Available online at www.sciencedirect.com

SCIENCE DIRECT®

Fuzzy Sets and Systems 154 (2005) 157–181

FUZZY sets and systems

www.elsevier.com/locate/fss

A methodology to design stable nonlinear fuzzy control systems

José M. Andújar*, Antonio J. Barragán

Departamento de Ingeniería Electrónica, Sistemas Informáticos y Automática, Universidad de Huelva, Escuela Politécnica Superior, Carretera Huelva-La Rábida, Palos de la Frontera 21071, Huelva, Spain

Received 29 March 2004; received in revised form 15 December 2004; accepted 18 March 2005
Available online 9 April 2005

Abstract

The stability of nonlinear systems has to be investigated without making linear approaches. In order to do this, there are several techniques based on Lyapunov's second method. For example, Krasovskii's method allows to prove the sufficient condition for the asymptotic stability of nonlinear systems. This method requires the calculation of the Jacobian matrix. In this paper, an equivalent mathematical closed loop model of a multivariable nonlinear control system based on fuzzy logic theory is developed. Later, this model is used to compute the Jacobian matrix of a closed loop fuzzy system. Next, an algorithm to solve the Jacobian matrix is proposed. The algorithm uses a methodology based on the extension of the state vector. The developed algorithm is completely general: it is independent of the type of membership function that is chosen for building the fuzzy plant and controller models, and it allows the compound of different membership functions in a same model. We have developed a MATLAB's[1] function that implements the improved algorithm, together with a series of additional applications for its use. The designed software provides complementary functions to facilitate the reading and writing of fuzzy systems, as well as an interface that makes possible the use of all the developed functions from the MATLAB's environment, which allows to complement and to extend the possibilities of the MATLAB's Fuzzy Logic Toolbox. An example with a fuzzy controller for a nonlinear system to illustrate the design procedure is presented. The work developed in this paper can be useful for the analysis and synthesis of fuzzy control systems.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Nonlinear fuzzy systems; Fuzzy control design; Stability; Jacobian matrix; Algorithm; Software design

* Corresponding author. Tel.: +34959017380; fax: +34959017304.
E-mail addresses: andujar@diesia.uhu.es (J.M. Andújar), antonio.barragan@diesia.uhu.es (A.J. Barragán).
[1] MATLAB is a registered trademark of The Math Works, Inc.

0165-0114/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.fss.2005.03.006

1. Introduction

One of the strongest drawbacks for the use of fuzzy logic in the control system design is its scarce mathematical formalization—the solutions are often empirical and require an expert's knowledge—and the very frequent solutions of ad-hoc problems, whereby the extrapolation of the solutions to another type of problems becomes unviable.

The authors of this paper are interested in the formalization of control fuzzy systems [1,2,8], so that the mathematical tools of analysis and synthesis of control systems accepted by the scientific community can be applicable to them.

Lyapunov's theory can be used to analyze the stability of Takagi–Sugeno–Kang (TSK) fuzzy systems [15,36,41], nevertheless, the problem is always the same: to find the suitable Lyapunov function.

Most of the developed methods to prove the stability of fuzzy control systems have been based on classic control theory [26]. The most well known methods designed to determine the stability of fuzzy systems are the following: Popov's stability criterion [11,17], the circle criterion [9,16,18], direct Lyapunov's method [10,13,16,21,30,31,34,37–39], analysis of system stability in phase space [9,19], the describing function method [4], methods of stability indices and systems robustness [4,5,10], methods based on the theory of input/output stability [4,9,16,18,20], conicity criterion [3,4,9], methods based on Popov's hyperstability theory [17,25,27,28] and heuristic methods [4,32,42,40].

In addition to the previous cites, there are classical general texts about nonlinear control systems [12,14,29,33] which analyze the above-mentioned methods and others.

If we examine the asymptotical stability of the equilibrium states of nonlinear systems, we will find it inadequate for a stability analysis of the linearized model of the nonlinear system. Stability of nonlinear systems must be investigated without making a linear approximation of them. To do this, there are several methods based on Lyapunov's second method, for example, Krasovskii's theorem.

Krasovskii's theorem for global asymptotic stability offers sufficient conditions for nonlinear systems and necessary and sufficient conditions for linear systems. An equilibrium state of a nonlinear system can be stable even if the conditions specified in this theorem are not satisfied; however the synthesis of systems that complete this theorem assures that they are asymptotically stable globally.

Krasovskii's theorem demonstrates that given a system $\dot{\mathbf{x}} = \boldsymbol{\xi}(\mathbf{x})$, if the sum of the system's Jacobian matrix $\mathbf{J}(\mathbf{x})$ and its transposed $\mathbf{J}(\mathbf{x})^{\mathrm{T}}$ is negative definite in a region around the equilibrium state, then the equilibrium state at the origin is asymptotically stable. A Lyapunov function for this system is $\boldsymbol{\xi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\xi}(\mathbf{x})$.

In this work, we present a general algorithm to compute the Jacobian matrix of a closed loop fuzzy system. This is a fundamental step in several techniques of analysis and synthesis of control systems as for example in Krasovskii's method.

The algorithm has been improved by a simplification of the equations based on the state vector extension. The algorithm is independent of the type of membership function used for building the models of the plant and the controller. The proposed algorithm in this paper also allows combining different types of membership functions in the same model.

In this paper, we also present a software that implements the improved algorithm, together with a series of additional applications for its use. The designed software provides complementary functions to facilitate the reading and writing of fuzzy systems, as well as an interface that makes possible the use of all the developed functions from the MATLAB's environment, which allows to complement and to extend the possibilities of the MATLAB's Fuzzy Logic Toolbox.

Finally, it is important to take into account that the error the system may have as regards the actual system at the point where the Jacobian matrix is calculated affects the matrix value. For this reason, the proposed design methodology for the synthesis of the controller does not use the Jacobian matrix directly, i.e. by using linearization techniques or adaptive methods per reference model. Instead, in the methodology proposed in this work, what is relevant is not the $J(x)$ value itself, but the sign of the eigenvalues of $\mathbf{J}(\mathbf{x}) + \mathbf{J}^{\mathrm{T}}(\mathbf{x})$. From our point of view, this makes the method more robust facing the plant modeling errors.

This paper is organized in the following sections: Section 2 describes the mathematical background of a fuzzy plant model and a fuzzy controller. Section 3 develops the expressions that are obtained deriving the closed loop fuzzy system model with regard to every coordinate of the state space. Developed expressions in the previous section are used to implement an algorithm in Section 4. Next, in Section 5, we present a MATLAB's function that implements the improved algorithm, together with a series of additional applications for its use. The proposed methodology is illustrated by an example. Finally, we provide some conclusions, an appendix and references.

2. Nonlinear fuzzy multivariable model of a control system

Consider a plant to be controlled, described by a dynamical system of the general form

$$\begin{aligned}
\dot{x}_1 &= f_1(x_1, x_2, \ldots, x_n,\ u_1, u_2, \ldots, u_m),\\
\dot{x}_2 &= f_2(x_1, x_2, \ldots, x_n,\ u_1, u_2, \ldots, u_m),\\
&\ \ \vdots\\
\dot{x}_n &= f_n(x_1, x_2, \ldots, x_n,\ u_1, u_2, \ldots, u_m),
\end{aligned} \tag{1}$$

where the multivariable controller is defined by

$$\begin{aligned}
u_1 &= \phi_1(x_1, x_2, \ldots, x_n),\\
u_2 &= \phi_2(x_1, x_2, \ldots, x_n),\\
&\ \ \vdots\\
u_m &= \phi_m(x_1, x_2, \ldots, x_n).
\end{aligned} \tag{2}$$

Expressions (1) and (2) indicate a completely general system, without constraints neither in the order of the state vector nor in the control one. Also, the expressions of the state vector and of the control vector can be nonlinear.

The system described by expressions (1) and (2) can be written in an abbreviated form as

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \tag{3}$$

where $\dot{\mathbf{x}} = (\dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n)^{\mathrm{T}}$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$, $\mathbf{u} = (u_1, u_2, \ldots, u_m)^{\mathrm{T}}$.

An equivalent multiple-input–multiple-output nonlinear dynamic fuzzy model to the process can be represented by the following group of rules [6,7,24,35]:

$$\begin{aligned}
&R^{(l)}:\ \text{IF } x_1 \text{ is } A^l_{1i} \text{ and } x_2 \text{ is } A^l_{2i}, \ldots, \text{ and } x_n \text{ is } A^l_{ni}\\
&\text{and } u_1 \text{ is } B^l_{1i} \text{ and } u_2 \text{ is } B^l_{2i}, \ldots, \text{ and } u_m \text{ is } B^l_{mi}\\
&\text{THEN } \dot{x}^l_i \text{ is } g^l_i(\mathbf{x}, \mathbf{u}; \boldsymbol{\theta}^l_i),
\end{aligned} \tag{4}$$

where $l = 1, \dots, M$ is the number of rules, A^l_{ki} are the fuzzy sets defined in a universe of discourse for the state variable x_k, $k = 1, \dots, n$, and the first-order differential equation i, B^l_{ji} is the fuzzy set defined in a universe of discourse for the jth control signal u_j, $j = 1, \dots, m$, and the ith first-order differential equation of the process; g^l_i and $\boldsymbol{\theta}^l_i$ will be defined next.

The consequent term in (4) can be a nonlinear consequent or a TSK consequent with affine term [35]:

$$g^l_i(\mathbf{x}, \mathbf{u}, \boldsymbol{\theta}^l_i) = a^l_{0i} + a^l_{1i}x_1 + \cdots + a^l_{ni}x_n + b^l_{1i}u_1 + b^l_{2i}u_2 + \cdots + b^l_{mi}u_m, \tag{5}$$

where a^l_{ki} and b^l_{ji} represent the respective constant coefficients for the state variable x_k and for the jth signal of control of the differential equation i that represents the process. Vector $\boldsymbol{\theta}^l_i$ represents the set of adaptive parameters:

$$\boldsymbol{\theta}^l_i = (a^l_{0i}, a^l_{1i}, \dots, a^l_{ni},\ b^l_{1i}, b^l_{2i}, \dots, b^l_{mi})^{\mathrm{T}}. \tag{6}$$

Considering a TSK fuzzy system with a product inference operator and center average defuzzifier, expression (3) can be written as [40]

$$\dot{x}_i = \frac{\sum_{l=1}^M w^l_i g^l_i(\mathbf{x}, \mathbf{u}; \boldsymbol{\theta}^l_i)}{\sum_{l=1}^M w^l_i}, \tag{7}$$

where w^l_i is the firing degree (matching degree or fulfillment degree) of the rule l, that is,

$$w^l_i = \prod_{k=1}^{n} \mu^l_{F_k}(x_k, \boldsymbol{\sigma}^l_i) \prod_{j=1}^{m} \mu^l_{F_j}(u_j, \boldsymbol{\delta}^l_i). \tag{8}$$

Being $\mu^l_{F_k}(x_k, \boldsymbol{\sigma}^l_i)$ and $\mu^l_{F_j}(u_j, \boldsymbol{\delta}^l_i)$ the membership functions defined in their corresponding universe of discourse X and U from a fuzzy set $F \in (X \cup U)$ represented by a set of orderly pairs $\{(x_k, \mu^l_{F_k}(x_k, \boldsymbol{\sigma}^l_i)) : x_k \in X\}$ and $\{(u_j, \mu^l_{F_j}(u_j, \boldsymbol{\delta}^l_i)) : u_j \in U\}$.

If the consequent of the rule is of type (5), then (7) can be written as

$$\dot{x}_i = \frac{1}{\sum_{l=1}^M w^l_i} \left[\begin{array}{c} \sum\limits_{l=1}^M w^l_i a^l_{0i} + \sum\limits_{l=1}^M w^l_i a^l_{1i} x_1 + \sum\limits_{l=1}^M w^l_i a^l_{2i} x_2 + \cdots + \sum\limits_{l=1}^M w^l_i a^l_{ni} x_n \\ + \sum\limits_{l=1}^M w^l_i b^l_{1i} u_1 + \sum\limits_{l=1}^M w^l_i b^l_{2i} u_2 + \cdots + \sum\limits_{l=1}^M w^l_i b^l_{mi} u_m \end{array} \right], \tag{9}$$

this is

$$\begin{aligned} \dot{x}_i &= a^t_{0i} + a^t_{1i}x_1 + a^t_{2i}x_2 + \cdots + a^t_{ni}x_n + b^t_{1i}u_1 + b^t_{2i}u_2 + \cdots + b^t_{mi}u_m \\ &= a^t_{0i} + \sum_{k=1}^{n} a^t_{ki}x_k + \sum_{j=1}^{m} b^t_{ji}u_j, \end{aligned} \tag{10}$$

where a^t_{0i}, a^t_{ki} and b^t_{ji} are variable coefficients [43] computed as

$$a^t_{0i} = \frac{\sum_{l=1}^M w^l_i a^l_{0i}}{\sum_{l=1}^M w^l_i}, \quad a^t_{ki} = \frac{\sum_{l=1}^M w^l_i a^l_{ki}}{\sum_{l=1}^M w^l_i}, \quad k = 1, \dots, n. \tag{11}$$

And

$$b_{ji}^t = \frac{\sum_{l=1}^{M} w_i^l b_{ji}^l}{\sum_{l=1}^{M} w_i^l}, \quad j = 1, \ldots, m. \tag{12}$$

Up to here, the methodology has been developed to obtain the fuzzy plant model. Next, we will develop the fuzzy controller model.

The process of the fuzzy controller design that stabilizes the plant is considered independent of the process of identification of the plant. Therefore, the fuzzy partition of the universe of discourse of the state for the plant model and for the controller design does not need to coincide. Of course, there is also no reason for the model plant and the controller to have the same number of rules.

Keeping in mind what has been said and proceeding in a similar way as for the achievement of the fuzzy plant model, the fuzzy controller can be represented by the following group of rules:

$$\begin{aligned} R^{(r)}\colon & \text{ IF } x_1 \text{ is } C_{1j}^r \text{ and } x_2 \text{ is } C_{2j}^r, \ldots, \text{ and } x_n \text{ is } C_{nj}^r \\ & \text{ THEN } u_j^r = c_{0j}^r + c_{1j}^r x_1 + \cdots + c_{nj}^r x_n, \end{aligned} \tag{13}$$

where C_{kj}^r represents the fuzzy sets of $k = 1, \ldots, n$ state variables, with $j = 1, \ldots, m$ control actions and $r = 1, \ldots, N$ rules.

Working in a similar way shown in (8)–(10), control signals can be expressed as follows:

$$u_j = \frac{1}{\sum_{r=1}^{N} \omega_j^r} \left[\sum_{r=1}^{N} \omega_j^r c_{0j}^r + \sum_{r=1}^{N} \omega_j^r c_{1j}^r x_1 + \sum_{r=1}^{N} \omega_j^r c_{2j}^r x_2 + \cdots + \sum_{r=1}^{N} \omega_j^r c_{nj}^r x_n \right]. \tag{14}$$

Or, in a more simplified way:

$$u_j = c_{0j}^t + c_{1j}^t x_1 + c_{2j}^t x_2 + \cdots + c_{nj}^t x_n = c_{0j}^t + \sum_{k=1}^{n} c_{kj}^t x_k, \tag{15}$$

where the variable coefficients are given by

$$c_{0j}^t = \frac{\sum_{r=1}^{N} \omega_j^r c_{0j}^r}{\sum_{r=1}^{N} \omega_j^r}, \quad c_{kj}^t = \frac{\sum_{r=1}^{N} \omega_j^r c_{kj}^r}{\sum_{r=1}^{N} \omega_j^r}, \quad k = 1, \ldots, n. \tag{16}$$

And the matching degree of the rule r of the controller, by

$$\omega_j^r = \prod_{k=1}^{n} \mu_{F_k}^r (x_k, \boldsymbol{\varphi}_j^r), \tag{17}$$

where vector $\boldsymbol{\varphi}_j^r$ represents the set of adaptive parameters of the controller's rule base antecedents.

Replacing expression (15) into (10), it is possible to write the following nonlinear closed loop output function:

$$\dot{x}_i = a_{0i}^t + \sum_{j=1}^{m} b_{ji}^t c_{0j}^t + \sum_{k=1}^{n} \left(a_{ki}^t + \sum_{j=1}^{m} b_{ji}^t c_{kj}^t \right) x_k, \quad i, k = 1, \ldots, n \quad \text{and} \quad j = 1, \ldots, m. \tag{18}$$

Expression (18) is an equivalent mathematical closed loop model of a multivariable nonlinear control system based on fuzzy logic theory.

In order to simplify Eq. (18) the state vector will be extended in a coordinate, that is,

$$\tilde{\mathbf{x}} = \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} = (x_0, x_1, \ldots, x_n)^{\mathrm{T}}. \tag{19}$$

Replacing the extended state vector into (18) the simplified following expression is obtained

$$\dot{x}_i = \sum_{k=0}^{n} \left(a_{ki}^t + \sum_{j=1}^{m} b_{ji}^t c_{kj}^t \right) \tilde{x}_k, \quad k = 0, 1, \ldots, n, \ i = 1, \ldots, n \quad \text{and} \quad j = 1, \ldots, m. \tag{20}$$

3. Computation of the Jacobian matrix of a closed loop fuzzy system

Let the multivariable nonlinear control system modeled by Eq. (20), which can be represented in the following generic way:

$$\begin{aligned} \dot{x}_1 &= \xi_1(x_1, x_2, \ldots, x_n), \\ \dot{x}_2 &= \xi_2(x_1, x_2, \ldots, x_n), \\ &\vdots \\ \dot{x}_n &= \xi_n(x_1, x_2, \ldots, x_n). \end{aligned} \tag{21}$$

Let $\mathbf{x}^e = (x_1^e, x_2^e, \ldots, x_n^e)^{\mathrm{T}}$ be an equilibrium state of system (18). Whereas the function is continuous over the range of interest, a Taylor series expansion about the point $\mathbf{x}^e$ may be used:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}(x_i - x_i^e) = (x_1 - x_1^e) \left. \frac{\partial \xi_i(x_1, x_2, \ldots, x_n)}{\partial x_1} \right|_{\mathbf{x}^e} \\ + \cdots + (x_n - x_n^e) \left. \frac{\partial \xi_i(x_1, x_2, \ldots, x_n)}{\partial x_n} \right|_{\mathbf{x}^e} + R_i(\mathbf{x}^e). \end{aligned} \tag{22}$$

When the nonlinear term $R_i(\mathbf{x}^e)$ fulfills (23), being K_i a sufficiently small constant, then the Jacobian matrix of the system determines the properties of stability of the equilibrium state $\mathbf{x}^e$:

$$|R_i(\mathbf{x}^e)| \leqslant K_i[(x_1 - x_1^e)^2 + \cdots + (x_n - x_n^e)^2]. \tag{23}$$

The Jacobian matrix is calculated applying the following expression:

$$\mathbf{J}_{iq}(\mathbf{x})\big|_{\mathbf{x}^e} = \begin{pmatrix} \left. \dfrac{\partial \xi_1\,(x_1, \ldots, x_n)}{\partial x_1} \right|_{\mathbf{x}^e} & \cdots & \left. \dfrac{\partial \xi_1\,(x_1, \ldots, x_n)}{\partial x_n} \right|_{\mathbf{x}^e} \\ \vdots & \cdots & \vdots \\ \left. \dfrac{\partial \xi_n\,(x_1, \ldots, x_n)}{\partial x_1} \right|_{\mathbf{x}^e} & \cdots & \left. \dfrac{\partial \xi_1\,(x_1, \ldots, x_n)}{\partial x_n} \right|_{\mathbf{x}^e} \end{pmatrix}. \tag{24}$$

In the previous expression it is needed to calculate the partial derivative from Eq. (20) with regard to each one of the state variables. This equation can be divided in two addends:

$$\dot{x}_i = \sum_{k=0}^{n} \left(a_{ki}^t + \sum_{j=1}^{m} b_{ji}^t c_{kj}^t \right) \tilde{x}_k = \sum_{k=0}^{n} a_{ki}^t \tilde{x}_k + \sum_{k=0}^{n} \sum_{j=1}^{m} b_{ji}^t c_{kj}^t \tilde{x}_k. \tag{25}$$

The partial derivatives will be obtained in a generic form for the $q = 1, \ldots, n$ state variables and the $i = 1, \ldots, n$ first degree differentials equations.

3.1. *Derivative of the first addend*

The first partial derivative to calculate is from the first adding of Eq. (25):

$$\frac{\partial(\sum_{k=0}^{n} a_{ki}^t \tilde{x}_k)}{\partial x_q} = \sum_{k=0}^{n} \frac{\partial(a_{ki}^t \tilde{x}_k)}{\partial x_q} = \sum_{k=0}^{n} \left(\frac{\partial a_{ki}^t}{\partial x_q} \tilde{x}_k \right) + a_{qi}^t. \tag{26}$$

Bearing in mind (11):

$$\frac{\partial a_{ki}^t}{\partial x_q} = \frac{\partial \left(\dfrac{\sum_{l=1}^{M} w_i^l a_{ki}^l}{\sum_{l=1}^{M} w_i^l} \right)}{\partial x_q} = \frac{\dfrac{\partial(\sum_{l=1}^{M} w_i^l a_{ki}^l)}{\partial x_q} \sum_{l=1}^{M} w_i^l - \dfrac{\partial(\sum_{l=1}^{M} w_i^l)}{\partial x_q} \sum_{l=1}^{M} w_i^l a_{ki}^l}{(\sum_{l=1}^{M} w_i^l)^2}, \tag{27}$$

but

$$\frac{\partial(\sum_{l=1}^{M} w_i^l a_{ki}^l)}{\partial x_q} = \sum_{l=1}^{M} \frac{\partial w_i^l}{\partial x_q} a_{ki}^l, \tag{28}$$

and

$$\frac{\partial(\sum_{l=1}^{M} w_i^l)}{\partial x_q} = \sum_{l=1}^{M} \frac{\partial w_i^l}{\partial x_q}. \tag{29}$$

Replacing now the two previous expressions in (27) and simplifying, it results in

$$\frac{\partial a_{ki}^t}{\partial x_q} = \frac{\sum_{l=1}^{M} \sum_{p=1}^{M} \left(\dfrac{\partial w_i^l}{\partial x_q} w_i^p (a_{ki}^l - a_{ki}^p) \right)}{(\sum_{l=1}^{M} w_i^l)^2}. \tag{30}$$

Finally, Eq. (26) can be written as

$$\frac{\partial(\sum_{k=0}^{n} a_{ki}^t \tilde{x}_k)}{\partial x_q} = \sum_{k=0}^{n} \left(\frac{\sum_{l=1}^{M} \sum_{p=1}^{M} \left(\dfrac{\partial w_i^l}{\partial x_q} w_i^p (a_{ki}^l - a_{ki}^p) \right)}{(\sum_{l=1}^{M} w_i^l)^2} \tilde{x}_k \right) + a_{qi}^t. \tag{31}$$

Notice that to evaluate expression (31) it is necessary to calculate previously the derivative of the matching degree of the rules of the plant, given from Eq. (8). This point is solved in Section 3.4.

3.2. *Derivative of the second addend*

To calculate the derivative of the second addend the following expression must be solved:

$$\frac{\partial(\sum_{k=0}^{n}\sum_{j=1}^{m} b_{ji}^{t} c_{kj}^{t} \tilde{x}_k)}{\partial x_q} = \sum_{k=0}^{n}\left(\frac{\partial(\sum_{j=1}^{m} b_{ji}^{t} c_{kj}^{t})}{\partial x_q}\tilde{x}_k\right) + \sum_{j=1}^{m} b_{ji}^{t} c_{qj}^{t}, \tag{32}$$

where

$$\frac{\partial(\sum_{j=1}^{m} b_{ji}^{t} c_{kj}^{t})}{\partial x_q} = \sum_{j=1}^{m}\left(\frac{\partial b_{ji}^{t}}{\partial x_q} c_{kj}^{t} + \frac{\partial c_{kj}^{t}}{\partial x_q} b_{ji}^{t}\right). \tag{33}$$

Given the similarity of the expressions for the calculation of a_{ki}^{t} and b_{ji}^{t}, it is easy to deduce that

$$\frac{\partial b_{ji}^{t}}{\partial x_q} = \frac{\sum_{l=1}^{M}\sum_{p=1}^{M}\left(\frac{\partial w_i^l}{\partial x_q} w_i^p (b_{ji}^{l} - b_{ji}^{p})\right)}{(\sum_{l=1}^{M} w_i^l)^2}. \tag{34}$$

The derivative of the term corresponding to the fuzzy controller will be:

$$\frac{\partial c_{kj}^{t}}{\partial x_q} = \frac{\frac{\partial(\sum_{r=1}^{N}\omega_j^r c_{kj}^r)}{\partial x_q}\sum_{r=1}^{N}\omega_j^r - \sum_{r=1}^{N}\omega_j^r c_{kj}^r \frac{\partial(\sum_{r=1}^{N}\omega_j^r)}{\partial x_q}}{(\sum_{r=1}^{N}\omega_j^r)^2}, \tag{35}$$

where

$$\frac{\partial(\sum_{r=1}^{N}\omega_j^r c_{kj}^r)}{\partial x_q} = \sum_{r=1}^{N}\frac{\partial \omega_j^r}{\partial x_q} c_{kj}^r, \tag{36}$$

and

$$\frac{\partial(\sum_{r=1}^{N}\omega_j^r)}{\partial x_q} = \sum_{r=1}^{N}\frac{\partial \omega_j^r}{\partial x_q}. \tag{37}$$

Replacing Eqs. (36) and (37) in (35), we obtain:

$$\frac{\partial c_{kj}^{t}}{\partial x_q} = \frac{\sum_{r=1}^{N}\sum_{s=1}^{N}\left(\frac{\partial \omega_j^r}{\partial x_q}\omega_j^s (c_{kj}^r - c_{kj}^s)\right)}{(\sum_{r=1}^{N}\omega_j^r)^2}. \tag{38}$$

Notice that to evaluate expression (38) it is necessary to calculate previously the derivative of the matching degree of the rules of the controller, given from Eq. (17). This point is solved in Section 3.3.

As soon as intermediate expressions (34) and (38) are calculated, they can be replaced in (33):

$$\frac{\partial(\sum_{j=1}^{m} b_{ji}^t c_{kj}^t)}{\partial x_q} = \sum_{j=1}^{m} \left(\frac{\sum_{l=1}^{M}\sum_{p=1}^{M} \left(\frac{\partial w_i^l}{\partial x_q} w_i^p (b_{ji}^l - b_{ji}^p) \right)}{(\sum_{l=1}^{M} w_i^l)^2} c_{kj}^t + \frac{\sum_{r=1}^{N}\sum_{s=1}^{N} \left(\frac{\partial \omega_j^r}{\partial x_q} \omega_j^s (c_{kj}^r - c_{kj}^s) \right)}{(\sum_{r=1}^{N} \omega_j^r)^2} b_{ji}^t \right). \tag{39}$$

Thus, the derivative of the second addend given by Eq. (32) is

$$\frac{\partial(\sum_{k=0}^{n}\sum_{j=1}^{m} b_{ji}^t c_{kj}^t \tilde{x}_k)}{\partial x_q} = \sum_{k=0}^{n} \left(\sum_{j=1}^{m} \left(\frac{\sum_{l=1}^{M}\sum_{p=1}^{M} \left(\frac{\partial w_i^l}{\partial x_q} w_i^p (b_{ji}^l - b_{ji}^p) \right)}{(\sum_{l=1}^{M} w_i^l)^2} c_{kj}^t + \frac{\sum_{r=1}^{N}\sum_{s=1}^{N} \left(\frac{\partial \omega_j^r}{\partial x_q} \omega_j^s (c_{kj}^r - c_{kj}^s) \right)}{(\sum_{r=1}^{N} \omega_j^r)^2} b_{ji}^t \right) \tilde{x}_k \right) + \sum_{j=1}^{m} b_{ji}^t c_{qj}^t. \tag{40}$$

3.3. Derivative of the matching degree of the rules of the controller

Given expression (17) that defines the matching degree of the rules of the controller, the partial derivative of this expression with regard to every state variable is obtained from

$$\frac{\partial \omega_j^r}{\partial x_q} = \frac{\partial}{\partial x_q}\left(\prod_{k=1}^{n} \mu_{F_k}^r (x_k, \boldsymbol{\varphi}_j^r) \right) = \frac{\partial \mu_{F_1}^r (x_1, \boldsymbol{\varphi}_j^r)}{\partial x_q} \prod_{\substack{k=1 \\ k\neq 1}}^{n} \mu_{F_k}^r (x_k, \boldsymbol{\varphi}_j^r) + \cdots + \frac{\partial \mu_{F_n}^r (x_n, \boldsymbol{\varphi}_j^r)}{\partial x_q} \prod_{\substack{k=1 \\ k\neq n}}^{n} \mu_{F_k}^r (x_k, \boldsymbol{\varphi}_j^r). \tag{41}$$

Bearing in mind that

$$\frac{\partial f(x_i)}{\partial x_j} = 0 \quad \forall i \neq j, \tag{42}$$

Eq. (41) can be written as

$$\frac{\partial \omega_j^r}{\partial x_q} = \frac{\partial \mu^r_{F_q}(x_q, \boldsymbol{\varphi}_j^r)}{\partial x_q} \prod_{\substack{k=1\\k\neq q}}^{n} \mu^r_{F_k}(x_k, \boldsymbol{\varphi}_j^r). \tag{43}$$

3.4. Derivative of the matching degree of the rules of the plant

Given expression (8) that defines the matching degree of the rules of the plant, the partial derivative of this expression with regard to every state variable is obtained from

$$\frac{\partial w_i^l}{\partial x_q} = \frac{\partial}{\partial x_q}\left(\underbrace{\prod\nolimits_{k=1}^{n} \mu^l_{F_k}(x_k, \boldsymbol{\sigma}_i^l)}_{w_i^l(\mathbf{x})}\underbrace{\prod\nolimits_{j=1}^{m} \mu^l_{F_j}(u_j, \boldsymbol{\delta}_i^l)}_{w_i^l(\mathbf{u})}\right) = w_i^l(\mathbf{u})\frac{\partial w_i^l(\mathbf{x})}{\partial x_q} + w_i^l(\mathbf{x})\frac{\partial w_i^l(\mathbf{u})}{\partial x_q}, \tag{44}$$

where

$$\frac{\partial w_i^l(\mathbf{u})}{\partial x_q} = \frac{\partial}{\partial x_q}\left(\prod_{j=1}^{m} \mu^l_{F_j}(u_j, \boldsymbol{\delta}_i^l)\right) = \frac{\partial \mu^l_{F_1}(u_1, \boldsymbol{\delta}_i^l)}{\partial x_q}\prod_{\substack{j=1\\j\neq 1}}^{m} \mu^l_{F_j}(u_j, \boldsymbol{\delta}_i^l)$$
$$+ \cdots + \frac{\partial \mu^l_{F_m}(u_m, \boldsymbol{\delta}_i^l)}{\partial x_q}\prod_{\substack{j=1\\j\neq m}}^{m} \mu^l_{F_j}(u_j, \boldsymbol{\delta}_i^l), \tag{45}$$

then

$$\frac{\partial w_i^l(\mathbf{u})}{\partial x_q} = \sum_{v=1}^{m}\left(\frac{\partial \mu^l_{F_v}\left(u_v, \boldsymbol{\delta}_i^l\right)}{\partial x_q}\prod_{\substack{j=1\\j\neq v}}^{m} \mu^l_{F_j}(u_j, \boldsymbol{\delta}_i^l)\right). \tag{46}$$

Now, bearing in mind (42):

$$\frac{\partial w_i^l(\mathbf{x})}{\partial x_q} = \frac{\partial}{\partial x_q}\left(\prod_{k=1}^{n} \mu^l_{F_k}(x_k, \boldsymbol{\sigma}_i^l)\right) = \frac{\partial \mu^l_{F_q}(x_q, \boldsymbol{\sigma}_i^l)}{\partial x_q}\prod_{\substack{k=1\\k\neq q}}^{n} \mu^l_{F_k}(x_k, \boldsymbol{\sigma}_i^l). \tag{47}$$

Replacing Eqs. (46) and (47) in (44):

$$\frac{\partial w_i^l}{\partial x_q} = \frac{\partial \mu^l_{F_q}(x_q, \boldsymbol{\sigma}_i^l)}{\partial x_q}\prod_{j=1}^{m} \mu^l_{F_j}(u_j, \boldsymbol{\delta}_i^l)\prod_{\substack{k=1\\k\neq q}}^{n} \mu^l_{F_k}(x_k, \boldsymbol{\sigma}_i^l) + \prod_{k=1}^{n} \mu^l_{F_k}(x_k, \boldsymbol{\sigma}_i^l)$$
$$\times \sum_{v=1}^{m}\left(\frac{\partial \mu^l_{F_v}(u_v, \boldsymbol{\delta}_i^l)}{\partial x_q}\prod_{\substack{j=1\\j\neq v}}^{m} \mu^l_{F_j}(u_j, \boldsymbol{\delta}_i^l)\right). \tag{48}$$

Finally, the calculation of $\partial w_i^l(\mathbf{u})/\partial x_q$ implies the derivative of the control vector, wherewith, writing again Eq. (15) with the extended state vector showed in (19):

$$u_j = \sum_{k=0}^{n} c_{kj}^t \tilde{x}_k. \tag{49}$$

Then

$$\frac{\partial u_j}{\partial x_q} = \frac{\partial(\sum_{k=0}^{n} c_{kj}^t \tilde{x}_k)}{\partial x_q} = c_{qj}^t + \sum_{k=0}^{n} \left(\frac{\partial c_{kj}^t}{\partial x_q} \tilde{x}_k \right). \tag{50}$$

Replacing now $\partial c_{ki}^t/\partial x_q$ from (38):

$$\frac{\partial u_j}{\partial x_q} = \frac{\partial(\sum_{k=0}^{n} c_{kj}^t \tilde{x}_k)}{\partial x_q} = c_{qj}^t + \sum_{k=0}^{n} \left(\frac{\sum_{r=1}^{N} \sum_{s=1}^{N} \left(\frac{\partial \omega_j^r}{\partial x_q} \omega_j^s (c_{kj}^r - c_{kj}^s) \right)}{(\sum_{r=1}^{N} \omega_j^r)^2} \tilde{x}_k \right). \tag{51}$$

3.5. Jacobian matrix

Every element of the Jacobian matrix, Eq. (24), is given by the derivative of the expression (25). Two addends of this expression have been calculated in Sections 3.1 and 3.2, and they are given by Eqs. (31) and (40), therefore,

$$\begin{aligned}\mathbf{J}_{iq}(\mathbf{x}) = \frac{\partial \xi_i(\mathbf{x})}{\partial x_q} = a_{qi}^t + \sum_{j=1}^{m} b_{ji}^t c_{qj}^t + \sum_{k=0}^{n} \Bigg\{ \Bigg[& \frac{\sum_{l=1}^{M} \sum_{p=1}^{M} \left(\frac{\partial w_i^l}{\partial x_q} w_i^p (a_{ki}^l - a_{ki}^p) \right)}{(\sum_{l=1}^{M} w_i^l)^2} \\ & + \sum_{j=1}^{m} \left(\frac{\sum_{l=1}^{M} \sum_{p=1}^{M} \left(\frac{\partial w_i^l}{\partial x_q} w_i^p (b_{ji}^l - b_{ji}^p) \right)}{(\sum_{l=1}^{M} w_i^l)^2} c_{kj}^t + b_{ji}^t \right. \\ & \left. \times \frac{\sum_{r=1}^{N} \sum_{s=1}^{N} \left(\frac{\partial \omega_j^r}{\partial x_q} \omega_j^s (c_{kj}^r - c_{kj}^s) \right)}{(\sum_{r=1}^{N} \omega_j^r)^2} \right) \Bigg] \tilde{x}_k \Bigg\}. \end{aligned} \tag{52}$$

Notice that Eqs. (43) and (48), and therefore Eq. (52), are independent on the type of membership function that is chosen to fulfill the fuzzy plant and controller models, so, it is possible to evaluate the Jacobian matrix for any membership function. Moreover, it is possible to combine different membership functions in a same model.

To implement the algorithm for calculating the Jacobian matrix is necessary to know the derivatives regarding state variables of the chosen membership functions. For example, if Gaussian membership functions are chosen for the fuzzy plant and controller models,

$$Gaussian(x : \gamma_{kj}^r, \beta_{kj}^r) = \exp\left[-\left(\frac{x - \gamma_{kj}^r}{\beta_{kj}^r} \right)^2 \right], \tag{53}$$

where γ_{kj}^r and β_{kj}^r are the center and width of the function, respectively, for kth state variable and the jth control signal. Then, Eqs. (43) and (48) can be written, respectively, as

$$\frac{\partial \omega_j^r}{\partial x_q} = -2\omega_j^r \left(\frac{x_q - \gamma_{qj}^r}{(\beta_{qj}^r)^2} \right), \tag{54}$$

and

$$\begin{aligned}\frac{\partial w_i^l}{\partial x_q} = -2w_i^l &\left\{ \left(\frac{x_q - \gamma_{qi}^l}{(\beta_{qi}^l)^2} \right) + \sum_{j=1}^{m} \left[\left(\frac{u_j - \gamma_{(j+n)i}^l}{(\beta_{(j+n)i}^l)^2} \right) \right.\right. \\ &\left.\left. \times \left(\frac{2\sum_{k=0}^{n} \left(\sum_{r=1}^{N} \sum_{s=1}^{N} \left(\omega_j^r \omega_j^s (c_{kj}^s - c_{kj}^r) \left(\frac{x_q - \gamma_{qj}^r}{(\beta_{qj}^r)^2} \right) \right) \tilde{x}_k \right)}{(\sum_{r=1}^{N} \omega_j^r)^2} + c_{qj}^t \right) \right] \right\}. \end{aligned} \tag{55}$$

4. Algorithms

Let the system defined by (1) and (2), whose fuzzy equivalent, considering TSK consequent with affine term, is given by (10), and the controller, considering equally TSK consequent with affine term, is given by (15); the fuzzy closed loop model is given by (20). For programming this model, given an equilibrium state, the algorithm shown in Fig. 1 is employed.

Given a concrete state $\mathbf{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$, the steps to compute the Jacobian matrix of the closed loop system are:

1. Calculate the matching degree of the rules of the fuzzy controller by (17).
2. Compute the variable coefficients of the fuzzy controller by second equation of (16), with $k = 0, 1, \ldots, n$.
3. Find the value of each one of the signals generated by the fuzzy controller by (15).
4. Calculate the matching degree of the rules of the fuzzy plant model by (8).
5. For the ith equation of the process and the qth state variable:
   5.1. Calculate the variable coefficient of the plant model by the second equation of (11) and by (12) with $k = 0, 1, \ldots, n$.

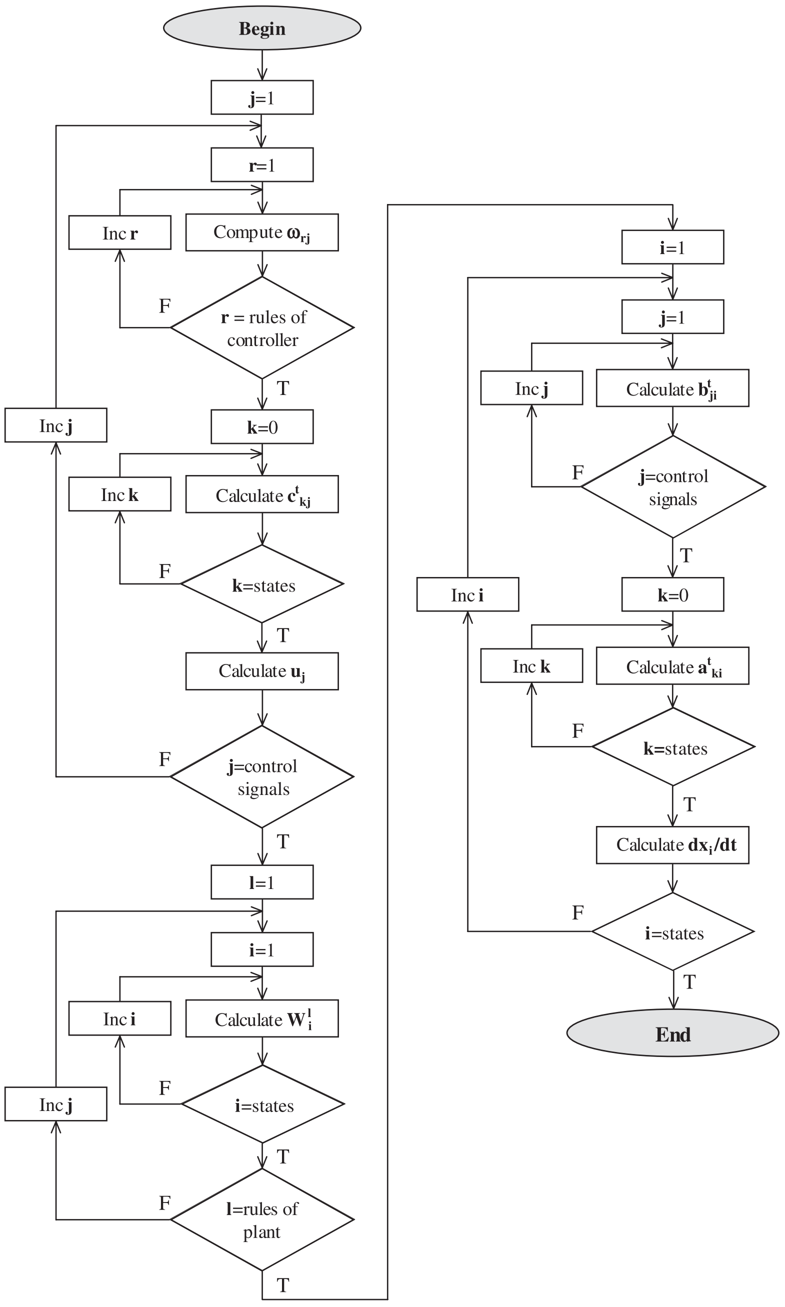


Fig. 1. Algorithm to compute Eq. (20) given an equilibrium state.

5.2. Compute the second addend of Eq. (52).
5.3. Calculate the partial derivative of the matching degree of the rules of the fuzzy controller model by (43).

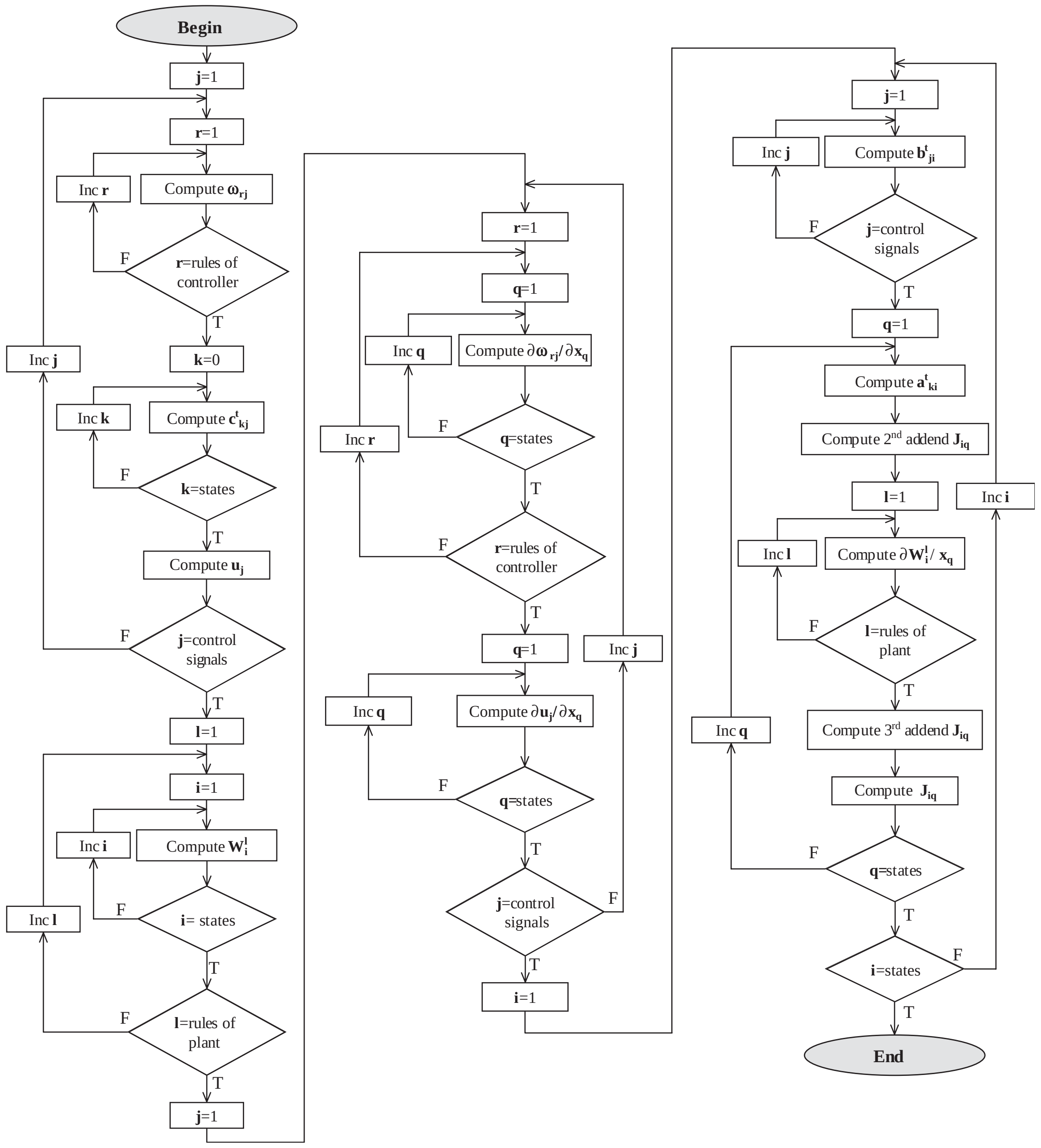


Fig. 2. Algorithm to compute Eq. (52).

5.4. Calculate the partial derivative of the matching degree of the rules of the fuzzy process model by (48). To complete this derivative, Eq. (51) is used.
5.5. Compute third addend of (52).

5.6. Calculate the corresponding term J_{iq} of the Jacobian matrix.

6. Repeat step 5 until all terms of the Jacobian matrix evaluated in the state $\mathbf{x}$ are computed (see Fig. 2).

As we reach this point, a general method is available for the calculation of the Jacobian matrix at a point of a fuzzy control system identified on an input/output data basis. Next, a software application will be developed, which allows us to implement and solve all the developed methodology.

5. Software design

For programming the developed algorithms in this paper, we have designed objects for saving the different parts of a fuzzy control system. For that, the Unified Modeling Language (UML) structure of classes of Fig. 3 has been implemented in C++, which can be compiled in any operating system. Moreover, a series of interfaces have been programmed which allow us to compile this software in MATLAB, integrating it completely into this environment and allowing greater possibilities for the Fuzzy Logic Toolbox.

The designed software allows the use of the most common membership functions: Bell; Gauss; Pi; Sigmoidal; Trapezoidal; Triangular; S and Z; and other variants as two Gaussian membership functions, one for each side (Gauss2mf); two Sigmoidal membership functions addend (Sigmoidal2mf) and finally, Sigmoidal membership functions product (Psigmoidalmf).

The designed software allows the use of all types of membership functions included in the MATLAB's Fuzzy Logic Toolbox. However, it is very easy to add any other membership function inheriting of the Membership class and extending enumeration TYPE_MF.

The NOMF membership type has been created to allow the reliable initialization of the rules of the fuzzy models. NOMF is a membership function without parameters. The evaluation of this function and its derivative is zero for any value of the input variable.

The designed software provides complementary functions for facilitating the reading and writing of fuzzy systems, as well as utilities not included in the MATLAB's Fuzzy Logic Toolbox. These developed complementary functions will be explained later, at the end of this section.

The "Membership'' class shown in Fig. 4 allows saving the parameters of a membership function included in the TYPE_MF enumeration.

This class provides the necessary methods for reading and writing the parameters of the defined membership function, as well as for modifying the type of the function that saves a concrete object.

Functions "eval'' and "evalder'' provide, respectively, the value of the saved membership function and its derivative on the x point.

The "Rule'' class (Fig. 5), a type of data, saves a complete rule base for the fuzzy plant (4) or the controller (13), therefore, it saves n or m IF–THEN rules.

Every object "Rule'' saves all the membership functions in the "membership'' matrix, and TSK consequents with affine term in the "TSK'' matrix.

Fig. 6 shows the "System'' class, a type of data. This class saves a complete fuzzy system, including as many rules as required. The numbers of rules are saved in the attribute "rules'', and every rule in a position of vector "R''.

Given the fuzzy model of a plant and its controller, the "Jacobian'' class allows to calculate and to save the Jacobian matrix of the formed control system (see Fig. 7).

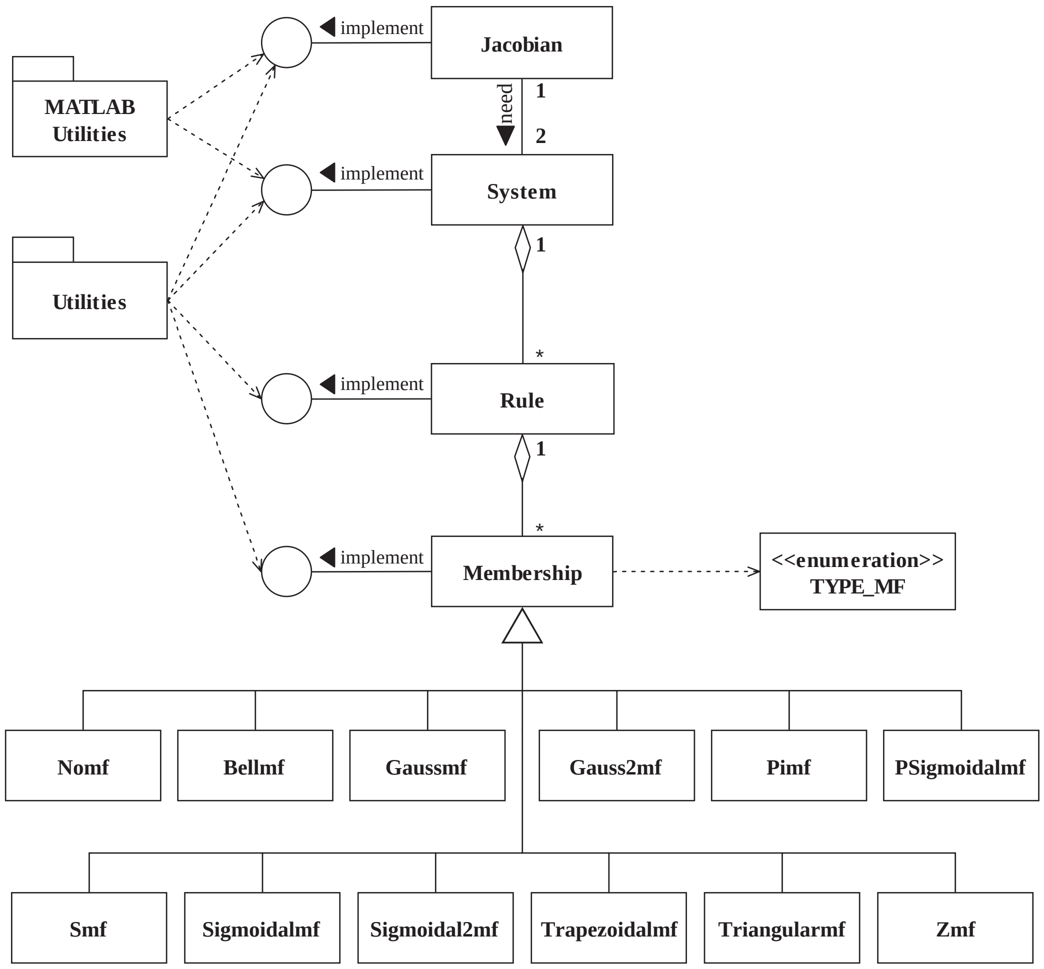


Fig. 3. UML diagram that shows the structure of classes.

The UML diagram of Fig. 3 shows two packages of utilities, which complement the designed software. “Utilities’’ (see Fig. 8) provides generic functions of reading/writing. These functions, based on the overload of the operators of insertion and extraction, make easy the reading and writing of the membership functions, even complete fuzzy systems, by using the standard input/output C++ flow.

The “Evaluate’’ method allows evaluating the output of a closed loop fuzzy control system by Eq. (25). This method uses the algorithm shown in Fig. 1 and explained in Section 4.

The “printSystem’’ function allows creating an ASCII text file. This file incorporates the linguistic representation of the fuzzy system given as argument, and it is possible to state the nouns of the input and output variables of the model.

Finally, within the “Utilities’’ package, “createMF’’ is a virtual constructor intended for building membership objects in execution time, namely, it allows building membership functions whose type is indicated at the beginning of the application.

Membership

# param : double*
# n: int
# type: TYPE_MF

+ eval (in x: double, out double)
+ evalder (in x: double, out double)
+ n_param (out int): const
+ type (outTYPE_MF): const
+ type (in type: TYPE_MF, out int)
+ edit (inindex: int, in value: double, out int)
+ read (inindex: int, out double): const
+ operator= (in P: const Membership&, out Membership&)
+ Membership (in P: const Membership&)
+ Membership ()
+ ~Membership ()

Fig. 4. “Membership’’ class.

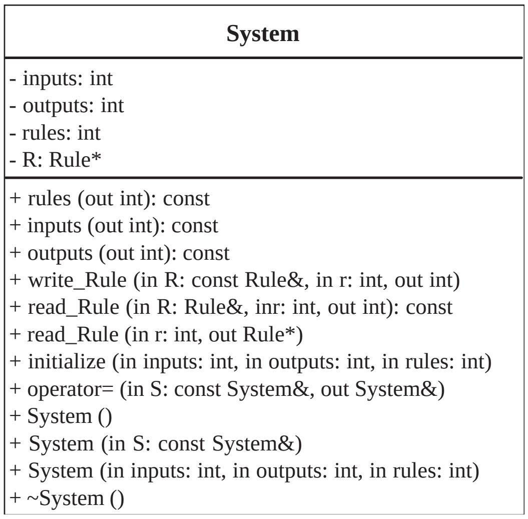
Rule

- inputs:int
- outputs:int
- TSK:double**
- membership: Membership***

+ inputs (out int): const
+ outputs (out int): const
+ write_TSK (in value: double, in input: int, in output: int, out int)
+ read_TSK (in input: int, in output: int, out double): const
+ read_function (in P: Membership&, in input: int, in output: int): const
+ read_function (in input: int, in output: int, out Membership*)
+ write_function (in P:const Membership&, in input: int, in output: int, out int)
+ change_function (in type: TYPE_MF, in input: int, in output:int, out int)
+ initialize (in input: int, in output: int)
+ operator= (in R: const Rule&, out Rule&)
+ Rule ()
+ Rule (in R: const Rule&)
+ Rule (in input: int, inoutput: int)
+ ~Rule ()

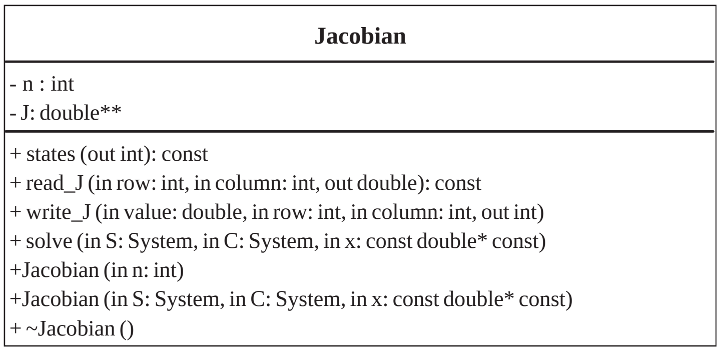
Fig. 5. “Rule’’ class.

Fig. 9 shows some interfaces designed for MATLAB. These utilities complement the previous packages serving as data “translators’’ between the MATLAB environment and the application compiled in C++. “FIS2System’’ and “System2FIS’’ functions, allow, respectively, writing and reading type FIS variables, which are the variables used by the MATLAB’s Fuzzy Logic Toolbox. The “readModel’’ function allows



System
- inputs: int - outputs: int - rules: int - R: Rule*
+ rules (out int): const + inputs (out int): const + outputs (out int): const + write_Rule (in R: const Rule&, in r: int, out int) + read_Rule (in R: Rule&, inr: int, out int): const + read_Rule (in r: int, out Rule*) + initialize (in inputs: int, in outputs: int, in rules: int) + operator= (in S: const System&, out System&) + System () + System (in S: const System&) + System (in inputs: int, in outputs: int, in rules: int) + ~System ()

Fig. 6. “System’’ class.



Jacobian
- n : int - J: double**
+ states (out int): const + read_J (in row: int, in column: int, out double): const + write_J (in value: double, in row: int, in column: int, out int) + solve (in S: System, in C: System, in x: const double* const) +Jacobian (in n: int) +Jacobian (in S: System, in C: System, in x: const double* const) + ~Jacobian ()

Fig. 7. “Jacobian’’ class.

reading a fuzzy model in three different formats: ASCII text files, FIS file and FIS defined variable in the MATLAB’s workspace.

In addition to the classes and packages explained so far, six applications (MEX-files) [22,23] have been programmed as interface between the C++ developed software and MATLAB. These functions are executable in the MATLAB’s environment in a transparent way for the user, but maintaining the C++ characteristic speed of calculation. The purpose of these functions is:

Fis2txt: Converts a FIS variable of the MATLAB’s Fuzzy Logic Toolbox in an ASCII text file. This function uses the “readModel’’ method included in the “MATLAB Utilities’’ Package for reading the fuzzy model from MATLAB’s environment or from a file. For writing, the output file uses the “System2TXT’’ method included in the “MATLAB Utilities’’ package.

Txt2FIS: Reads an ASCII text file with the definition of the fuzzy model and makes the corresponding MATLAB’s FIS variable. This function makes the reverse step of the previous function. For reading the fuzzy model from the text file, it uses the “TXT2System’’ method included in the “Utilities’’ package, and

```
Utilities

TXT2System (in file: const char*, in S: System&, out int)
System2TXT (in S: const System* const, in file: const char*, out int)
Evaluate (in x: const double* const, in dX: double*, in S: const System&, in C:const System&, out int)
printSystem (in file: const char*, in S: const System&, in inputs: char*=NULL, in outputs: char *=NULL, out int)
createMF (in type: TYPE_MF, Membership*)
operator<< (in F: ostream&, in P: Membership&, out ostream&)
operator>> (in F: istream&, in P: Membership&, out istream&)
operator<< (in F: ostream&, in R: Rule&, out ostream&)
operator>> (in F: istream&, in R:Rule&, out istream&)
operator<< (in F: ostream&, in S: System&, out ostream&)
operator>> (in F: istream&, in S: System&, out istream&)
operator<< (in F: ostream&, in J: Jacobian&, out ostream&)
operator>> (in F: istream&, in J: Jacobian&, out istream&)
```

Fig. 8. “Utilities’’ package.

```
MATLAB Utilities

FIS2System (in FIS: const mxArray*, in S: System&, out int)
System2FIS (in FIS: mxArray*, in S: const System&, out int)
readModel (inmodel: mxArray*, in S: System, out int)
```

Fig. 9. “MATLAB Utilities’’ package.

it makes, by the “System2FIS’’ function included in the “MATLAB Utilities’’ package, the corresponding variable in MATLAB’s workspace.

FuzPrint: Converts a fuzzy model in a text file with its linguistic representation. Optionally, it is possible to indicate the nouns of the input and output of the fuzzy model. By the “readModel’’ method, the system is read and written in a text file by using the “printSystem’’ function included in the “Utilities’’ package.

FuzComb: Combines several fuzzy models in a single model with several outputs, as long as the models have the same number of inputs and rules (most of the used programs for fuzzy systems modeling just allow identifying systems with an output only; therefore, it is necessary to obtain a model for each output). The resulting model can be saved in an ASCII file or in the MATLAB’s workspace.

FuzEval: Given an x point of the state space, this function calculates the closed loop fuzzy system output in this point. This function uses the “Evaluate’’ method included in the “Utilities’’ package, which is the result of the programming algorithm show in Fig. 1 for computing Eq. (20). This function is similar to “evalfis’’ function included in the MATLAB’s fuzzy logic toolbox, except that “evalfis’’ calculates a fuzzy model and “FuzEval’’ calculates a closed loop fuzzy system formed by the plant fuzzy model and the controller fuzzy model.

FuzJac: Calculates the Jacobian matrix of the closed loop fuzzy system at a given point. This function uses the “solve’’ method included in the “Jacobian’’ class, which is the result of the programming algorithm showed in Fig. 2 for computing Eq. (52).

6. Example

Let the following plant given by its internal representation be:

$$\dot{x}_1 = x_1 + 2x_2 + u_1,$$
$$\dot{x}_2 = x_1 - 2x_2^3 + u_2. \tag{56}$$

This plant has an equilibrium point at the origin of the state space. Linearizing the nonlinear model given by (56) around this point, the following system is obtained:

$$\dot{x}_1 = x_1 + 2x_2 + u_1,$$
$$\dot{x}_2 = x_1 + u_2, \tag{57}$$

whose dynamic matrix is

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 1 & 0 \end{pmatrix}.$$

Eigenvalues of the matrix **A** are: $\lambda_1 = -1$ and $\lambda_2 = 2$, so the system (56) is unstable, and the origin of the state space is a saddle point for this system.

For the implementation of this example, the plant dynamics is supposed to be unknown, so that Eq. (56) is only used to obtain input/output data of the plant.

The fuzzy model of the plant has been obtained by the methodology based on descending gradient developed in [1] (see Appendix). The input/output data of the plant has been obtained by a random number generator in the interval $(x_1, x_2) \in [-10, 10]$.

The model's membership functions are Gaussian type defined by expression (53), and the consequents are TSK with affine term.

Now, once the plant dynamics is known, it is possible to design a controller that stabilizes the system around the equilibrium point in the origin of the state space. A possible controller could be:

IF x1 is GAUSSMF(−9.77,3.08) and x2 is GAUSSMF(−9.05,13.7) THEN u1 = −4.66*x1−0.00438*x2
IF x1 is GAUSSMF(−8.34,14.4) and x2 is GAUSSMF(6.71,4.07) THEN u1 = 0.0002−5.69*x1
IF x1 is GAUSSMF(12.6,9.16) and x2 is GAUSSMF(−13,11) THEN u1 = 0.00047−3.01*x1+0.00385*x2
IF x1 is GAUSSMF(7.05,11.4) and x2 is GAUSSMF(15.1,10.3) THEN u1 = 0.00044−2.62*x1+0.00155*x2
IF x1 is GAUSSMF(−6.01,16.1) and x2 is GAUSSMF(−9.7,13.9) THEN u2 = 0.0004+1.83*x1−8.42*x2
IF x1 is GAUSSMF(−13.5,10.6) and x2 is GAUSSMF(12.4,17.6) THEN u2 = 0.000319+0.34*x1−5.21*x2
IF x1 is GAUSSMF(3.88,12) and x2 is GAUSSMF(−15.6,15.4) THEN u2 = 1.32*x1−9.16*x2
IF x1 is GAUSSMF(18.2,9.17) and x2 is GAUSSMF(14.8,18.3) THEN u2 = −0.000504+2.06*x1−2.81*x2

The parameters of the Gaussian membership functions (GAUSSMF) are the center and width, respectively.

In order to prove the developed algorithm, the Jacobian matrix of the closed loop fuzzy system evaluated in the equilibrium state by the "FuzJac'' function is

$$\mathbf{J} = \begin{bmatrix} -2.589 & 1.984 \\ 2.302 & -7.945 \end{bmatrix}.$$

To determine the control system stability by the application of Krasovskii's theorem, it is necessary to prove that $\mathbf{J}(\mathbf{x})^{\mathrm{T}} + \mathbf{J}(\mathbf{x})$ is negative definite in an environment of the equilibrium state. Evaluating this

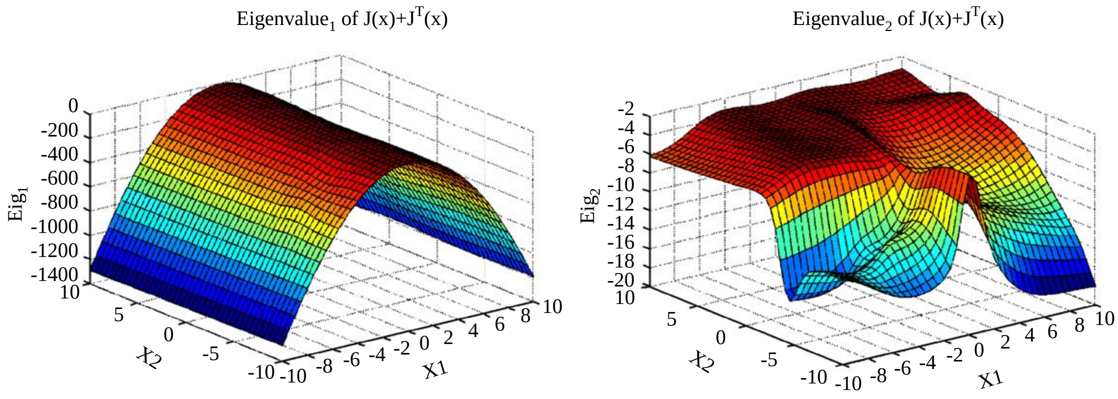


Fig. 10. Evolution of the Eigenvalues of the matrix $\mathbf{J}(\mathbf{x})^{\mathrm{T}} + \mathbf{J}(\mathbf{x})$ in the environment $(x_1, x_2) \in [-10, 10]$.

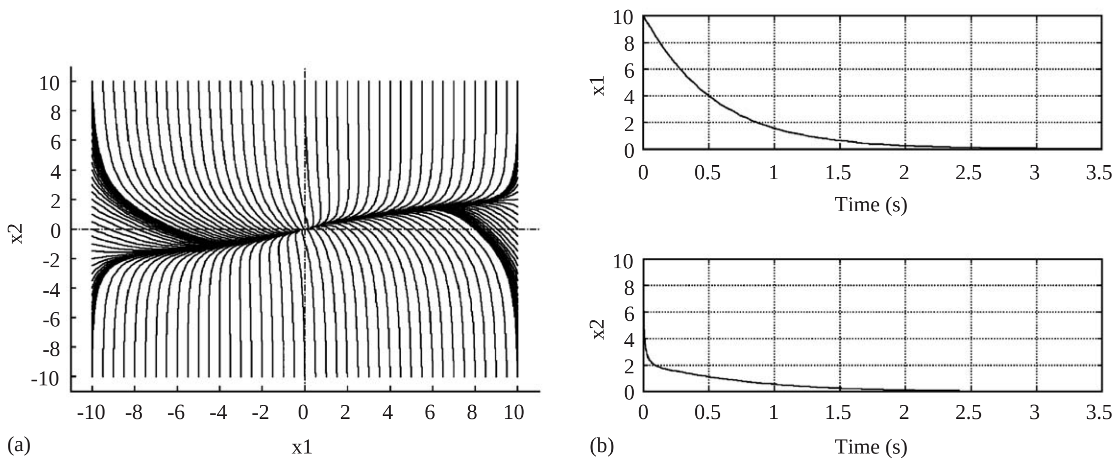


Fig. 11. (a) Phase portrait. (b) Simulation shows the stability of the fuzzy closed loop control system.

sum in $(x_1, x_2) \in [-10, 10]$, Fig. 10 shows the evolution of the Eigenvalues. Note that maximum values are $\lambda_{1,\max} = -9.62$ and $\lambda_{2,\max} = -3.01$, so Krasovskii's theorem is accomplished, and the fuzzy closed loop control system is asymptotically stable in the studied environment.

Finally, a phase portrait for this system is shown in Fig. 11(a). Observe how all paths converge on the equilibrium state. Fig. 11(b) shows a simulation of the fuzzy closed loop control system starting from the initial state $x_1 = 10,\ \ x_2 = 10$. Note the stability.

In order to prove the functionality of the proposed method, with small errors inherent to the plant modeling present, simulations of the system have been carried out applying control signals generated by the fuzzy controller to the actual plant given by (56) instead of the model. To do this, the MATLAB's function “ode45’’ and the “FuzEval’’ function included in the designed software have been used.

7. Conclusions

In this paper, an equivalent mathematical closed loop model of a multivariable nonlinear control system based on fuzzy logic theory has been developed. Later, this model was used to compute the Jacobian matrix of a closed loop fuzzy system. Next, an algorithm to solve the Jacobian matrix was proposed, and finally, a software implementing the improved algorithm and an interface with MATLAB making its use easier under this environment have been developed. Also, a series of additional applications for its use have been presented (the designed software implements more than 3500 lines of C++ code). The proposed methodology has been illustrated by an example.

The main goal of this paper is to solve the Jacobian matrix of a completely general fuzzy control system, without constraints neither in the order of the state vector nor in the control one. The algorithm is independent also of the type of membership used.

The solution of the Jacobian matrix developed in this paper, opens the possibility of analyzing and designing control fuzzy systems with assured asymptotical stability. For this purpose, it proposes to use the Krasovskii's theorem, which demonstrates that if the matrix $\mathbf{J}(\mathbf{x}) + \mathbf{J}(\mathbf{x})^{\mathrm{T}}$ is negative definite in a region around the equilibrium state $\mathbf{x}^{e}$, this is asymptotically stable in its proximities. Moreover, if $\boldsymbol{\xi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\xi}(\mathbf{x}) \rightarrow \infty$ when $\|\mathbf{x}\| \rightarrow \infty$, the asymptotical stability is global. (In fact, if we know the plant only by its fuzzy model, the global stability condition can never be assured. However, we can guarantee stability in the space defined by the universe of discourse of the plant fuzzy model state variables, as in the example in this paper.)

The methodology developed in this work is somehow robust with respect to the plant modeling errors: what is important is not the value of $J(x)$ itself (which will be affected by the modeling error at the point of calculation), but the sign of the eigenvalues of $\mathbf{J}(\mathbf{x}) + \mathbf{J}^{\mathrm{T}}(\mathbf{x})$.

The complete goal in this paper is only a step within a wider objective for us: The synthesis of control fuzzy systems stable by design.

From the point of view of the complete control problem that we are trying to solve, two very important questions are still pending:

(1) To find the equilibrium points of the closed loop fuzzy system given by Eq. (20). This problem, at the time of writing this paper, is practically solved, and in a next work coming very soon, we will present at least two methods to find the equilibrium points of (20).

(2) To adjust the controller's parameters so that the closed loop fuzzy control system fulfils Krasovskii's theorem. Now, we are developing algorithms to do this, and we hope to be able to present good results soon.

The two previous questions, still not solved for us, cause the shown example to be of analysis rather than of synthesis, as it is our final goal. Anyway, the possibility is left open to synthesize, by means of Krasovskii's theorem, fuzzy control systems that assure global asymptotic stability.

Finally, in oder for the interested readers to test the developed software (comments and criticism are welcome), we provide the following urls for download:

- Functions written in C++, regardless of the environment and operating system in which they are compiled: Download file "Fuzzy Logic Toolbox Add-on'' from research section of http://www.uhu.es/antonio.barragan/modules/mydownloads.
- The same aforementioned functions prepared to be used from MATLAB: Download file "Fuzzy Control Systems Add-On'' from http://www.mathworks.com/matlabcentral/fileexchange/ Section: Controls and Systems Modeling+Neural and Fuzzy Systems.

Appendix. Identification of the plant

The algorithm that we have used to identify the plant is a way of taking advantage of the learning ability of neuronal networks for the constructions of fuzzy models. In particular, we have applied one of the most important neural network learning algorithms, known as the Backpropagation (BP) algorithm, to the parameter estimation of the plant. This way, the identification of the plant is neurofuzzy. The algorithm is explained in [1]. The fuzzy model of the plant is the following:

IF x1 is GAUSSMF(− 14.1,13.8) and x2 is GAUSSMF(− 13.4,13.5) and u1 is GAUSSMF(− 13.8,14.7) and u2 is GAUSSMF (− 13.1,14.6) THEN dx1 = − 16.7+1.93*x1+1.26*x2− 0.114*u1−0.235*u2

IF x1 is GAUSSMF(− 11.2,14) and x2 is GAUSSMF(− 11.6,13.7) and u1 is GAUSSMF(− 10.6,14.3) and u2 is GAUSSMF (− 10.8,14.6) THEN dx1 = − 12.8− 0.339*x1+2.44*x2+1.47*u1− 0.0821*u2

IF x1 is GAUSSMF(− 6.01,15.1) and x2 is GAUSSMF(− 7.05,14.1) and u1 is GAUSSMF(− 6.85,14) and u2 is GAUSSMF (− 5.24,15) THEN dx1 = − 13.5+1.13*x1+1.01*x2+1.57*u1− 1.13*u2

IF x1 is GAUSSMF(− 4.68,15.1) and x2 is GAUSSMF(− 5.54,14.1) and u1 is GAUSSMF(− 5.69,13.9) and u2 is GAUSSMF (− 4.06,15) THEN dx1 = − 8.27+0.232*x1+1.99*x2− 0.507*u1+0.506*u2

IF x1 is GAUSSMF(− 0.68,13.5) and x2 is GAUSSMF(− 3.67,12.4) and u1 is GAUSSMF(− 1.62,13.9) and u2 is GAUSSMF (− 2.59,14.1) THEN dx1 = − 11.5+0.617*x1+1.36*x2+0.423*u1− 0.498*u2

IF x1 is GAUSSMF(2.15,14.2) and x2 is GAUSSMF(3.85,13.6) and u1 is GAUSSMF(2.6,13.7) and u2 is GAUSSMF(1.48,14.7) THEN dx1 = 5.29+1.29*x1+1.45*x2+0.78*u1+0.842*u2

IF x1 is GAUSSMF(2.09,14.3) and x2 is GAUSSMF(3.71,13.9) and u1 is GAUSSMF(2.21,13.8) and u2 is GAUSSMF(2.55,15) THEN dx1 = 8.35+0.0477*x1+0.536*x2+0.459*u1− 2.2*u2

IF x1 is GAUSSMF(7.54,14.8) and x2 is GAUSSMF(7.69,14.3) and u1 is GAUSSMF(7.1,14.1) and u2 is GAUSSMF(7.09,15.1) THEN dx1 = 17.6− 0.933*x1+2.02*x2− 0.4*u1+0.421*u2

IF x1 is GAUSSMF(8.74,15.1) and x2 is GAUSSMF(9.1,14.6) and u1 is GAUSSMF(9.24,13.7) and u2 is GAUSSMF(8.27,15) THEN dx1 = 18.5+2.2*x1+2.05*x2+0.917*u1− 0.894*u2

IF x1 is GAUSSMF(11.5,14.3) and x2 is GAUSSMF(12.5,14.5) and u1 is GAUSSMF(13.1,14.7) and u2 is GAUSSMF(11.7,15.9) THEN dx1 = 11.1+0.551*x1+1.37*x2+1.26*u1+0.0346*u2

IF x1 is GAUSSMF(− 18.6,9.25) and x2 is GAUSSMF(− 14.7,12) and u1 is GAUSSMF(− 17.2,11.1) and u2 is GAUSSMF (− 22.6,3.59) THEN dx2 = 0.473− 3.82*x1− 4.11*x2− 2.74*u1− 4.7*u2

IF x1 is GAUSSMF(− 18.5,8.88) and x2 is GAUSSMF(− 5.4,4.49) and u1 is GAUSSMF(− 16.7,10.2) and u2 is GAUSSMF (− 19.8,8.46) THEN dx2 = 5.91e+003+130*x1− 68*x2+222*u1+244*u2

IF x1 is GAUSSMF(− 7.26,17.2) and x2 is GAUSSMF(− 17.6,12.7) and u1 is GAUSSMF(− 6.32,16.3) and u2 is GAUSSMF (− 7.28,17.4) THEN dx2 = 3.2e+003+11*x1− 698*x2− 34.5*u1+14.8*u2

IF x1 is GAUSSMF(− 6.82,17.2) and x2 is GAUSSMF(− 9.74,7.29) and u1 is GAUSSMF(− 6.48,16.3) and u2 is GAUSSMF (− 6.78,17.4) THEN dx2 = − 3.04e+003+10.8*x1+52.7*x2+15.3*u1+10.3*u2

IF x1 is GAUSSMF(1.05,17.5) and x2 is GAUSSMF(1.84,10.7) and u1 is GAUSSMF(1.2,17.8) and u2 is GAUSSMF(1.12,17.4) THEN dx2 = − 5.62e+003− 100*x1+1.06e+003*x2− 37.3*u1− 109*u2

IF x1 is GAUSSMF(1.21,17.5) and x2 is GAUSSMF(7.63,21.2) and u1 is GAUSSMF(1.24,17.8) and u2 is GAUSSMF(1.29,17.4) THEN dx2 = − 449− 15*x1− 875*x2− 7.84*u1− 17.1*u2

IF x1 is GAUSSMF(1.28,17.5) and x2 is GAUSSMF(3.07,11.5) and u1 is GAUSSMF(1.37,17.8) and u2 is GAUSSMF(1.37,17.4) THEN dx2 = 6.36e+003+110*x1− 370*x2+39.2*u1+121*u2

IF x1 is GAUSSMF(12.1,11.8) and x2 is GAUSSMF(9.97,5.66) and u1 is GAUSSMF(10.7,12.4) and u2 is GAUSSMF(12.4,11.5) THEN dx2 = 2.13e+003+27.7*x1− 168*x2+32*u1+30.4*u2

IF x1 is GAUSSMF(13.6,12) and x2 is GAUSSMF(18.4,10.7) and u1 is GAUSSMF(11.9,12.7) and u2 is GAUSSMF(13.6,11.6) THEN dx2 = − 5.76e+003+88*x1− 515*x2+19.2*u1+62.4*u2

IF x1 is GAUSSMF(17.9,11.2) and x2 is GAUSSMF(15,10.9) and u1 is GAUSSMF(20.2,9.48) and u2 is GAUSSMF(19.5,10.8) THEN dx2 = 945− 122*x1− 292*x2+199*u1− 106*u2

The parameters of the Gaussian membership functions (GAUSSMF) are the center and width, respectively.

References

[1] J.M. Andújar, J.M. Bravo, Multivariable fuzzy control applied to the physical–chemical treatment facility of a cellulose factory, Fuzzy Sets and Systems 150 (3) (2005) 475–492.

[2] J.M. Andújar, J.M. Bravo, A. Peregrín, Stability analysis and synthesis of multivariable fuzzy systems using interval arithmetic, Fuzzy Sets and Systems 148 (3) (2004) 337–353.

[3] J. Aracil, A. García, A. Ollero, Fuzzy control of dynamical systems, Stability analysis based on the conicity criterion, in: Proc. Fourth Internat. Fuzzy Systems Association Congress, Brussels, Belgium, 1991, pp. 5–8.

[4] J. Aracil, F. Gordillo, Stability issues in fuzzy control, Physica-Verlag, Heidelberg, New York, 2000.

[5] J. Aracil, A. Ollero, A. García, Stability indices of the global analysis of expert control systems, IEEE Trans. Systems Man Cybernet. 19 (5) (1987) 998–1007.

[6] R. Babuška, Fuzzy modeling a control engineering perspective, in: Proc. FUZZ-IEEE/IFES’95, Yokohama, Japan, 1995, pp. 1897–1902.

[7] R. Babuška, H.B. Verbruggen, A new identification method for linguistic fuzzy models, in: Proc. FUZZ-IEEE/ IFES’95, Yokohama, Japan, 1995, pp. 905–912.

[8] J.M. Bravo, O. Sánchez, J.M. Andújar, E. Fernández, Stability analysis and synthesis of fuzzy systems using intervalar arithmetic, in: Proc. 15th IFAC World Congress, Barcelona, Spain, 2002.

[9] D. Driankov, H. Hellendoor, M. Reinfrank, An Introduction to Fuzzy Control, Springer, Berlin, 1993.

[10] G. Feng, S.G. Cao, N.W. Rees, Stable adaptive control of fuzzy dynamics systems, Fuzzy Sets and Systems 131 (2002) 217–224.

[11] C.C. Fuh, P.C. Tung, Robust stability analysis of fuzzy control systems, Fuzzy Sets and Systems 88 (1997) 289–298.

[12] T. Glad, L. Ljung, Control Theory: Multivariable and Nonlinear Methods, Taylor & Francis, London, 2000.

[13] T. Hung, M. Sugeno, R. Tong, R.R. Yager, Theoretical Aspects of Fuzzy Control, Wiley, New York, 1995.

[14] A. Isidori, Nonlinear Control Systems II, Springer, London, 1999.

[15] M. Johanson, A. Rantzer, K.-E. Arzen, Picewise quadratic stability for Affine Sugeno systems, in: Proc. Fuzz-IEEE’98, Anchorage, USA, 1998.

[16] H.K. Khalil, Nonlinear Systems, Prentice-Hall, Englewood Cliffs, NJ, 2000.

[17] Y. Li, Y. Yonezawa, Stability analysis of a fuzzy control system by the hyperstability theorem, Japan J. Fuzzy Theory and Systems 3 (2) (1991) 209–214.

[18] H.R. Lin, W.J. Wang, L_2-stability analysis of fuzzy control systems, Fuzzy Sets and Systems 100 (1998) 159–172.

[19] M. Maeda, S. Murakami, Stability analysis of fuzzy control systems using phase planes, Japan J. Fuzzy Theory and Systems 3 (2) (1991) 149–160.

[20] H.A. Malki, H. Li, G. Chen, New design and stability analysis of fuzzy proportional-derivative control systems, IEEE Trans. Fuzzy Systems 2 (4) (1994) 245–254.

[21] J.P. Marin, A. Titli, Necessary and sufficient conditions for quadratic stability of a class of Takagi–Sugeno fuzzy systems, in: Proc. EUFIT’95 (2), Aachen, Germany, 1995, pp. 786–790.

[22] MATLAB External Interfaces, The MathWorks, Inc., 2002.

[23] MATLAB External Interfaces Reference, The MathWorks, Inc., 2002.

[24] T. Nguyen, M. Sugeno, R. Tong, R.R. Yager, Theoretical Aspects of Fuzzy Control, Wiley, New York, 1995.

[25] A. Piegat, Hyperstability of fuzzy-control systems and degrees of freedom, in: Proc. EUFIT’97 (2), Aachen, Germany, 1997, pp. 1446–1450.

[26] A. Piegat, Fuzzy Modeling and Control, Physica-Verlag, Heidelberg, 2001.

[27] V.M. Popov, The solution of a new stability problem for controlled systems, Automat. Remot. Control 24 (1963) 1–23.

[28] V.M. Popov, Hyperstability of Control Systems, Springer, Berlin, 1973.
[29] S. Sastry, Nonlinear System: Analysis, Stability, and Control, Springer, New York, 1999.
[30] T. Sheel, H. Kiendl, Stability analysis of fuzzy and other nonlinear systems using integral Lyapunov functions, in: Proc. EUFIT'95, Aachen, Germany, 1995, pp. 786–790.
[31] C.I. Siettos, G.V. Bafas, Semiglobal stabilization of nonlinear systems using fuzzy control and singular perturbation methods, Fuzzy Sets and Systems 129 (2002) 275–294.
[32] S. Singh, Stability analysis of discrete fuzzy control systems, in: Proc. First IEEE Conf. on Fuzzy Systems, 1992, pp. 527–534.
[33] J.-J.E. Slotine, W. Li, Applied Nonlinear Control, Prentice-Hall, Englewood Cliffs, NJ, 1991.
[34] Q. Sun, R. Li, P. Zhang, Stable and optimal adaptive fuzzy control of complex systems using fuzzy dynamic model, Fuzzy Sets and Systems 133 (2003) 1–17.
[35] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, IEEE Trans. Systems Man Cybernet. 15 (1985) 116–132.
[36] K. Tanaka, Stability and stabilizability of fuzzy-neural-linear control systems, IEEE Trans. Fuzzy Systems 3 (4) (1995) 438–447.
[37] K. Tanaka, M. Sugeno, Stability analysis of fuzzy systems using Liapunov's direct method, in: Proc. NAFIPS'90 (1), Toronto, USA, 1990, pp. 133–136.
[38] K. Tanaka, M. Sugeno, Stability analysis and design of fuzzy control systems, Fuzzy Sets and Systems 45 (1992) 135–156.
[39] S. Tong, H.H. Li, Observer-based robust fuzzy control of nonlinear system with parametric uncertainties, Fuzzy Set and Systems 131 (2002) 165–184.
[40] L.X. Wang, Adaptive Fuzzy Systems and Control, Prentice-Hall, Englewood Cliffs, NJ, 1994.
[41] L.X. Wang, A Course in Fuzzy Systems and Control, Prentice-Hall, New Jersey, USA, 1997.
[42] H.O. Wang, K. Tanaka, M. Griffin, An approach to fuzzy control of nonlinear system: stability and design issues, IEEE Trans. Fuzzy Systems 4 (1) (1996) 14–23.
[43] L. Wong, F. Leung, P. Tam, Stability design of TS model based fuzzy systems, in: Proc. Sixth IEEE Internat. Conf. on Fuzzy Systems, Barcelona, Spain, 1997, pp. 83–86.