td. A rede do parâmetro Ti não consegue melhorar, devido ao problema relacionado com os pesos já descrito em 4.3.2.



a)



b)



c)

Fig. 21 - Sistema final - Evolução dos erros relativos dos parâmetros do controlador, obtidos pelas redes dos parâmetros a) kc, b) Ti e c) Td

Capítulo 6

CONCLUSÕES

O objectivo principal deste trabalho foi demonstrar a possibilidade de funcionamento e implementar um sistema de sintonia automática de controladores PID, segundo o critério ITAE, usando redes neuronais do tipo *B-spline*, treinadas inicialmente *off-line* e posteriormente adaptadas em tempo real.

No primeiro capítulo, enquadra-se o tema, entre os últimos desenvolvimentos na área, apresentam-se os principais componentes e meios a utilizar e expõe-se a estrutura de desenvolvimento dos assuntos.

A descrição das medidas de identificação do sistema a utilizar e do critério ITAE é apresentada no segundo capítulo, onde se salienta que, a escolha deste tipo de medidas de identificação se deve à vantagem de poderem ser usadas quer em malha aberta quer em malha fechada.

As redes *B-spline*, a regra de aprendizagem usada no seu treino e o algoritmo ASMOD são descritos no capítulo três. Neste capítulo, fundamenta-se e detalha-se a utilização do algoritmo ASMOD como um bom método, por um lado, para gerar redes com uma estrutura ajustada a uma posterior adaptação em tempo real e, por outro, para suplantar o problema conhecido como "maldição dimensional". Justifica-se a utilização da regra de aprendizagem NLMS por, para além de ser razoável na aprendizagem *off-line*, ser muito apropriada para a adaptação *on-line*.

Ao longo do quarto capítulo, descreve-se o processo de implementação e desenvolvimento das redes e apresentam-se resultados, bastante bons para as redes de parâmetros do controlador, e resultados promissores para a rede de modelização do

ITAE. Salienta-se a extrema dificuldade encontrada no desenvolvimento destas redes devido ao gasto computacional envolvido. Analisam-se e comparam-se várias estratégias de construção da rede de modelização do ITAE e propõe-se uma nova regra de treino das redes, que apresenta vantagens relativamente à NLMS, para o treino *off-line*. Como esta regra pode levar à produção de vectores de pesos mal condicionados, deverá ser melhor estudada a sua aplicação e compará-la analiticamente com a NLMS.

No quinto capítulo, apresenta-se um modelo que permite implementar a sistema final, com simulação do processo a controlar, do controlador PID e do módulo de Sintonia Automática. Apresentam-se alguns resultados que confirmam a possibilidade de utilização deste método de sintonia automática de controladores PID.

Deve ser realçado que a técnica de sintonia proposta nesta Dissertação é uma técnica totalmente nova, e, como tal, deve ser mais estudada, abrindo novos caminhos para investigação.

Assim, no tocante a problemas surgidos no decorrer da implementação, como a minimização do ITAE, sugeria que se desenvolvesse um algoritmo de optimização específico para essa tarefa.

A regra de aprendizagem proposta deveráser melhor estudada, para ultrapassar o problema de gerar vectores de pesos mal condicionados. Desta forma conseguir-se-ia determinar modelos mais bem condicionados, numericamente, que se traduziriam num melhor funcionamento do sistema global.

As regras para construção de modelos segundo o algoritmo ASMOD ou outro semelhante devem ser mais desenvolvidas, e estudadas as diferentes estratégias de desenvolvimento das redes. Devido ao elevado gasto computacional, devia também tentar-se a paralelização do algoritmo já que, como em cada passo de construção estão envolvidos vários modelos que são treinados independentemente uns dos outros, e só

quando estão todos prontos é que são comparados, presume-se uma fácil e eficiente paralelização.

Um outro problema que não foi devidamente analisado neste trabalho, por falta de tempo, relaciona-se com a determinação das taxas de aprendizagem utilizadas para o treino *on-line* das diversas redes neuronais utilizadas. Obviamente, e relacionado com este assunto, um estudo analítico da estabilidade desta técnica deveria ser realizado, embora se anteveja ser um problema de difícil resolução.

Finalmente, dado o interesse deste assunto, a nível industrial, considero que seria justificável desenvolver-se este sistema no sentido duma aplicação prática, devendo adicionalmente analizar-se a capacidade de extensão desta aplicação ao funcionamento como regulador.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Åström, Karl J. e Björn Wittenmark, *Computer-Controlled Systems, Theory and Design,* Prentice-Hall, E.U.A., 1990

[2] Ruano, A., P. Fleming e D. Jones, " A Connectionist Approach to PID Autotuning", IEE Proceedings, Parte D., Vol. 139, Nº 3, 1992, pp. 279-285

[3] Ruano, A., J. Lima, R. Mamat e P. Fleming, "Comparison of Alternative Approaches to Neural Network PID Autotuning", JSE, Vol. 6, Nº 3, 1996, pp. 166-176

[4] Brown, Martin e Chris Harris, *Neurofuzzy Adaptive Modelling and Control*, Prentice Hall, London, 1994

[5] Weyer, E. e T. Kavli, "The ASMOD algorithm. Some new theoretical and experimental results", SINTEF report STF31 A95024, Oslo, 1995

[6] Åström, Karl e T. Hagglünd, " A frequency domain method for automatic tuning of simple feedback loops", Proc. 23rd Conference On Decision and Control, 1984, pp. 299-304

[7] Åström, Karl e T. Hagglünd, *Automatic tuning PID controllers,* I.S.A. Research Triangle Park, 1988

[8] Nishikawa, Y., N. Sannomya, T. Ohta e H. Tanaka, "A method for auto-tuning of PID control parameters", Automatica, 20,3,1994, pp. 221-332

[9] Ljung, L., *System Identification: Theory for the user*, Prentice-Hall, 1987

[10] Åström, Karl J. e Björn Wittenmark, *Adaptive Control*, Addison-Wesley, 1989

[11] Wang, P. e D. Kwok, " Optimal design of PID controllers based on genetic algorithms, IFAC 12th World Congress, 5, 1993, pp. 261-265

[12] Shroder, P., B. Green, N. Grum e P.J. Fleming, " On-line Genetic Autotuning of PID controllers for an Active Bearing Application, 5th IFAC Workshop on Algorithms and Architectures for Real-Time Control (AARTC'98), Cancun, México, Abril 1998, pp. 243-245

[13] AA.VV., *MATLAB® User's Guide*, The MathWorks Inc., E.U.A., 1993

[14] AA.VV., *SIMULINK™ - Dynamic System Simulation Software – User's Guide*, The MathWorks Inc., E.U.A., 1993

[15] AA.VV., *Control System Toolbox User's Guide*, The MathWorks Inc., E.U.A., 1992

[16] Grace, Andrew, *Optimization Toolbox User's Guide*, The MathWorks Inc., E.U.A., 1992

[17] Palm, William J., *Control Systems Engineering*, John Wiley &Sons, Inc., E.U.A. 1986

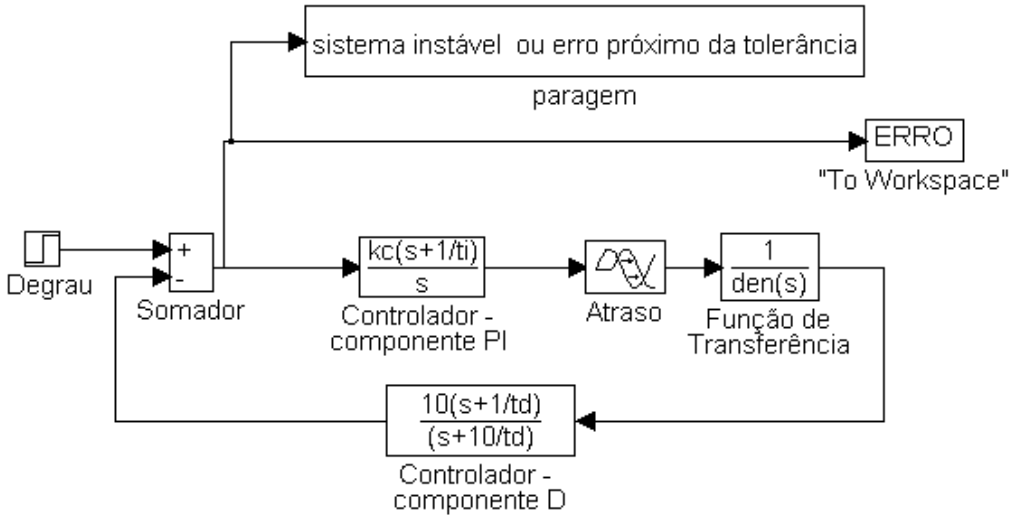
[18] Azevedo, A. B., Ruano, A. E. B., *B-splines Neural Networks PID Autotuning,* Int. Conf. On Engineering Applications of Neural Networks (EANN98), Gibraltar, Junho1998 (a aguardar publicação).

[19] Lima, J., "Modelo em Simulink para Sintonia Automática de Controladores PID usando rede neuronais", Unidade de Ciências Exactas e Humanas, Universidade do Algarve, 1995

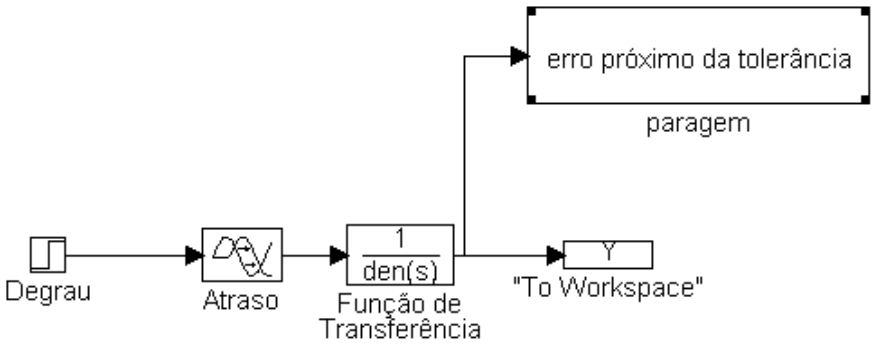
APÊNDICES

A – Modelos realizados em SIMULINK

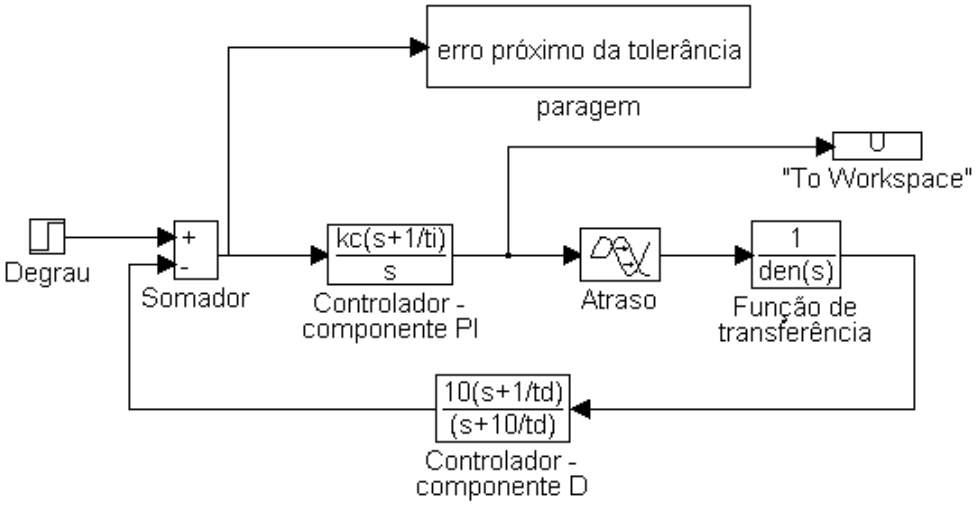
A.1 – sistema.m



A.2- sinid.m



A.3 – csist.m



A.4 – sistfim.m

```
function [ret,x0,str]=sistfim(t,x,u,flag);
%SISTFIM      is the M-file description of the SIMULINK system named SISTFIM.
%      The block-diagram can be displayed by typing: SISTFIM.
%
%      SYS=SISTFIM(T,X,U,FLAG) returns depending on FLAG certain
%      system values given time point, T, current state vector, X,
%      and input vector, U.
%      FLAG is used to indicate the type of output to be returned in SYS.
%
%      Setting FLAG=1 causes SISTFIM to return state derivatives, FLAG=2
%      discrete states, FLAG=3 system outputs and FLAG=4 next sample
%      time. For more information and other options see SFUNC.
%
%      Calling SISTFIM with a FLAG of zero:
%      [SIZES]=SISTFIM([],[],[],0),  returns a vector, SIZES, which
%      contains the sizes of the state vector and other parameters.
%            SIZES(1) number of states
%            SIZES(2) number of discrete states
%            SIZES(3) number of outputs
%            SIZES(4) number of inputs.
%      For the definition of other parameters in SIZES, see SFUNC.
%      See also, TRIM, LINMOD, LINSIM, EULER, RK23, RK45, ADAMS, GEAR.
```

```
% Note: This M-file is only used for saving graphical information;
%       after the model is loaded into memory an internal model
%       representation is used.
% the system will take on the name of this mfile:
sys = mfilename;
new_system(sys)
simver(1.2)
if(0 == (nargin + nargout))
     set_param(sys,'Location',[4,42,628,440])
     open_system(sys)
end;
set_param(sys,'algorithm',          'RK-45')
set_param(sys,'Start time',   '0.0')
set_param(sys,'Stop time',          '1000')
set_param(sys,'Min step size',      '0.000001')
set_param(sys,'Max step size',      '0.001')
set_param(sys,'Relative error','1e-12')
set_param(sys,'Return vars',  '')
add_block('built-in/Sum',[sys,'/','Somador'])
set_param([sys,'/','Somador'],'inputs','+-',...
            'position',[100,263,115,292])
add_block('built-in/Switch',[sys,'/',['Comutador',13,'PI']])
set_param([sys,'/',['Comutador',13,'PI']],...
            'position',[155,254,180,286])
add_block('built-in/Zero-Pole',[sys,'/',['Controlador -
',13,'componente PI ']])
set_param([sys,'/',['Controlador - ',13,'componente PI ']],...
            'Zeros','-1/ti',...
            'Poles','[0]',...
            'Gain','[kc]',...
            'position',[215,253,285,287])
add_block('built-in/Signal Generator',[sys,'/',['Gerador de
sinal',13,'Entrada r(t)']])
set_param([sys,'/',['Gerador de sinal',13,'Entrada r(t)']],...
            'Peak','1.000000',...
            'Peak Range','5.000000',...
            'Freq','0.100000',...
            'Freq Range','525.000000',...
            'Wave','Sqr',...
            'Units','Rads')
open_system([sys,'/',['Gerador de sinal',13,'Entrada r(t)']])
```

```
set_param([sys,'/',['Gerador de sinal',13,'Entrada r(t)']],...
            'position',[15,253,60,287])
add_block('built-in/Zero-Pole',[sys,'/',['Controlador -
',13,'componente D']])
set_param([sys,'/',['Controlador - ',13,'componente D']],...
            'orientation',2,...
            'Zeros','[-1/td]',...
            'Poles','[-10/td]',...
            'Gain','[10]',...
            'position',[140,345,245,385])
add_block('built-in/Switch',[sys,'/',['Comutador',13,'D']])
set_param([sys,'/',['Comutador',13,'D']],'orientation',2,...
            'position',[280,349,305,381])
add_block('built-in/To Workspace',[sys,'/',['To
Workspace',13,'y(t)']])
set_param([sys,'/',['To Workspace',13,'y(t)']],...
            'mat-name','saida',...
            'position',[565,367,615,383])
add_block('built-
in/Switch',[sys,'/',['Comutador',13,'malha',13,'aberta//fechada']])
set_param([sys,'/',['Comutador',13,'malha',13,'aberta//fechada']],...
            'position',[350,182,370,288])
add_block('built-in/Transport Delay',[sys,'/','Atraso'])
set_param([sys,'/','Atraso'],'Delay Time','l',...
            'position',[410,220,455,250])
add_block('built-in/Scope',[sys,'/',['Saída',13,'y(t)']])
set_param([sys,'/',['Saída',13,'y(t)']],...
            'Vgain','6.000000',...
            'Hgain','80.000000',...
            'Vmax','12.000000',...
            'Hmax','160.000000',...
            'Window',[100,100,488,480],...
            'position',[590,222,610,248])
add_block('built-in/Transfer Fcn',[sys,'/',['Função
de',13,'Transferência']])
set_param([sys,'/',['Função de',13,'Transferência']],...
            'Numerator','kp',...
            'Denominator','den',...
            'position',[480,216,535,254])
add_block('built-in/Scope',[sys,'/',['Sinal de Controlo',13,'u(t)']])
set_param([sys,'/',['Sinal de Controlo',13,'u(t)']],...
```

```
            'Vgain','6.000000',...
            'Hgain','80.000000',...
            'Vmax','12.000000',...
            'Hmax','160.000000',...
            'Window',[100,100,516,480],...
            'position',[575,12,595,38])

add_block('built-in/To Workspace',[sys,'/',['To
Workspace',13,'u(t)']])
set_param([sys,'/',['To Workspace',13,'u(t)']],...
            'mat-name','s_controlo',...
            'position',[565,77,615,93])
%     Subsystem  'Adaptacao'.
new_system([sys,'/','Adaptacao'])
set_param([sys,'/','Adaptacao'],'Location',[4,42,628,440])
open_system([sys,'/','Adaptacao'])
add_block('built-in/Outport',[sys,'/','Adaptacao/out_1'])
set_param([sys,'/','Adaptacao/out_1'],...
            'position',[470,120,490,140])
add_block('built-in/MATLAB Fcn',[sys,'/','Adaptacao/id_sint'])
set_param([sys,'/','Adaptacao/id_sint'],...
            'MATLAB Fcn','id_sint',...
            'position',[390,115,440,145])
add_block('built-in/Outport',[sys,'/','Adaptacao/out_3'])
set_param([sys,'/','Adaptacao/out_3'],...
            'Port','3',...
            'position',[475,285,495,305])
add_block('built-in/Outport',[sys,'/','Adaptacao/out_2'])
set_param([sys,'/','Adaptacao/out_2'],...
            'Port','2',...
            'position',[475,230,495,250])
add_block('built-in/Switch',[sys,'/','Adaptacao/Switch'])
set_param([sys,'/','Adaptacao/Switch'],...
            'position',[210,161,230,219])
add_block('built-in/To Workspace',[sys,'/',['Adaptacao/To
Workspace',13,'']])
set_param([sys,'/',['Adaptacao/To Workspace',13,'']],...
            'mat-name','tempo',...
            'position',[195,12,245,28])
add_block('built-in/Clock',[sys,'/','Adaptacao/Clock'])
set_param([sys,'/','Adaptacao/Clock'],...
```

```
            'position',[105,10,125,30])
add_block('built-in/Mux',[sys,'/','Adaptacao/t_r_m_s'])
set_param([sys,'/','Adaptacao/t_r_m_s'],...
            'position',[330,49,365,211])
add_block('built-in/Inport',[sys,'/','Adaptacao/in_2'])
set_param([sys,'/','Adaptacao/in_2'],...
            'Port','2',...
            'position',[40,225,60,245])
add_block('built-in/Inport',[sys,'/','Adaptacao/in_3'])
set_param([sys,'/','Adaptacao/in_3'],...
            'Port','3',...
            'position',[40,160,60,180])
add_block('built-in/Inport',[sys,'/','Adaptacao/in_1'])
set_param([sys,'/','Adaptacao/in_1'],...
            'position',[40,100,60,120])
add_block('built-in/Constant',[sys,'/','Adaptacao/cte'])
set_param([sys,'/','Adaptacao/cte'],...
            'position',[50,180,70,200])
add_block('built-in/Gain',[sys,'/','Adaptacao/aberta+fechada-'])
set_param([sys,'/','Adaptacao/aberta+fechada-'],...
      'Gain','-1',...
      'position',[100,180,120,200])
add_line([sys,'/','Adaptacao'],[235,190;320,190])
add_line([sys,'/','Adaptacao'],[370,130;380,130])
add_line([sys,'/','Adaptacao'],[130,20;130,70;320,70])
add_line([sys,'/','Adaptacao'],[130,20;185,20])
add_line([sys,'/','Adaptacao'],[65,110;320,110])
add_line([sys,'/','Adaptacao'],[445,130;460,130])
add_line([sys,'/','Adaptacao'],[145,190;155,190;155,150;320,150])
add_line([sys,'/','Adaptacao'],[155,190;155,295;465,295])
add_line([sys,'/','Adaptacao'],[285,190;285,240;465,240])
add_line([sys,'/','Adaptacao'],[65,170;200,170])
add_line([sys,'/','Adaptacao'],[65,235;170,235;170,210;200,210])
add_line([sys,'/','Adaptacao'],[95,190;110,190])
add_line([sys,'/','Adaptacao'],[155,190;200,190])
set_param([sys,'/','Adaptacao'],...
            'Mask Display','Módulo de Sintonia')
%     Finished composite block 'Adaptacao'.
set_param([sys,'/','Adaptacao'],'position',[170,46,300,124])
%     Subsystem  'Inicialização'.
new_system([sys,'/','Inicialização'])
```

```
set_param([sys,'/','Inicialização'],'Location',[-10,380,117,533])
set_param([sys,'/','Inicialização'],...
            'Mask Display','START',...
            'Mask Type','',...
            'Mask Dialogue','eval(''inicia'')',...
            'Mask Translate','',...
            'Mask Help','')
%     Finished composite block 'Inicialização'.
set_param([sys,'/','Inicialização'],'position',[375,29,446,65])
add_line(sys,[65,270;90,270])
add_line(sys,[460,235;470,235])
add_line(sys,[135,365;80,365;80,285;90,285])
add_line(sys,[120,280;145,280])
add_line(sys,[185,270;205,270])
add_line(sys,[290,270;340,270])
add_line(sys,[375,235;400,235])
add_line(sys,[540,235;550,235;550,375;315,375])
add_line(sys,[275,365;255,365])
add_line(sys,[540,235;580,235])
add_line(sys,[550,375;555,375])
add_line(sys,[65,270;65,60;160,60])
add_line(sys,[540,235;550,235;550,140;160,140;160,110])
add_line(sys,[305,110;320,110;320,235;340,235])
add_line(sys,[320,235;320,365;315,365])
add_line(sys,[320,235;135,235;135,270;145,270])
add_line(sys,[65,200;340,200])
add_line(sys,[385,235;385,165;145,165;145,85;160,85])
add_line(sys,[305,85;555,85])
add_line(sys,[515,85;515,25;565,25])
% Return any arguments.
if (nargin | nargout)
      % Must use feval here to access system in memory
      if (nargin > 3)
            if (flag == 0)
                  eval(['[ret,x0,str]=',sys,'(t,x,u,flag);'])
            else
                  eval(['ret =', sys,'(t,x,u,flag);'])
            end
      else
            [ret,x0,str] = feval(sys);
      end
```

```
end
```

B – Funções do MATLAB

B.1 – *itaef.m*

```
function f=itaef(x,tf,options)
%
%     Cálculo do ITAE
%
%     itaef(x,tf,options), se não se indicar options
%     options=[ 10^(-12) 0.000001 1 0 0 2]
global kc ti td ERRO TEMPO
if nargin<3, options=[10^(-12) 0.000001 1 0 0 2]; end
if nargin==1, tf=10000; end
kc=x(1); ti=x(2); td=x(3);
a=['kc= ' num2str(kc) ' ti= ' num2str(ti) ' td= ' num2str(td)];
disp(a)
erro=1; stop=0; fa=0;
TEMPO=rk45('sistema',tf,[],options);
f=0;
e=abs(ERRO); t=TEMPO;m=size(t,1); erro=ERRO(size(ERRO,1));
i=1;
if abs(erro)>1
      f=1/eps; stop=1;
      disp('O integral tende para infinito!!!')
end
while (i<=m-1) & stop==0
      dt=t(i+1)-t(i);
      tt=(t(i+1)+t(i))/2;
      ee=(e(i+1)+e(i))/2;
      f=f+tt*ee*dt;
      if f==NaN|f==Inf
            f=1/eps; stop=1;
            disp('O integral tende para infinito!!!')
      end
      i=i+1;
end
a=['itae= ' num2str(f) ' tf= ' num2str(t(i))];
disp(a)
```

B.2 – *initaezn*

```
function [kc,ti,td]=initaezn
%
%     Determina os parâmetros do PID
%     usando o critério de Ziegler Nichols
%
[t,yy]=rk45('sinid',10);
y=yy(:,size(yy,2));
[yy,i]=max(diff(y));
m=(y(i+1)-y(i))/(t(i+1)-t(i));
b=y(i)-m*t(i); l=-b/m; z=m*t+b;
ymax=y(size(y,1));
[zz,j]=min(abs(ymax-z));
r=m;
kc=1.2*r*l; ti=2*l; td=l/2;
```

B.3 – *fsigmaop.m*

```
function [f,kp,tt]=fsigmaop(sigma)
%
%     Cálculo de f(sigma) em malha aberta
%
global TEMPO Y B
options=[ 10^(-12) 0.00000001 1 0 0 2]; tf=10000;
TEMPO=rk45('sinid',tf,[],options);
f=zeros(size(sigma,2),1);
yfim=Y(length(Y)); t=TEMPO; m=size(t,1); s0=0; i=1;
while i<m-1
      dt=t(i+1)-t(i); yy=(Y(i+1)+Y(i))/2;
      s0=s0+(yfim-yy)*dt;
      i=i+1;
end
kp=yfim/B; tt=s0/yfim;
for j=1:length(sigma)
      ssigma=0;
```

```
      for i=1:length(t)-1
            tau=(t(i+1)+t(i))/2; dtau=t(i+1)-t(i);
            yy=(Y(i+1)+Y(i))/2;
            ssigma=ssigma+exp(-tau*sigma(j)/tt)*(yfim-yy)*dtau;
      end
      f(j)=1-sigma(j)*ssigma/s0;
end
```

B.4 – *fsigmacp.m*

```
function [f,kp,tt]=fsigmacp(sigma,x)
%
%     Cálculo do f(sigma) em malha fechada
%
global kc ti td U TEMPO

options=[ 10^(-12) 0.00000001 1 0 0 2];
tf=1000;
kc=x(1); ti=x(2); td=x(3);
TEMPO=rk45('csist',tf,[],options);
f=zeros(size(sigma,2),1); ufim=U(length(U)); t=TEMPO; m=size(t,1);
su0=0; i=1;
while i<m-1
      dt=t(i+1)-t(i); uu=(U(i+1)+U(i))/2;
      su0=su0+(ufim-uu)*dt; i=i+1;
end
kp=B/ufim;
tt=ti/(kp*kc)+.9*td-su0/ufim;
for j=1:length(sigma)
      susigma=0;
      for i=1:length(t)-1
            tau=(t(i+1)+t(i))/2; dtau=t(i+1)-t(i);
            uu=(U(i+1)+U(i))/2;
            susigma=susigma+exp(-tau*sigma(j)/tt)*(ufim-uu)*dtau;
      end
      frac=(1+sigma(j)*td/(10*tt))/(1+sigma(j)*td/tt);
```

```
      f(j)=frac*(tt/(tt-sigma(j)*kp*susigma)-
sigma(j)*ti/(kc*kp*(tt+sigma(j)*ti)));
end
```

B.5 – *bspasmod.m*

```
%
%                 Programa que implementa uma rede B-spline
%              (de estrutura adaptativa usando o algoritmo ASMOD)
%        com um espaço de entrada de dimensão 1
%
help bspasmod
rede=menu('A rede a treinar é:','1/(kc*kp)', 'ti/tt' , 'td/tt')
posicao=input('Os valores de treino são (nº de ordem): ');
load optimos
X=fsigma(posicao)';
if rede==1, X(2,:)=1./kc(posicao)', end
if rede==2, X(2,:)=ti(posicao)'./tt(posicao)', end
if rede==3, X(2,:)=td(posicao)'./tt(posicao)', end
xmin=input('O valor mínimo da medida de identificação que a rede pode
tratar é:');
xmax=input('E o máximo é:');
ord=[];
while size(ord,1)~=1, ord=input('A ordem é (~=1) : ');end
%
%     Inicialização dos vectores de nós
%
m=size(X,2);            % m=nº de pares de treino
KNOTS1=iniknot(xmin,xmax,ord,m)
%
%     Determinação da dimensão de W inicial
%
s=size(KNOTS1,2)-2*ord+2; w=zeros(1,s); w1=w;
refstep=0; stopref=0; refindex=1;
mod=0; maisum=0; gp=1; ii=0; bayk=0; baykold=0;
kmodelo=zeros(1,20); wmodelo=zeros(1,20);
%
%     ASMOD
%
while stopref==0
```

```
      if gp==1, disp('Growing'), else, disp('Pruning'), end
      refstep=refstep+1
      while mod<=length(w1)-1
            mod=mod+1, k=KNOTS1;
%
%     Growing/Pruning
%
            if mod~=1
                  w=w1;
                  if gp==1
                        [k,w]=instknot(k,w,ord,mod-1);
                        else
                        [k,w]=delknot(k,w,ord,mod-1);
                        end
            end
%
%     Treino usando "NLMS error correction rule"
%
            epocas=0; stopt=0; eta=1;
            patamarer=0; er=[10 20 30 40]; id=0;
            while stopt==0;
                  se=0; s=size(k,2)-2*ord+2;
                  for q=1:m
                        a=ambf(X(1,q),k,ord);
                              o=a*w'; y=X(2,q); e=y-o;
                        w=aprendn(eta,w,a,y,e);
                  end
                  for q=1:m
                        a=ambf(X(1,q),k,ord);
                        o=a*w'; y=X(2,q); e=y-o;
                        se=se+.5*(e^2);
                        ofinal(mod,q)=o;
                  end
                  for p=4:-1:2, er(p)=er(p-1); end
                  er(1)=se;
                  if er(1)==0
                        se=realmin; stopt=1;
                  elseif condi(er)==1
                        eta=eta/2; id=id+1;
                        patamarer(id)=se;
```

```
                        if (id>2 & er(1)>0)
                              if er(1)>=1
if (log10(patamarer(id))>=.9999*log10(patamarer(id-1))...
& log10(patamarer(id-1))>=.9999*log10(patamarer(id-2)))
                                    stopt=1;end
                        else
if (abs(log10(patamarer(id)))<=abs(1.0001*log10(patamarer(id-1)))...
& abs(log10(patamarer(id-1)))<=abs(1.0001*log10(patamarer(id-2))))
                                    stopt=1;end
                              end
                        end
                  end
                  epocas=epocas+1;
            end
            bayk(mod)=m*log(se/m)+s*log(m);
            erroq(mod)=er(1); stopt=0;
            for i=1:length(k), kmodelo(mod,i)=k(i); end
            for i=1:length(w), wmodelo(mod,i)=w(i); end
            a=['A aprendizagem decorreu ao longo de ' num2str(epocas) '
epocas.'];
            disp(a)
            a=['O erro médio quadrático final no conjunto de treino é '
num2str(er(1)/m) '.'];
            disp(a)
      end
      bayk, [b,i]=min(bayk);
      if (maisum>=1 & b<baykold), maisum=0, ii=0; end
      if i==1, ii=ii+1; else, ii=0; end
      KNOTS1=kmodelo(i,find(kmodelo(i,:)));
      w1=wmodelo(i,find(wmodelo(i,:)));
      a=['A melhor hipótese foi o modelo n° '  num2str(i)'.'];
      disp(a)
      bayka=bayk(i);
      if i~=1, ofinal(1,:)=ofinal(i,:); end
      of=ofinal(i,:);
            if (refstep>=5 & (rem(refindex,5)==0|ii==1))
                  if gp==1, gp=2; else, gp=1; end
                  refindex=0;
            elseif ii==2
                  if (er(1)~=0 & maisum<=1)
                        baykold=bayka;
```

```
                        [bb,j]=min(bayk(2:mod));
                        kold=KNOTS1; wold=w1;
                        KNOTS1=kmodelo(j+1,find(kmodelo(j+1,:)));
                        w1=wmodelo(j+1,find(wmodelo(j+1,:)));
                        maisum=maisum+1, gp=1; refindex=0;
                  else
                        stopref=1;
                  end
            elseif (refstep>=6 & ii==3)
                  KNOTS1=kold; w1=wold; stopref=1;
            end
            bayk=bayk(i); mod=1; refindex=refindex+1;
            wmodelo=w1; kmodelo=KNOTS1; KNOTS1, w1
end
of, X(2,:)
% Figura comparando os resultados obtidos e esperados
if rede==1, d=1./kc; end
if rede==2, d=ti./tt; end
if rede==3, d=td./tt; end
      for q=1:length(d)
            a=ambf(fsigma(q),KNOTS1,ord);
            o=a*w1'; ot(q)=o;
      end
figure
plot(fsigma,ot,'b-.',fsigma,d,'r-')
if rede==1, kk=KNOTS1; wk=w1;save asmpc kk wk,  end
if rede==2, ki=KNOTS1; wi=w1;save asmpi ki wi,  end
if rede==3, kd=KNOTS1; wd=w1;save asmpd kd wd,  end
```

B.6 – *iniknot.m*

```
function knot=iniknot(xmin,xmax,ord,m)
%
%     Inicializa os vectores dos nós
%
teixo=xmax-xmin; separacao=teixo/m; k=ord;
for j=1:k, knot(j)=xmin-(k-j)*separacao; end
knot(k+1)=(xmin+xmax)/2;
for j=k+2:2*k+1, knot(j)=xmax+(j-k-2)*separacao; end
```

B.7 – *instknot.m*

```
function [k,w]=instknot(k,w,ord,j);
%
%     Insere nós no meio do vector de nós existente
%     e duplica o valor de w respectivo
%
kins=(k(j+ord-1)+k(j+ord))/2;
for i=length(k):-1:j+ord, k(i+1)=k(i); end
k(j+ord)=kins;
for i=length(w):-1:j, w(i+1)=w(i); end
```

B.8 – *delknot.m*

```
function [k,w]=delknot(k1,w1,ord,j);
%
%     Apaga um nó no meio do vector de nós existente
%     e interpola o valor de w respectivo
%
k=[];w=w1(1:j);
k(1:j+ord-1)=k1(1:j+ord-1);
k(j+ord:length(k1)-1)=k1(j+ord+1:length(k1));
w(j)=(w1(j)+w1(j+1))/2;
w(j+1:length(w1)-1)=w1(j+2:length(w1));
```

B.9 – *ambf.m*

```
function a=ambf(x,k,ord);

%
%     Calculo dos valores dos ubf
%
a=zeros(1,length(k)-2*ord+2);
m=size(k,2);

ad=aad(x,k);

if ((ad+ord-1)>(m-ord+1)), kk=m-ord+1; else, kk=ad+ord-1; end

for m=ad:kk
      if ord==1
```

```
            a(m-ord+1)=ubf1(x,m,k);
      else
            a(m-ord+1)=ubf2(x,m,ord,k);
      end
end
```

B.10 – *aad.m*

```
function adress=aad(xj,knots)
%     Indica-nos o intervalo a que pertence xj num vector de nós
%     simples
adress=0;
m=size(knots,2);
if (xj<min(knots)|xj>max(knots))
      a=['A entrada x=' num2str(xj) ' não pertence ao espaço de entrada.'];
      disp(a)
      adress=NaN;
elseif xj==knots(m)
      adress=m-1;
else
      for j=1:m-1
            if (knots(j)<=xj & xj<knots(j+1))
                  adress=j;
            end
      end
end
```

B.11 – *ubf1.m*

```
function a=ubf1(x,j,knots)
%
%     Calcula a função-base univariável de 1ª ordem
%
if (x>=knots(1,j) & x<knots(1,j+1))
      a=1;
elseif j==(size(knots,2)-1)  % último intervalo do eixo
      if x==knots(1,j+1), a=1; else, a=0;      end
else
      a=0;
end
```

B.12 – *ubf2.m*

```
function a=ubf2(x,j,o,k)
%
%     calcula funções-base univariáveis de 2ª ordem e superior
if o~=2
      a1=((x-k(j-o+1))/(k(j)-k(j-o+1)))*ubf2(x,j-1,o-1,k);
      a2=((-x+k(j+1))/(k(j+1)-k(j-o+2)))*ubf2(x,j,o-1,k);
      o=o-1;
else
      a1=((x-k(j-o+1))/(k(j)-k(j-o+1)))*ubf1(x,j-1,k);
      a2=((-x+k(j+1))/(k(j+1)-k(j-o+2)))*ubf1(x,j,k);
end
a=a1+a2;
```

B.13 – *aprendn.m*

```
function w=aprendn(eta,w,a,y,e);
%
%     Inicializa e actualiza a matriz de pesos
%     pela regra "NLMS error correction rule"
%
val=find(a); m=size(val,2);
modmbf=a*a';
for i=1:m
      if w(val(i))==0
            w(val(i))=y;
      else
            w(val(i))=w(val(i))+eta*e*a(val(i))/modmbf;
      end
end
```

B.14 – *condi.m*

```
function c=condi(er)
%
```

```
%     indica se o erro está: c=0 - a decrescer;
%           c=1 - a crescer, estacionário (ou quase)
%
c=0;
if (er(1)==er(2) & er(1)==er(3))
      c=1;
elseif (er(1)==er(3) & er(2)==er(4))
      c=1;
elseif  (er(1)>er(2) & er(2)>er(3) & er(3)>er(4))
      c=1;
elseif (er(1)<er(2) & er(1)>=1 & log10(er(1))>0.999*log10(er(2)))...
& er(2)<er(3) & er(2)>0.999*log10(er(3)))
      c=1;
elseif (er(1)<er(2) & er(1)<1 &
abs(log10(er(1)))<abs(1.001*log10(er(2)))...
& er(2)<er(3) & abs(er(2))<abs(1.001*log10(er(3))))
      c=1;
end
```

B.15 – *bspitasu.m*

```
%
%                 Programa que implementa uma rede B-spline
%              (de estrutura adaptativa usando o algoritmo ASMOD)
%                      com um espaço de entrada de dimensão 4
%
help bspitasu
clear
load dmitae
nrede=input('Rede? - inicial=1; intermédia=2; ');
if nrede==1
      fsigmin=.3748; fsigmax=.5;
      kcmin=.1108; kcmax=4.69;
      timin=.22; timax=1.137;
      tdmin=0.028; tdmax=0.37;
end

ord=[]; troca=0;
while size(ord,2)~=4, ord=input('A ordem de cada é [figma kc ti td]: '); end
mw=15;
```

```
if nrede==1
%
%     Inicialização dos vectores de nós
%
KNOTSF=iniknot(fsigmin,fsigmax,ord(1),6,2)
KNOTSC=iniknot(kcmin,kcmax,ord(2),9,2)
KNOTSI=iniknot(timin,timax,ord(3),9,4)
KNOTSD=iniknot(tdmin,tdmax,ord(4),18,2)

%
%     Determinação de W's iniciais
%
      W1=sparse(zeros(1,mw)); W2=sparse(zeros(1,mw));
      W3=sparse(zeros(1,mw)); W4=sparse(zeros(1,mw));
      gp=1; pass=1; rede=4, multi=[1 1 2 0; 4 0 0 1]
else
      gp=input('growing(1)/pruning(2)?');
      rede=input('rede? - f(1), kc(2), ti(3), td(4)');
      pass=input('passagem?');
      mais=input('Carregar dados? (sim=1)');
      if mais==1, keyboard, end
end

k1i=KNOTSF; k2i=KNOTSC; k3i=KNOTSI; k4i=KNOTSD;
refstep=0; stopref=0; grava=0;
mod=0; ii=0; iii=0; bayk=0;

%
%     ASMOD
%

t0=clock;
t1=t0;
while stopref==0

      if gp==1, disp('Growing'), else disp('Pruning'), end
      refstep=refstep+1

            if rede==1, k=KNOTSF; elseif rede==2, k=KNOTSC; ...
            elseif rede==3, k=KNOTSI; else k=KNOTSD; end
```

```
      while mod<=length(k)-2*ord(rede)+gp-1
          mod=mod+1;
            if rede==1, k=KNOTSF; elseif rede==2, k=KNOTSC; ...
            elseif rede==3, k=KNOTSI; else k=KNOTSD; end
%
%     Growing/Pruning
%
          if mod~=1
            j=mod-1; ordem=ord(rede);
            if gp==1
                  kins=(k(j+ordem-1)+k(j+ordem))/2;
                  for i=length(k):-1:j+ordem
                        k(i+1)=k(i);
                  end
                  k(j+ordem)=kins;
            else
                  k1=[];
                  k1(1:j+ordem-1)=k(1:j+ordem-1);
                  k1(j+ordem:length(k)-1)=k(j+ordem+1:length(k));
                  k=k1;
            end
          end
          kf=KNOTSF; kc=KNOTSC; ki=KNOTSI; kd=KNOTSD;
          if rede==1, kf=k; elseif rede==2, kc=k; elseif rede==3, ki=k; ...
          else kd=k; end

%
%     Treino usando método instantâneo
%

[w1,w2,w3,w4,kf,kc,ki,kd,se,m]=treino4(itaede,rede,multi,mw...
,ord,kf,kc,ki,kd);

%
%     Determinação do bayk do modelo
%

s=length(find(w1))+length(find(w2))+length(find(w3))+length(find(w4))+...
length(kf)+length(kc)+length(ki)+length(kd);

          bayk(mod)=m*log(se/m)+s*log(m);
```

```
         stopt=0;

%
%     Colocação dos k e w do modelo numa matriz deste passo
%
         for i=1:length(kf), kmod_f(mod,i)=kf(i); end
         for i=1:length(kc), kmod_c(mod,i)=kc(i); end
         for i=1:length(ki), kmod_i(mod,i)=ki(i); end
         for i=1:length(kd), kmod_d(mod,i)=kd(i); end
         if any(w1~=0), w1mod(mod,:)=sparse(w1); else, w1mod=[]; end
         if any(w2~=0), w2mod(mod,:)=sparse(w2); else, w2mod=[]; end
         if any(w3~=0), w3mod(mod,:)=sparse(w3); else, w3mod=[]; end
         if any(w4~=0), w4mod(mod,:)=sparse(w4); else, w4mod=[]; end


         a=['O erro médio quadrático final no conjunto de treino é '
num2str(se/m) '.'];
         disp(a)
      end

%
%     Determinação do melhor modelo e reinicialização dos vectores
%     deste passo de refinamento
%
      mod=1;
      bayk
      [b,i]=min(bayk);

      if i==1
            if gp==1 & (troca==1|(length(k)-2*ord(rede))==2)
                  if ii==0, ii=2, else; ii=1, end
                  iii=iii+2
            else
                  ii=ii+1, iii=iii+1
            end
      else
            ii=0; iii=0;
      end
      troca=0;
```

```
      sf=find(kmod_f(i,:));
      sc=find(kmod_c(i,:));
      si=find(kmod_i(i,:));
      sd=find(kmod_d(i,:));
      KNOTSF=kmod_f(i,sf);
      KNOTSC=kmod_c(i,sc);
      KNOTSI=kmod_i(i,si);
      KNOTSD=kmod_d(i,sd);
      if length(w1mod)~=0, W1=sparse(w1mod(i,:)); else, W1=[]; end
      if length(w2mod)~=0, W2=sparse(w2mod(i,:)); else, W2=[]; end
      if length(w3mod)~=0, W3=sparse(w3mod(i,:)); else, W3=[]; end
      if length(w4mod)~=0, W4=sparse(w4mod(i,:)); else, W4=[]; end
      kmod_f=KNOTSF; w1mod=W1;
      kmod_c=KNOTSC; w2mod=W2;
      kmod_i=KNOTSI; w3mod=W3;
      kmod_d=KNOTSD; w4mod=W4;
      a=['A melhor hipótese foi o modelo nº '  num2str(i)'.'];
      disp(a)
%
%     Determinação do caminho a seguir no próximo passo
%
      if ii==0
         if gp==2 & (length(k)-2*ord(rede))==1
            gp=1, troca=1;
         elseif length(bayk)>=5
            if sign(min(bayk))==sign(max(bayk))
                  if sign(min(bayk))==1
                        if min(bayk)>.95*mean(bayk), if gp==1, gp=2; end, end
                  else
                        if min(bayk)>1.15*mean(bayk), if gp==1, gp=2; end, end
                  end
            end
         end
      else
         if ii==1 & (gp~=2 | (length(k)-2*ord(rede))~=1)
            if gp==1, gp=2; else gp=1; end
         elseif ii==2 | (gp==2 & (length(k)-2*ord(rede))==1)
            if iii<8
                  etime(clock,t1), t1=clock;
                  rede=rede+1, if rede==5, rede=1, end
                  gp=1; if ii==2, ii=0; end
```

```
            elseif pass==1
                  etime(clock,t1), t1=clock;
                  rede=4, multi=[1 1 2 0; 0 4 0 2], pass=2
                  baykq1=bayk(i), w1q1=W1, w2q1=W2, w3q1=W3
                  k1q1=KNOTSF, k2q1=KNOTSC, k3q1=KNOTSI, k4q1=KNOTSD
save 1x4m2m3 k1q1 k2q1 k3q1 k4q1 w1q1 w2q1 w3q1 baykq1
                  w1mod=[]; w2mod=[]; w3mod=[];
                  gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                  KNOTSF=k1i, KNOTSC=k2i, KNOTSI=k3i, KNOTSD=k4i
                  kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
            elseif pass==2
                  etime(clock,t1), t1=clock;
                  rede=3, multi=[1 1 2 0; 0 0 4 3], pass=3
                  baykq2=bayk(i), w1q2=W1, w2q2=W2, w3q2=W3
                  k1q2=KNOTSF, k2q2=KNOTSC, k3q2=KNOTSI, k4q2=KNOTSD
save 1m2x4m3 k1q2 k2q2 k3q2 k4q2 w1q2 w2q2 w3q2 baykq2
                  w1mod=[]; w2mod=[]; w3mod=[];
                  gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                  KNOTSF=k1i, KNOTSC=k2i, KNOTSI=k3i, KNOTSD=k4i
                  kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
            elseif pass==3
                  etime(clock,t1), t1=clock;
                  rede=2, multi=[1 1 2 0; 0 3 2 0], pass=4
                  baykq3=bayk(i), w1q3=W1, w2q3=W2, w3q3=W3
                  k1q3=KNOTSF, k2q3=KNOTSC, k3q3=KNOTSI, k4q3=KNOTSD
save 1m2m3x4 k1q3 k2q3 k3q3 k4q3 w1q3 w2q3 w3q3 baykq3
                  w1mod=[]; w2mod=[]; w3mod=[];
                  gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                  KNOTSF=k1i, KNOTSC=k2i, KNOTSI=k3i, KNOTSD=k4i
                  kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
            elseif pass==4
                  etime(clock,t1), t1=clock;
                  rede=2, multi=[1 1 2 0; 3 0 1 0], pass=5
                  baykq4=bayk(i), w1q4=W1, w2q4=W2, w3q4=W3
                  k1q4=KNOTSF, k2q4=KNOTSC, k3q4=KNOTSI, k4q4=KNOTSD
save 1m2x3m4 k1q4 k2q4 k3q4 k4q4 w1q4 w2q4 w3q4 baykq4
                  w1mod=[]; w2mod=[]; w3mod=[];
                  gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                  KNOTSF=k1i, KNOTSC=k2i, KNOTSI=k3i, KNOTSD=k4i
                  kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
            elseif pass==5
```

```
                etime(clock,t1), t1=clock;
                rede=1, multi=[1 1 2 0; 2 1 0 0], pass=6
                baykq5=bayk(i), w1q5=W1, w2q5=W2, w3q5=W3
                k1q5=KNOTSF, k2q5=KNOTSC, k3q5=KNOTSI, k4q5=KNOTSD
save 1x3m2m4 k1q5 k2q5 k3q5 k4q5 w1q5 w2q5 w3q5 baykq5
                w1mod=[]; w2mod=[]; w3mod=[];
                gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                KNOTSF=k1i, KNOTSC=k2i, KNOTSI=k3i, KNOTSD=k4i
                kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
          elseif pass==6
                etime(clock,t1), t1=clock; pass=7
                baykq6=bayk(i), w1q6=W1, w2q6=W2, w3q6=W3
                k1q6=KNOTSF, k2q6=KNOTSC, k3q6=KNOTSI, k4q6=KNOTSD
save 1x2m3m4 k1q6 k2q6 k3q6 k4q6 w1q6 w2q6 w3q6 baykq6
                [m,b]=min([baykq1 baykq2 baykq3 baykq4 baykq5 baykq6]);
                W4=0;
                if b==1
                      bayku1=baykq1, multi=[1 1 2 0;4 0 0 1]
                      KNOTSF=k1q1; KNOTSC=k2q1; KNOTSI=k3q1; KNOTSD=k4q1;

                      W1=w1q1, W2=w2q1, W3=w3q1
save 112 KNOTSF KNOTSC KNOTSI KNOTSD W1 W2 W3 W4 multi bayku1
                elseif b==2
                      bayku1=baykq2, multi=[1 1 2 0;0 4 0 2]
                      KNOTSF=k1q2; KNOTSC=k2q2; KNOTSI=k3q2; KNOTSD=k4q2;
                      W1=w1q2, W2=w2q2, W3=w3q2
save 112 KNOTSF KNOTSC KNOTSI KNOTSD W1 W2 W3 W4 multi bayku1
                elseif b==3
                      bayku1=baykq3, multi=[1 1 2 0; 0 0 4 3]
                      KNOTSF=k1q3; KNOTSC=k2q3; KNOTSI=k3q3; KNOTSD=k4q3;
                      W1=w1q3, W2=w2q3, W3=w3q3
save 112 KNOTSF KNOTSC KNOTSI KNOTSD W1 W2 W3 W4 multi bayku1
                elseif b==4
                      bayku1=baykq4, multi=[1 1 2 0; 0 3 2 0]
                      KNOTSF=k1q4; KNOTSC=k2q4; KNOTSI=k3q4; KNOTSD=k4q4;
                      W1=w1q4, W2=w2q4, W3=w3q4
save 112 KNOTSF KNOTSC KNOTSI KNOTSD W1 W2 W3 W4 multi bayku1
                elseif b==5
                      bayku1=baykq5, multi=[1 1 2 0; 3 0 1 0];
                      KNOTSF=k1q5; KNOTSC=k2q5; KNOTSI=k3q5; KNOTSD=k4q5;
                      W1=w1q5, W2=w2q5, W3=w3q5
```

```
save 112 KNOTSF KNOTSC KNOTSI KNOTSD W1 W2 W3 W4 multi bayku1
                else
                      bayku1=baykq6, multi,
save 112 KNOTSF KNOTSC KNOTSI KNOTSD W1 W2 W3 W4 multi bayku1
                end
                multi=[2 2 0 0; 4 3 2 1 ]; rede=1
                KNOTSF=k1q1, KNOTSC=k2q1, KNOTSI=k3q1, KNOTSD=k4q1
                w1mod=[]; w2mod=[]; w3mod=[];
                gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
          elseif pass==7
                etime(clock,t1), t1=clock;
                pass=8, rede=1, bayku2=bayk(i), w1u2=W1, w2u2=W2
                k1u2=KNOTSF, k2u2=KNOTSC, k3u2=KNOTSI, k4u2=KNOTSD
save 1x4m2x31 k1u2 k2u2 k3u2 k4u2 w1u2 w2u2 bayku2 multi
                multi=[2 2 0 0; 3 4 1 2];
                KNOTSF=k1q2, KNOTSC=k2q2, KNOTSI=k3q2, KNOTSD=k4q2
                w1mod=[]; w2mod=[]; w3mod=[];
                gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
          elseif pass==8
                etime(clock,t1), t1=clock;
                pass=9, rede=1, bayku3=bayk(i), w1u3=W1, w2u3=W2
                k1u3=KNOTSF, k2u3=KNOTSC, k3u3=KNOTSI, k4u3=KNOTSD
save 1x3m2x41 k1u3 k2u3 k3u3 k4u3 w1u3 w2u3 bayku3 multi
                multi=[2 2 0 0; 2 1 4 3];
                KNOTSF=k1q3, KNOTSC=k2q3, KNOTSI=k3q3, KNOTSD=k4q3
                w1mod=[]; w2mod=[]; w3mod=[];
                gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
          elseif pass==9
                etime(clock,t1), t1=clock;
                pass=10, rede=1, bayku4=bayk(i), w1u4=W1, w2u4=W2
                k1u4=KNOTSF, k2u4=KNOTSC, k3u4=KNOTSI, k4u4=KNOTSD
save 1x2m3x41 k1u4 k2u4 k3u4 k4u4 w1u4 w2u4 bayku4 multi
                multi=[2 2 0 0; 4 3 2 1 ]; rede=1
                KNOTSF=k1q4, KNOTSC=k2q4, KNOTSI=k3q4, KNOTSD=k4q4
                w1mod=[]; w2mod=[]; w3mod=[];
                gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
          elseif pass==10
```

```
                etime(clock,t1), t1=clock;
                pass=11, rede=1, bayku5=bayk(i), w1u5=W1, w2u5=W2
                k1u5=KNOTSF, k2u5=KNOTSC, k3u5=KNOTSI, k4u5=KNOTSD
save 1x4m2x32 k1u5 k2u5 k3u5 k4u5 w1u5 w2u5 bayku5 multi
                multi=[2 2 0 0; 3 4 1 2];
                KNOTSF=k1q5, KNOTSC=k2q5, KNOTSI=k3q5, KNOTSD=k4q5
                w1mod=[]; w2mod=[]; w3mod=[];
                gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
          elseif pass==11
                etime(clock,t1), t1=clock;
                pass=12, rede=1, bayku6=bayk(i), w1u6=W1, w2u6=W2
                k1u6=KNOTSF, k2u6=KNOTSC, k3u6=KNOTSI, k4u6=KNOTSD
save 1x3m2x42 k1u6 k2u6 k3u6 k4u6 w1u6 w2u6 bayku6 multi
                multi=[2 2 0 0; 2 1 4 3];
                KNOTSF=k1q6, KNOTSC=k2q6, KNOTSI=k3q6, KNOTSD=k4q6
                w1mod=[]; w2mod=[]; w3mod=[];
                gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
          else
                etime(clock,t1), t1=clock;
                pass=13, rede=1, bayku7=bayk(i), w1u7=W1, w2u7=W2
                k1u7=KNOTSF, k2u7=KNOTSC, k3u7=KNOTSI, k4u7=KNOTSD
save 1x2m3x42 k1u7 k2u7 k3u7 k4u7 w1u7 w2u7 bayku7 multi
                stopref=1
                w1mod=[]; w2mod=[]; w3mod=[];
                gp=1; refstep=0; mod=0; ii=0; bayk=0; iii=0;
                kmod_f=[]; kmod_c=[]; kmod_i=[]; kmod_d=[];
          end
        end
    end
    bayk=bayk(i); %%%%%%
%
%     Gravação em ficheiro dos valores obtidos
%
  if grava==1
    save ritae KNOTSF KNOTSC KNOTSI KNOTSD W1 W2 W3 W4 multi
    grava=0;
  end
end
etime(clock,t0)
```

B.16 – *treino4.m*

```
function [w1,w2,w3,w4,kf,kc,ki,kd,se,m]=treino4(itaede,rede,multi,mw,...
ord,kf,kc,ki,kd)
%
%     Treino usando método instantâneo
%

ep=0; se=0; lista1=[]; lista2=[]; lista3=[]; lista4=[];
w1=0; w2=0; w2=0; w3=0; w4=0;

m=810;
d=itaede(:,5);
ent1=itaede(:,1);
ent2=itaede(:,2);
ent3=itaede(:,3);
ent4=itaede(:,4);

for q=1:m
      a1=ambf(ent1(q),kf,ord(1));
      a2=ambf(ent2(q),kc,ord(2));
      a3=ambf(ent3(q),ki,ord(3));
      a4=ambf(ent4(q),kd,ord(4));
      y(q)=1/d(q);
      if any(multi(1,:)==1)
            if any(multi(1,:)==3)
                  if multi(2,1)==0
                        aa(q,:)=a1; ab=a2; ac=a3; ad=a4;
                  elseif multi(2,2)==0
                        aa(q,:)=a2; ab=a1; ac=a3; ad=a4;
                  elseif multi(2,3)==0
                        aa(q,:)=a3; ab=a1; ac=a2; ad=a4;
                  else
                        aa(q,:)=a4; ab=a1; ac=a2; ad=a3;
                  end
                  a(q,:)=form_a(3,ab,ac,ad,0,mw);
                  id=1;
            else
                  if multi(2,1)==0
```

```
                        aa(q,:)=a1;
                        if multi(2,2)==0
                              ab(q,:)=a2; ac=a3; ad=4;
                        elseif multi(2,3)==0
                              ab(q,:)=a3; ac=a2; ad=4;
                        else
                              ab(q,:)=a4; ac=a2; ad=3;
                        end
                  elseif multi(2,2)==0
                        aa(q,:)=a2;
                        if multi(2,3)==0
                              ab(q,:)=a3; ac=a1; ad=a4;
                        else
                              ab(q,:)=a4; ac=a1; ad=a2;
                        end
                  else
                        aa(q,:)=a3; ab(q,:)=a4; ac=a1; ad=a2;
                  end
                  a(q,:)=form_a(2,ac,ad,0,0,mw);
                  id=2;
            end
      else
                  aa=a1;
                  if multi(2,2)==1
                        ab=a2; ac=a3; ad=a4;
                  elseif multi(2,3)==1
                        ab=a3; ac=a2; ad=a4;
                  else
                        ab=a4; ac=a2; ad=a3;
                  end
                  aa1(q,:)=form_a(2,aa,ab,0,0,mw);
                  aa2(q,:)=form_a(2,ac,ad,0,0,mw);
                  id=3;
      end
end

if id==1
      [sa1l,sa1c]=size(aa);
      for i=1:sa1c
            size(find(aa(:,i)~=zeros(sa1l,1)));
            if ans~=[0 0], lista1=[lista1 i]; end
```

```
      end
      [sa2l,sa2c]=size(a);
      for i=1:sa2c
            size(find(a(:,i)~=zeros(sa2l,1)));
            if ans~=[0 0], lista2=[lista2 i]; end
      end
      ww=[aa(:,lista1) a(:,lista2)]\y';
      w1=sparse(zeros(1,mw)); w2=sparse(zeros(1,mw^3));
      leg1=length(lista1); leg2=length(lista2);
      for i=1:leg1, w1(lista1(i))=ww(i); end
      for i=leg1+1:length(ww), w2(lista2(i-leg1))=ww(i); end
      w1=sparse(w1); w2=sparse(w2);
      o=aa*w1(1:sa1c)+a*w2(1:sa2c);
      e=y'-o;
      se=0.5*sum(e.^2);
      %%%%% erro em percentagem do valor desejado
      ep=100*(d-1./o)./d;
elseif id==2
      [sa1l,sa1c]=size(aa);
      for i=1:sa1c
            size(find(aa(:,i)~=zeros(sa1l,1)));
            if ans~=[0 0], lista1=[lista1 i]; end
      end
      [sa2l,sa2c]=size(ab);
      for i=1:sa2c
            size(find(ab(:,i)~=zeros(sa2l,1)));
            if ans~=[0 0], lista2=[lista2 i]; end
      end
      [sa3l,sa3c]=size(a);
      for i=1:sa3c
            size(find(a(:,i)~=zeros(sa3l,1)));
            if ans~=[0 0], lista3=[lista3 i]; end
      end
      w1=sparse(zeros(1,mw)); w2=sparse(zeros(1,mw));
      w3=sparse(zeros(1,mw^2));
      A=[aa(:,lista1) ab(:,lista2) a(:,lista3)];
      ww=A\y';
      leg1=length(lista1); leg2=length(lista2); leg3=length(lista3);
      for i=1:leg1, w1(lista1(i))=ww(i); end
      for i=leg1+1:leg1+leg2, w2(lista2(i-leg1))=ww(i); end
      for i=leg1+leg2+1:length(ww), w3(lista3(i-leg1-leg2))=ww(i); end
```

```
      w1=sparse(w1); w2=sparse(w2); w3=sparse(w3);
      o=aa*w1(1:sa1c)+ab*w2(1:sa2c)+a*w3(1:sa3c);
      e=y'-o;
      se=0.5*sum(e.^2);
      %%%%% erro em percentagem do valor desejado
      ep=100*(d-1./o)./d;
else
      [sa1l,sa1c]=size(aa1);
      for i=1:sa1c
            size(find(aa1(:,i)~=zeros(sa1l,1)));
            if ans~=[0 0], lista1=[lista1 i]; end
      end
      [sa2l,sa2c]=size(aa2);
      for i=1:sa2c
            size(find(aa2(:,i)~=zeros(sa2l,1)));
            if ans~=[0 0], lista2=[lista2 i]; end
      end
      w1=sparse(zeros(1,mw^2)); w2=sparse(zeros(1,mw^2));
      ww=[aa1(:,lista1) aa2(:,lista2)]\y';
      for i=1:length(lista1), w1(lista1(i))=ww(i); end
      for i=length(lista1)+1:length(ww), w2(lista2(i-
length(lista1)))=ww(i); end
      w1=sparse(w1); w2=sparse(w2);
      o=aa1*w1(1:sa1c)+aa2*w2(1:sa2c);
      e=y'-o;
      se=0.5*sum(e.^2);
      %%%%% erro em percentagem do valor desejado
      ep=100*(d-1./o)./d;
end

if se==0, se=realmin;   end

a=['O erro máx em % (valores directos) é ' num2str(max(abs(ep))) '%.'];
disp(a)
a=['O erro médio em % (valores directos) é ' num2str(mean(abs(ep))) '%.'];
disp(a)
```

B.17 – *form_a.m*

```
function a=form_a(num,a1,a2,a3,a4,mw)
```

```
%
%     Produz o valor da mbf formada pelas ubf a1,a2,a3,a4
%

if length(a1)<mw, a1(length(a1)+1:mw)=zeros(1,mw-length(a1)); end
if length(a2)<mw, a2(length(a2)+1:mw)=zeros(1,mw-length(a2)); end
if num==2
      for i=1:mw
            for j=1:mw, a((i-1)*mw+j)=a1(i)*a2(j); end
      end

elseif any(a3~=0)
      if length(a3)<mw, a3(length(a3)+1:mw)=zeros(1,mw-length(a3)); end

      for i=1:mw
            for j=1:mw
                  for l=1:mw
                        a(((i-1)*mw+j-1)*mw+l)=a1(i)*a2(j)*a3(l);
                  end
            end
      end
else
      for i=1:mw^2, for j=1:mw, a((i-1)*mw+j)=a1(i)*a2(j); end, end
end
```

B.18 – *t_o_itae.m*

```
function [w1,w2]=t_o_itae(itaee,f1,kc,ti,td,k1,k2,k3,k4,w1,w2,eta)
%
%     Treino da rede de minimização do ITAE
%       usando "NLMS error correction rule"
%

a1=ambf(f1,k1,2);
a2=ambf(kc,k2,3);
a3=ambf(ti,k3,2);
a4=ambf(td,k4,3);
aa1=form_a(2,a1,a4,0,0,15);
```

```
aa2=form_a(2,a2,a3,0,0,15);
[sa1l,sa1c]=size(aa1);
[sa2l,sa2c]=size(aa2);
o=aa1*w1(1:sa1c)+aa2*w2(1:sa2c);

e=itaee-o;

ww=[w1 w2]; a=[aa1 aa2];
w=aprendn(eta,ww,a,itaee,e);
for i=1:sa1c, w1(i)=w(i); w1=sparse(w1); end
for i=sa1c+1:sa2c, w2(i-sa1c)=w(i); w2=sparse(w2); end

save r_m_itae k1 k2 k3 k4 w1 w2
```

B.19 – *inicia.m*

```
function []=inicia()
%
%     Inicialização do sistema
%     executado quando se abre o bloco START em sistfim
%
global tempo_actual tempo_ante tempo tempo_mud
global entrada_actual entrada_actual_v entrada_ante_v
global uy_actual uy
global aguarda
global amplitude itae kp tt
global t_estac uy_estac estac_ante
global malha
global KC TI TD
global muy tempo_muy uym
global kp den l
global fsig tt kp
muy=[]; tempo_muy=[]; uym=[];
aguarda=0; t_estac=0; uy_estac=0;
entrada_actual_v=-1; entrada_actual=-1;
uy=[]; tempo=[]; tempo_actual=0; tempo_mud=0;
uy_actual=0;
fsig=0;
den=[1 1]; kp=1; l=1; KC=1; TI=1; TD=1;
set_param('sistfim/Adaptacao/aberta+fechada-','Gain','1')
set_param(['sistfim','/',['Controlador - ',13,'componente PI ']],...
```

```
            'Zeros',mat2str(-
1/TI),'Poles',mat2str(0),'Gain',mat2str(KC))
set_param(['sistfim','/',['Controlador - ',13,'componente D']],...
            'Zeros',mat2str(-1/TD),'Poles',mat2str(-
10/TD),'Gain',mat2str(10))
set_param(['sistfim','/','Atraso'],'Delay Time',mat2str(l))
set_param(['sistfim','/',['Função de',13,'Transferência']],...
            'Numerator',mat2str(kp),'Denominator',mat2str(den))
```

B.19 - *id_sint.m*

```
function [xxx]=id_sint(sinal)
%
%     Identificação, Sintonia e Adaptação
%
%     usada pelo modelo sistfim
global tempo_actual tempo_ante tempo tempo_mud
global entrada_actual entrada_actual_v entrada_ante_v
global uy_actual uy
global aguarda
global amplitude fsig itae
global t_estac uy_estac estac_ante
global malha
global KC TI TD tt kp
global uym muy tempo_muy
xxx=zeros(1,4);
tempo_seg=sinal(1);
entrada_seg=sinal(2);
malha=sinal(3);
uy_seg=sinal(4);
if tempo_seg==0, amplitude=sinal(2), end
%     validação do instante de amostragem
if (tempo_seg>tempo_actual & tempo_actual>tempo_ante)
      entrada_ante_v=entrada_actual_v;
      entrada_actual_v=entrada_actual;
      s=length(uy);
      tempo(s+1)=tempo_actual;
      uy(s+1)=uy_actual;
      %     verifica instabilidade
```

```
      if (entrada_ante_v==entrada_actual_v)
            if s>1 & (tempo_actual-tempo_mud)>5
                  maximor(tempo(s-1:s+1),uy(s-1:s+1));
            end
            if aguarda==0
                  t=length(tempo);
                  if (tempo_actual-tempo_mud)>5, mudac(uy_actual); end
                  if (t_estac==0 & (tempo_actual-tempo_mud)>5)
                        est=estacio(tempo,uy);
                        if est==1
                              t_estac=tempo_actual
                              uy_estac=uy_actual
                              esta_ante=1;
                  [fsig kp tt
itae]=identif(tempo,uy,malha,amplitude,[KC TI TD])
                        end
                  end
            end
      else
            tempo_mud=tempo_actual
            tempo=tempo_actual
            uy=uy_actual
            uym=[]; muy=[]; tempo_muy=[];
            if aguarda==1
                  aguarda=0;
            else
                  disp('parâmetros do controlador')
                  [kc ti td]=redes(fsig,kp,tt)
                  if kc~=KC | ti~=TI | td~=TD
                        KC=kc; TI=ti; TD=td;

set_param(['sistfim','/',['Controlador - ',13,'componente PI ']],...
            'Zeros',mat2str(-
1/TI),'Poles',mat2str(0),'Gain',mat2str(KC))
set_param(['sistfim','/',['Controlador - ',13,'componente D']],...
            'Zeros',mat2str(-1/TD),'Poles',mat2str(-
10/TD),'Gain',mat2str(10))
                        muy=[];
                        tempo_muy=[];
                        xlb=[.11081 .2201 .02801];
                        xub=[4.69 1.137 .37];
```

```
                        x0=[kc ti td];
                        [x op]=constr('itaer',x0,[],xlb,xub)
                        kc=x(1);
                        ti=x(2);
                        td=x(3);
                        KC=kc; TI=ti; TD=td;

set_param(['sistfim','/',['Controlador - ',13,'componente PI ']],...
            'Zeros',mat2str(-
1/TI),'Poles',mat2str(0),'Gain',mat2str(KC))
set_param(['sistfim','/',['Controlador - ',13,'componente D']],...
            'Zeros',mat2str(-1/TD),'Poles',mat2str(-
10/TD),'Gain',mat2str(10))
                        aprendo([KC TI TD])
                  else
                        disp('iguais aos anteriores')
                  end
                  if malha==1
                        disp('fechar malha')
                        malha=-1;
                        muy=[]; tempo_muy=[];
                        tempo_actual
                        disp('aguardar próxima transição')
                        aguarda=1;
      set_param('sistfim/Adaptacao/aberta+fechada-','Gain','-1')
                  end
            end
      end
end
entrada_actual=entrada_seg;
tempo_ante=tempo_actual;
tempo_actual=tempo_seg;
uy_actual=uy_seg;
```

B.20 – *maximor.m*

```
function []=maximor(tt,uyy)
%
%     Identifica máximos relativos e utiliza-os para determinar
```

```
%     se o sistema se tornou instável
%
global muy tempo_muy
global aguarda
global t_estac uy_estac
if (uyy(3)<uyy(2) & uyy(2)>uyy(1))
      l=length(muy);
      muy(l+1)=uyy(2);
      tempo_muy(l+1)=tt(2);
      if l==2
            if (muy(3)>muy(2) & muy(2)>muy(1))
                  disp('sistema instável - a malha vai ser aberta')
                  aguarda=1;
                  set_param('sistfim/Adaptacao/aberta+fechada-
','Gain','1')
                  muy=[];
                  tempo_muy=[];
                  malha=1;
                  tt
            else
                  muy=muy(2:3);
                  tempo_muy=tempo_muy(2:3);
            end
      end
end
```

B.21 – *identif.m*

```
function [fsig,kp,tt,itae]=identif(t,uy,malha,amplitude,para)
%
%     Identificação do sistema sigma=1 e determinação do ITAE
%
sigma=1;
load optimos
f=0;
m=length(t);
itae=0;
if malha==1
      disp('Identificação em malha aberta')
```

```
      yfim=uy(length(uy));
      s0=0;
      i=1;
      for i=1:m-1
            dt=t(i+1)-t(i);
            yy=(uy(i+1)+uy(i))/2;
            s0=s0+(yfim-yy)*dt;
            i=i+1;
      end
      kp=yfim/amplitude;
      tt=s0/yfim;
      ssigma=0;
      for i=1:m-1
            tau=(t(i+1)+t(i))/2;
            dtau=t(i+1)-t(i);
            yy=(uy(i+1)+uy(i))/2;
            ssigma=ssigma+exp(-tau*sigma/tt)*(yfim-yy)*dtau;
      end
      f=1-sigma*ssigma/s0;
      fsig=interp1(fsigmaa,fsigma,f,'spline');
else
      disp('Identificação em malha fechada')
      disp('e determinação do ITAE')
      kc=para(1);
      ti=para(2);
      td=para(3);
      ufim=uy(length(uy));
      su0=0;
      i=1;
      for i=1:m-1
            dt=t(i+1)-t(i);
            uu=(uy(i+1)+uy(i))/2;
            su0=su0+(ufim-uu)*dt;
            tt=(t(i+1)+t(i))/2;
            ee=(e(i+1)+e(i))/2;
            itae=itae+tt*ee*dt;
            i=i+1;
      end
      kp=amplitude/ufim;
      tt=ti/(kp*kc)+.9*td-su0/ufim;
      susigma=0;
```

```
      for i=1:m-1
            tau=(t(i+1)+t(i))/2;
            dtau=t(i+1)-t(i);
            uu=(uy(i+1)+uy(i))/2;
            susigma=susigma+exp(-tau*sigma/tt)*(ufim-uu)*dtau;
      end
      frac=(1+sigma*td/(10*tt))/(1+sigma*td/tt);
      fsig=frac*(tt/(tt-sigma*kp*susigma)-
sigma*ti/(kc*kp*(tt+sigma*ti)));
end
```

B.22 – *estacio.m*

```
function est=estacio(t,y)
%
%     Verifica se se atingiu regime estacionário
%
t_ini=0.9*(t(length(t))-t(1))+t(1);
i=find(t>t_ini);
if length(i)>2
      maxy=max(y(i));
      miny=min(y(i));
      if miny==0, miny=miny+1; maxy=maxy+1; end
      desvio=abs(100*(maxy-miny)/miny);
      if desvio>.001
            est=0;
      else
            est=1;
            disp('Atingiu Regime Estacionário')
      end
else
      est=0;
end
```

B.23 – *mudac.m*

```
function mudac(uy)
%
```

```
%     Determina se houve mudança no sistema
%
global uym
global aguarda
lm=length(uym);
if lm==2
      muym=mean(uym);
      if (abs(uy)>1.2*abs(muym)|abs(uy)<0.8*abs(muym))
            disp('mudança na plant - aguardar próxima transição')
            uym
            uym=[];
            aguarda=1;
            t_estac=0;
            muym, uy
      end
end
uym(lm+1)=uy;
if lm==4, uym=uym(2:4); end
```

B.24 – *redes.m*

```
function [kc,ti,td]=redes(fsigma,kp,tt)
%
%     redes de parâmetros do controlador
%
load asmpc
load asmpi
load asmpd
a=ambf(fsigma,kk,3);
kinv=a*wk';
a=ambf(fsigma,ki,3);
tie=a*wi';
a=ambf(fsigma,kd,3);
tde=a*wd';
kc=1/(kinv*kp);
ti=tie*tt;
td=tde*tt;
```

B.25 – *itaer.m*

```
function [f,g]=itaer(x)
%
%     rede de modelização do ITAE
%
load riniitae
global fsig
a1=ambf(fsig,KNOTSF,3);
a2=ambf(x(1),KNOTSC,3);
a3=ambf(x(2),KNOTSI,3);
a4=ambf(x(3),KNOTSD,3);
f=a1*WF'+a2*WC'+a3*WI'+a4*WD'
```

B.26 – *aprendo.m*

```
function aprendo(x)

global fsig tt kp itae
%
%     Treino das redes
%
load riniitae
eta=0.01
for rr=1:4
      a1=ambf(fsig,KNOTSF,3);
      a2=ambf(1/(x(1)*kp),KNOTSC,3);
      a3=ambf(x(2)/tt,KNOTSI,3);
      a4=ambf(x(3)/tt,KNOTSD,3);
      o=a1*wf'+a2*wc'+a3*wi'+a4*wd';
      y=itae;
      e=y-o;
      if rr==1, WF=aprendn(eta,WF,a1,y,e); end
      if rr==2, WC=aprendn(eta,WC,a2,y,e); end
      if rr==3, WI=aprendn(eta,WI,a3,y,e); end
      if rr==4, WD=aprendn(eta,WD,a4,y,e); end
end
save riniitae KNOTSF WF KNOTSC WC KNOTSI WI KNOTSD WD
load asmpc
load asmpi
```

```
load asmpd
for rede=1:4
      if rede==1, k=kk, w=wk, d=1./kc; end
      if rede==2, k=ki, w=wi, d=ti./tt; end
      if rede==3, k=kd, w=wd, d=td./tt; end
      a=ambf(fsig,k,3);
      o=a*w';
      e=d-o;
      w=aprendn(eta,w,a,d,e);
      if rede==1, wk=w, end
      if rede==2, wi=w, end
      if rede==3, wd=w, end
end
if rede==1, save asmpc kk wk,  end
if rede==2, save asmpi ki wi,  end
if rede==3, save asmpd kd wd,  end
```

C – Resultados Intermédios

C. 1 – Excerto do processo de construção duma rede (melhor rede da Tabela 4)

Excerto	Observações
[…]	Nota: o 1º modelo visível para refstep>1 é já o 2º modelo
Growing refstep = 9	
O erro máx em % (valores directos) é 183.7%. O erro médio em % (valores directos) é 20.58%. O erro médio quadrático final no conjunto de treino é 0.3672.	Modelo seleccionado e com melhores erros relativos
O erro máx em % (valores directos) é 189.9%. O erro médio em % (valores directos) é 21.02%. O erro médio quadrático final no conjunto de treino é 0.4828.	
O erro máx em % (valores directos) é 188.7%. O erro médio em % (valores directos) é 21.3%. O erro médio quadrático final no conjunto de treino é 0.5191.	
O erro máx em % (valores directos) é 233.7%. O erro médio em % (valores directos) é 20.81%. O erro médio quadrático final no conjunto de treino é 0.527.	
O erro máx em % (valores directos) é 259.8%. O erro médio em % (valores directos) é 20.68%. O erro médio quadrático final no conjunto de treino é 0.5254. bayk = 4.9416 -249.0212 -40.6574 18.0602 30.3519 27.8320 A melhor hipótese foi o modelo nº 2	
Growing refstep = 10	
O erro máx em % (valores directos) é 191.2%. O erro médio em % (valores directos) é 20.75%. O erro médio quadrático final no conjunto de treino é 0.3599.	
O erro máx em % (valores directos) é 176.4%. O erro médio em % (valores directos) é 20.53%. O erro médio quadrático final no conjunto de treino é 0.3448.	Modelo seleccionado
O erro máx em % (valores directos) é 184.8%. O erro médio em % (valores directos) é 20.29%. O erro médio quadrático final no conjunto de treino é 0.363.	
O erro máx em % (valores directos) é 182.6%. O erro médio em % (valores directos) é 20.15%. O erro médio quadrático final no conjunto de treino é 0.3656.	
O erro máx em % (valores directos) é 160.4%. O erro médio em % (valores directos) é 19.57%. O erro médio quadrático final no conjunto de treino é 0.366.	
O erro máx em % (valores directos) é 124.4%. O erro médio em % (valores directos) é 19.52%. O erro médio quadrático final no conjunto de treino é 0.364. bayk = -249.0212 -251.8353 -273.2543 -231.4432 -225.6139 -224.7252 -229.2530 A melhor hipótese foi o modelo nº 3	Modelo com melhores erros relativos
Pruning	

```
refstep =
    11

O erro máx em % (valores directos) é 186%.
O erro médio em % (valores directos) é 20.65%.
O erro médio quadrático final no conjunto de treino é 0.422.


O erro máx em % (valores directos) é 183.7%.
O erro médio em % (valores directos) é 20.58%.
O erro médio quadrático final no conjunto de treino é 0.3672.


O erro máx em % (valores directos) é 179.3%.
O erro médio em % (valores directos) é 20.61%.
O erro médio quadrático final no conjunto de treino é 0.3556.

O erro máx em % (valores directos) é 210.6%.
O erro médio em % (valores directos) é 22.37%.
O erro médio quadrático final no conjunto de treino é 0.3635.

O erro máx em % (valores directos) é 234.6%.
O erro médio em % (valores directos) é 22.07%.
O erro médio quadrático final no conjunto de treino é 0.3528.

O erro máx em % (valores directos) é 318.5%.
O erro médio em % (valores directos) é 23.11%.
O erro médio quadrático final no conjunto de treino é 0.3557.
bayk =
 -273.2543 -142.9111 -249.0212 -275.0317 -257.0574 -281.4649 -
274.7986
A melhor hipótese foi o modelo nº 6
```

Modelo com melhores erros relativos

Modelo seleccionado

[…]

C. 2 – Sumário dos resultados obtidos ao treinar rede de estrutura $F(1) * \overline{T_d} + \overline{k_{inv}} * \overline{T_i}$**,**

para todas as combinações de ordens das redes

Ordem ($F(1), \overline{k_{inv}}, \overline{T_i}, \overline{T_d}$)	Bayk	MSE	Méd. erro relativo (%)	Máx. erro relativo (%)	q.m.c.m
2,2,2,2	-425.5	0.3547	20.1	312	9
2,2,2,3	-583.3	0.2289	20.6	958.3	7
2,2,2,4	-453	0.3205	21.1	499.9	2
2,2,3,2	-516.1	0.294	16.5	91.5	0
2,2,3,3	-450.1	0.3035	17.9	119.8	2
2,2,3,4	**-680.8**	0.236	21.8	384	5
2,2,4,2	-555.7	0.2556	16.6	83.6	0
2,2,4,3	-518	0.27	19	121.1	5
2,2,4,4	-418.7	0.2871	20.2	318.6	1
2,3,2,2	-534.3	0.2417	19.4	129.8	0
2,3,2,3	-600.9	0.2282	22.1	585.4	**19**
2,3,2,4	981	2.08	71.7	9441	0
2,3,3,2	-479.9	0.31	16.5	207.9	0
2,3,3,3	-631.4	0.2234	28.2	4478	2

Ordem ($F(1), \overline{k_{inv}}, \overline{T_i}, \overline{T_d}$)	Bayk	MSE	Méd. erro relativo (%)	Máx. erro relativo (%)	q.m.c.m
2,3,3,4	-612.4	0.2231	31.2	6520	4
2,3,4,2	-572.8	0.2503	16.07	**67.1**	0
2,3,4,3	1025	2.193	51.4	7685	3
2,3,4,4	1037	2.19	41	4176	0
2,4,2,2	-553	0.2605	21.7	111.4	0
2,4,2,3	-589.9	**0.2077**	24.7	376.5	16
2,4,2,4	-503.3	0.215	97.2	49050	10
2,4,3,2	-549.9	0.2512	16.91	122.2	0
2,4,3,3	-614.3	0.2189	24.4	1075	4
2,4,3,4	-575	0.2297	25.3	1411	4
2,4,4,2	-549.9	0.2491	16.9	93.8	0
2,4,4,3	-497.1	0.2428	23.4	885.7	10
2,4,4,4	-588.9	0.2278	23.9	1493	2
3,2,2,2	-445.9	0.3314	19.8	361	0
3,2,2,3	-442.1	0.299	22.6	963	2
3,2,2,4	1056	2.136	26.8	297.2	0
3,2,3,2	-485.2	0.2931	16.8	84.1	0
3,2,3,3	-552.3	0.261	17	122.3	4
3,2,3,4	-521.8	0.2644	18.1	136.2	4
3,2,4,2	-540.7	0.254	16.6	79.5	0
3,2,4,3	-518.5	0.2676	18	160.5	4
3,2,4,4	-378	0.3487	20.5	273.6	1
3,3,2,2	-457.6	0.3083	17.2	193.3	1
3,3,2,3	-477	0.2817	25	367.9	8
3,3,2,4	-427.5	0.3199	27.8	446.2	2
3,3,3,2	-457.5	0.3109	17	155.7	0
3,3,3,3	-550.33	0.251	23	1660	0
3,3,3,4	-604.5	0.2234	18	435	4
3,3,4,2	-549	0.2452	15.8	78.3	1
3,3,4,3	-555.1	0.2452	16	120.7	6
3,3,4,4	1056	2.207	53.3	9841	0
3,4,2,2	-525.1	0.2404	18	103.8	0
3,4,2,3	1018	2.054	47.6	9612	1
3,4,2,4	-492.3	0.2631	53	7406	2
3,4,3,2	-541.8	0.2688	19.5	152.6	1
3,4,3,3	-623.6	0.2129	18.9	362.9	3
3,4,3,4	-568.5	0.2355	32.3	8505	3
3,4,4,2	-543.9	0.2468	17	98.7	1
3,4,4,3	-500.2	0.2562	27.8	1857	3
3,4,4,4	-558.9	0.2231	34.4	5731	9
4,2,2,2	-469.6	0.3245	19.9	298.5	1
4,2,2,3	-450	0.3112	20.9	773.3	2
4,2,2,4	-432.5	0.3287	30	7900	2
4,2,3,2	997	2.103	49.4	5807	0
4,2,3,3	-520.8	0.2691	17.54	254.9	3
4,2,3,4	-526.3	0.2629	17.48	129.6	4

Ordem ($F(1), \overline{k_{inv}}, \overline{T_i}, \overline{T_d}$)	Bayk	MSE	Méd. erro relativo (%)	Máx. erro relativo (%)	q.m.c.m
4,2,4,2	-508.2	0.2516	16.3	87.5	1
4,2,4,3	-634.7	0.23	17	119	3
4,2,4,4	-471.8	0.2812	19.2	255.8	2
4,3,2,2	1023	2.082	34.4	1483	0
4,3,2,3	1034	2.093	38.7	2647	0
4,3,2,4	1038	2.069	49.2	4517	0
4,3,3,2	-538.4	0.2548	16.2	74.6	1
4,3,3,3	-596.2	0.2432	22.3	239	7
4,3,3,4	-589	0.2203	17.2	179.8	5
4,3,4,2	-538.2	0.2425	**15.3**	72.87	1
4,3,4,3	-572	0.2123	15.5	84.5	13
4,3,4,4	1071	2.174	49.9	6248	0
4,4,2,2	1034	2.095	43.9	8341	1
4,4,2,3	1058	2.08	40.1	1640	1
4,4,2,4	1069	2.082	51.2	1105	1
4,4,3,2	-510.2	0.266	19.5	223.6	1
4,4,3,3	-572.5	0.2324	21.4	330.9	12
4,4,3,4	-546.3	0.2361	21.7	378.7	2
4,4,4,2	-618.1	0.2251	17.1	97.3	2
4,4,4,3	-580.8	0.2171	18.2	167.8	10
4,4,4,4	-529.1	0.2239	29	665	9

Nota: q.m.c.m - Quantidade de mínimos correctamente mapeados