International Journal of Innovative
Computing, Information and Control
Volume **12**, Number **5**, October **2016**

ICIC International ©2016 ISSN 1349-4198
pp. 1721–1743

MODELING AUTONOMOUS NONLINEAR DYNAMIC SYSTEMS USING MEAN DERIVATIVES, FUZZY LOGIC AND GENETIC ALGORITHMS

Mateus Oliveira de Figueiredo, Paulo Marcelo Tasinaffo
and Luiz Alberto Vieira Dias

Computer Science Division
Technological Institute of Aeronautics
Praça Marechal Eduardo Gomes, 50 Vila das Acácias, São José dos Campos/SP 12228-900, Brazil
mtsodf@gmail.com; tasinaffo@ita.br; l_vdias@yahoo.com.br

Received November 2015; revised August 2016

Abstract. *Any mathematical function with a universal approximator of function's feature can be used in the modeling of autonomous nonlinear dynamical systems. Such modeling can always be numerically solved when seen or interpreted as a nonlinear parameters estimation problem. Among the best-known universal approximators we can mention these: artificial neural networks and polynomials of n dimension. Nevertheless, there are other universal approximators of this type such as fuzzy inference systems and support vector machines. Therefore, what it is intended in this article is to use the Mamdani type fuzzy inference system with triangular pertinence functions and defuzzification by the calculation of the center of gravity (centroids) in order to represent nonlinear autonomous dynamic systems through the direct methodology of mean derivatives. The IF-THEN rules of the fuzzy system are automatically obtained by means of a supervised learning by using genetic algorithms. Three case studies of linear dynamic systems and nonlinear are presented to validate this methodology.*

Keywords: Fuzzy systems, Genetic algorithms, Universal approximators of functions, Autonomous nonlinear dynamic systems, Fuzzy modeling for dynamic systems, Direct mean derivatives methodology

1. **Introduction.** The modeling of dynamic systems is very important for the development of mathematical tools (see [2,3]) that can be used for a better understanding of phenomenon that occur in nature. In this sense, the nonlinear systems usually are those which present a better representation for real systems. For being nonlinear systems, they present a greater mathematical complexity in its modeling.

Artificial neural networks have been much used in the modeling of nonlinear dynamic systems in the last decades, because they have a high capacity of approximating nonlinear mappings. Several studies were developed in this area by utilizing NARMAX (*Nonlinear AutoRegressive Moving Average with eXogenous inputs*) methodology with latter application in control (e.g., [1,7,10,11,14]). In addition to this, other methodologies were also developed in the latest years like: instantaneous derivatives methodology (e.g., [9,12,17]) and mean derivatives methodology (e.g., [14-16,31]).

Alternatively, *Fuzzy* systems have been much used in control, prediction and pattern classification. The main feature of *Fuzzy* systems is that they are built from *Fuzzy* Logic and can accurately deal with vague information. Another important feature is that *Fuzzy* systems, similarly to neural networks, are universal approximators. This is the main motivation for this work, because from then on it is theoretically possible to substitute artificial neural networks by *Fuzzy* systems in applications. There are several meaningful

works involving *Fuzzy* Logic as developed in [18-20,26-30]. Continuing this work, this article intends to develop a methodology by using *Fuzzy* Logic and mean derivatives functions for modeling of nonlinear dynamic systems. A Mamdani-type *Fuzzy* inference system (see [27]) is specifically used to learn the mean derivatives functions.

It is important to note that, in this paper, the mean derivatives methodology is tested by using an inference *Fuzzy* system in a different way of the one that occurs in [15,16], which uses the mean derivatives methodology with artificial neural networks. The methodology presented here is an alternative method to NARMAX methodology, which can be found and described in detail in [7,10].

In addition to this, the mean derivatives methodology is also an alternative method to the ones described in [9,12,17], which are methods that use instantaneous derivatives instead of mean derivatives. Three linear and nonlinear case studies are proposed to validate this methodology by using mean derivatives in a Mamdani-type inference *Fuzzy* system. The *Fuzzy* system learns the mean derivatives functions by means of a supervised learning, where the IF-THEN rules of *Fuzzy* system are automatically obtained through the resolution of an optimization problem that uses a genetic algorithm. Thus, the IF-THEN rules of the *Fuzzy* system considered do not need to be established by a human expert. A mathematical and rigorous detail of mean derivatives method can be found in [31]. Nevertheless, Algorithm 1 presented in Section 4 of this article briefly describes the mean derivatives method.

This paper is organized into six sections. After introduction, there is a brief explanation on genetic algorithms in Section 2. In Section 3, *Fuzzy* systems are described. Section 4 describes the two algorithms proposed in this article in algorithmic notation. They are computationally compared in Section 5. Section 5 also refers to numerical outcomes for three distinct case studies. General appreciation and conclusions are presented in Section 6. The whole implementation will be done in Matlab software.

2. Genetic Algorithms.

An *evolutionary algorithm* can be defined as a search iterative procedure inspired on biological evolutionary devices and can be directly related to system optimization in engineering (e.g., [24,25]). *Evolutionary Computing* is the expression used to name the line of research that handles with *evolutionary algorithms*. The main goal of computing intelligence and therefore, of the evolutionary algorithms is to determinate solutions to complex optimization problems. For example, evolutionary algorithms can be applied to solving the following engineering problems: the traveling salesman, robotics (a safe feasible way and collision-free which should be traversed by a robot), control theory, economy, image processing and data mining problems, etc.

Based on this information, several evolutionary algorithms were proposed to solve common numerical optimization problems in engineering and numerical analysis. Among them, genetic algorithms, evolutionary strategies, evolutionary programming, genetic programming and classifier systems can be mentioned. All these algorithms together form a field called Evolutionary Computing.

The basic concepts involved in genetic algorithms are relatively simple. Initially, they originate from an initial chromosome population (e.g., ones and zeros strings or bits) which evolve to a new population exclusively utilizing a kind of "natural selection" together with mathematical operators inspired by genetics like selection, *crossover* and mutation operators (see [25]). Each chromosome consists of "genes" (e.g., bits) and each gene is an instance of a particular allele (e.g., zero or one). The selection operator probabilistically chooses those chromosomes that will be allowed to reproduce in a given population and thus, on average, the more adapted chromosomes will produce more offspring than those less adapted [25]. The *crossover* will be responsible for exchanging subparts of

two chromosomes, clumsily mimicking the biological recombination between two simple chromosomes (*single-chromosome* or *haploid*) deriving from two different organisms. The mutation randomly varies the allele's values in some specific localization of the chromosome.

When we watch the higher level "evolutionary rules", they are extremely simple: species involved in an environment which suffer random changes (through mutation, recombination end other evolutionary operators) that are followed by a natural selection process in which the fittest specie tends to survive and reproduces itself, thus spreading its genetic material to future generations. Therefore, by applying these evolutionary concepts to computational engineering, it is possible to propose a basic evolutionary algorithm or a pattern with the following features:

a) a population of applicants for solution (individuals with a genetic inheritance that are able to exchange the genetic substance – *crossover*);

b) genetic variability and/or genetic mutation;

c) natural selection: individuals assessment in their environment by means of an assessment or *fitness* function.

When all the previous steps: reproduction, genetic variability and selection had been performed, one can say that a generation has occurred. In each iteration P is a population containing N individuals $P = \{x_1, x_2, \ldots, x_N\}$ (data structure). F_i, $i = 1, 2, \ldots, N$, is the *fitness* function value of each population individual, and pc and pm respectively are the probabilities of happening *crossover* and *mutation*. Thus, it is possible to propose a pattern evolutionary algorithm like this: codification of variables (usually in binary notation), creation of an initial population, assessment of response, crossing, natural selection, crossover and mutation. The stopping criterion used is usually a stable number of traversed generations or a determination of a satisfactory solution.

In evolutionary computation, the concept of *Evolution* is a solution search method (in parallel) in a wide range of applicants submitted to variable conditions of adaptation. The rules of evolution are simple enough: species randomly evolve being submitted to natural selection before limited resources. The more adapted individuals survive and reproduce themselves, by spreading their genetic substance to the following generations. From this perspective, the evolutionary algorithms are numerical methods of optimization that involve the creation of random numbers. Nevertheless, these methods are not purely random, because genetic algorithms stand out in relation to those first, by requiring the exploration of only a small fraction of the possible space for solutions. Classic genetic algorithms also deal with: a stable size population; data structure of binary chain kind; natural selection proportional to *fitness* via *Roulette Wheel* algorithm; simple *crossover* and point mutation.

3. **Fuzzy Logic.** The main feature of *Fuzzy* Logic comes from its ability to infer conclusions and create responses from vague, ambiguous and qualitatively incomplete and inaccurate information. *Fuzzy* Logic is based on *Fuzzy* sets theory. *Fuzzy* Logic can be considered a generalization of the theory of traditional sets to solve the paradoxes created out of the usual "true or false" arrangement of Classical Logic.

Human communication involves natural words that are often vague, imprecise, uncertain and ambiguous. This way of communication is named natural language. *Fuzzy* Logic is based on words and not on numbers, that is, values are linguistically expressed, for example, hot, warm, fresh, and cold. In classical logic systems there are existential quantifiers ($\exists$) and universal ($\forall$) only. In addition to those, *Fuzzy* Logic [20] acknowledges a wide diversity of quantifiers (e.g., few, several, frequently, usually, and around five).

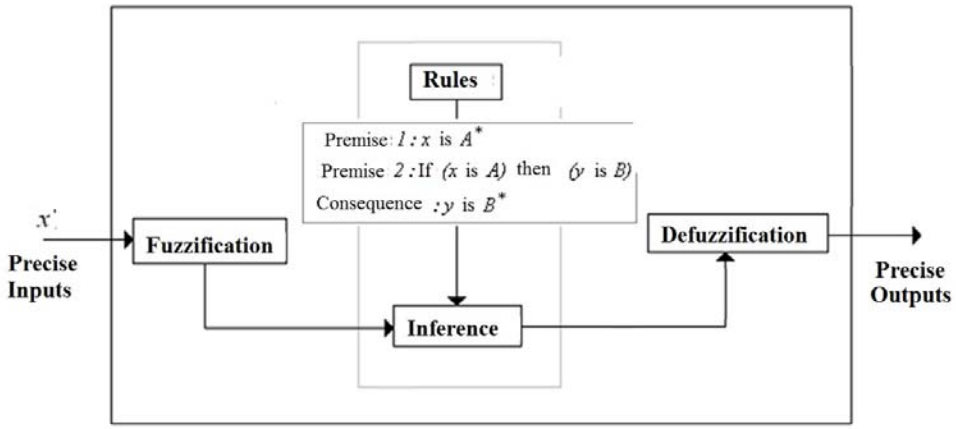

FIGURE 1. Blocks that compose a *Fuzzy* system

Fuzzy Systems are used to shape cognitive functions similar to higher order human decisions. Their project can be divided in four stages (e.g., [8,13,21]) as schematized in Figure 1: 1) a rule basis that reproduces the knowledge of a human concerning a specific domain of application; 2) an inference device that indicates how to reason in relation to the rules basis; 3) the fuzzification, which turns the input information into a shape that can be used by the *Fuzzy* System and the 4) defuzzification that puts the conclusions of an inference device in a proper shape to create an accurate output.

Fuzzy controllers use the *Fuzzy* sets and *Fuzzy* Logic (e.g., [22,23]) to implement a set of rules concerning how to control a system. Thus, it is said that the *Fuzzy* controller emulates the human cognition in a decision-making process, that is, it makes inference. These are the processes that change the information so that it might be used by an inference device (fuzzification), and transform the inference device actions so that they might be used in practical application (defuzzification).

The main feature of *Fuzzy* modeling for systems and processes can be summed up as follows: it is proper to describe complex systems with a reasonable amount of knowledge; it is easy to select linguistic variables and corresponding values to be used in nebulous rules in a relatively small category of words; the knowledge memorization is simpler and allows more easiness in the communication of models with designers and analysts due to the use of natural language.

3.1. **Theoretical foundations of *Fuzzy* Logic.** A *Fuzzy* system with multiple inputs and one output (MISO) is a nonlinear mapping of an input vector $x = [x_1, x_2, \ldots, x_n]^{\mathrm{T}} \in R^n$ to an output $y \in R$. In order to determine a MIMO (*multiple-input/multiple-output*) *Fuzzy* system with m outputs, we simply define mMISO *Fuzzy* systems. The *Fuzzy* system is featured by a set of p rules of the kind *If-then* stored in the rules basis and expressed like:

$$\begin{array}{lll} R_1: \text{ If } \left(\tilde{x}_1 \acute{e} \tilde{F}_1^{k_1} e \ldots e \tilde{x}_n \acute{e} \tilde{F}_n^{l_1}\right) \text{ then } \left(\tilde{y} \acute{e} \tilde{G}^{a_1}\right) \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ R_p: \text{ If } \left(\tilde{x}_1 \acute{e} \tilde{F}_1^{k_p} e \ldots e \tilde{x}_n \acute{e} \tilde{F}_n^{l_p}\right) \text{ then } \left(\tilde{y} \acute{e} \tilde{G}^{a_p}\right) \end{array} \tag{1}$$

Here, $\tilde{F}_b^a$ is the Ath linguistic value associated to a variable $\tilde{x}_b$ that describes an input x_b. In a similar way, $\tilde{G}^a$ is the Ath linguistic value associated to the linguistic variable $\tilde{y}$ that describes the output y. The linguistic variables vary in time and thus have a specific linguistic value. In order to apply the knowledge represented by the linguistic rules, we

are to quantify the rules meaning by using the concepts of *Fuzzy* sets and *Fuzzy* Logic. In particular, when the concepts of *Fuzzy* sets theory are used, the rules basis is expressed as a set of *Fuzzy* implications of the kind:

$$\begin{array}{lll} R_1: \text{ If } \left(F_1^{k_1} e \ldots e F_n^{l_1}\right) \text{ then } G^{a_1} \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ R_p: \text{ If } \left(F_1^{k_p} e \ldots e F_n^{l_p}\right) \text{ then } G^{a_p} \end{array} \tag{2}$$

where F_b^a and G^a are *Fuzzy* sets defined by

$$\begin{aligned} F_b^a &= \left\{(x, \mu_{F_b^a}(x_b)) : x_b \in R\right\} \\ G^a &= \{(y, \mu_{G^a}(y)) : y \in R\} \end{aligned} \tag{3}$$

The pertinence functions $\mu_{F_b^a}(x_b)$ and $\mu_{G^a}(y) \in [0,1]$ describe how correct a particular linguistic word is in numerical terms. There are several ways to define pertinence functions. Nevertheless, both of the most used functions are triangular functions and Gaussian curves.

Fuzzification. Fuzzification process is simply the process of obtaining values for the inputs x_i by calculating $\mu(x_i)$ to each of pertinence functions with input μ.

The inference device. The premises of *Fuzzy* set are indicated by the Cartesian product $F_1 \times F_2 \times \cdots \times F_n$ of each rule with a pertinence function $\mu_{F_1^{k_i} \times \cdots \times F_n^{l_i}}(x)$ that can be obtained by using the *t-norm*, that is,

$$\mu_{F_1^{k_i} \times \cdots \times F_n^{l_i}}(x) = \mu_{F_1^{k_i}}(x_1) * \ldots * \mu_{F_n^{l_i}}(x_n) \tag{4}$$

That can be defined by the minimum operator (namely, $*$ defined as minimum)

$$\mu_{F_1^{k_i} \times \cdots \times F_n^{l_i}}(x) = \min\left\{\mu_{F_1^{k_i}}(x_1), \ldots, \mu_{F_n^{l_i}}(x_n)\right\} \tag{5}$$

or the product operator (namely, $\cdot$ defined as a conventional mathematical product)

$$\mu_{F_1^{k_i} \times \cdots \times F_n^{l_i}}(x) = \mu_{F_1^{k_i}}(x_1) \cdot \ldots \cdot \mu_{F_n^{l_i}}(x_n) \tag{6}$$

among others. The *t-norm* simply quantifies the conjunction in each rule premise, being this the basic operation in *Fuzzy* Logic. Both of the approaches to quantify *Fuzzy* implications for the Ith rule are:

$$\mu_{F_1^{k_i} \times \cdots \times F_n^{l_i} \to G^{a_i}}(x, y) = \min\left\{\mu_{F_1^{k_i} \times \cdots \times F_n^{l_i}}(x), \mu_{G^{a_i}}(y)\right\}, \tag{7a}$$

and

$$\mu_{F_1^{k_i} \times \cdots \times F_n^{l_i} \to G^{a_i}}(x, y) = \mu_{F_1^{k_i} \times \cdots \times F_n^{l_i}}(x) \cdot \mu_{G^{a_i}}(y) \tag{7b}$$

The use of one or another approach depends on how one desires to quantify *Fuzzy* implications. For a given x, the pertinence function of the left sides of these two equations represents the conclusions reached by the inference device. This is named *Fuzzy* implication sets.

Defuzzification. The defuzzification process converts *Fuzzy* implication sets into numbers that will be the system output. Generally, more than one rule will be applied in each moment and therefore there will be more than one conclusion (i.e., more than one implication about *Fuzzy* sets with non-null pertinence functions) obtained by the inference device in each moment of time (i.e., the conclusions created by the inference device are the *Fuzzy* implication sets where for the Ith rule, the pertinence function of *Fuzzy* implication set is given by $\mu_{F_1^{k_i} \times \cdots \times F_n^{l_i} \to G^{a_i}}(x, y)$ with a specified x). The defuzzification combines the conclusions of all the rules and provides or offers a simple number that represents the

conclusions. So, the defuzzification can be obtained by the determination of a center of gravity that mathematically assumes the following shape:

$$y = F(x) = \frac{\sum_{i=1}^{p} c_i \int_{\tau} \mu_{F_1^{k_i} \times \cdots \times F_n^{l_i} \to G^{a_i}}(x, \tau) d\tau}{\sum_{i=1}^{p} \int_{\tau} \mu_{F_1^{k_i} \times \cdots \times F_n^{l_i} \to G^{a_i}}(x, \tau) d\tau} \tag{8}$$

Here, c_i is the $\mu_{G^{a_i}}(y)$ center of area for the Ith rule. Another way to accomplish that is through max-average norm, and $y = F(x)$ can be established like:

$$y = F(x) = \frac{\sum_{i=1}^{p} c_i \mu_{F_1^{k_i} \times \cdots \times F_n^{l_i}}(x)}{\sum_{i=1}^{p} \mu_{F_1^{k_i} \times \cdots \times F_n^{l_i}}(x)} \tag{9}$$

where each c_i is the point in which $\mu_{G^{a_i}}(y)$ reaches its maximum. It is assumed that *Fuzzy* system is defined for all $x \in R^n$, where one has $\mu_{F_1^{k_i} \times \cdots \times F_n^{l_i}}(x) > 0$ for at least one rule i, so that Equations (8) and (9) are well defined.

Fuzzy rules can be provided by experts in the form of linguistic sentences of *if-then* kind. The consistent determination of these rules consists in a foundational aspect for the good performance in *Fuzzy* inference process. Nevertheless, drawing out rules from experts cannot be an easy task. Therefore, there are automatic methods of drawing out rules from numerical data. Though they are not presented in this paper, some of them can be mentioned, for instance, the integration between *Fuzzy* inference systems and artificial neural networks, what brings about *Fuzzy-neuro* systems, which has been proven suitable for two important tasks: 1) tuning of the pertinence functions of both the input variables (backgrounds) and the output variable (consequents); 2) for the automatic creation of rules. However, if the tuning of the pertinence functions is manually or arbitrarily established, one should always keep in mind that the higher the number of sets is assigned to the backgrounds, the greater the difficulty of establishing a basis of consistent rules is; always remembering that a very large basis of rules, besides being extremely difficult of being manually obtained, inevitably causes a higher computational cost. In this paper, an alternative methodology is developed to automatically create *Fuzzy* inference rules via supervised training by using genetic algorithms. The *Fuzzy* inference system discussed in this paper is the Mamdani-type.

4. **Proposed Algorithms.** This section proposes two hybrid algorithms involving evolutionary computation and Mamdani-type *Fuzzy* inference systems for numerical solution of ordinary differential equations of first order systems, without considering control variables. The first algorithm is named Static Function methodology and the second, Dynamic Functions methodology. The essential idea behind these two algorithms is to use a supervised learning technique, which calibrates the *Fuzzy* inference system by using a genetic algorithm. The *Fuzzy* system is projected in such a way that the mean derivatives might be obtained by direct methodology in its output.

4.1. **Static Functions Methodology.** The Static Functions Methodology is so called because the pertinence functions of Fuzzy system sets both the input and output are not changed over the generations of the genetic algorithm. Thus, the only difference from one system to another is its inference rules that are changed.

First of all, one must establish how many *Fuzzy* levels each one of the inputs and outputs must have. In fact, each *Fuzzy* level is a *Fuzzy* set that classifies an input or an output.

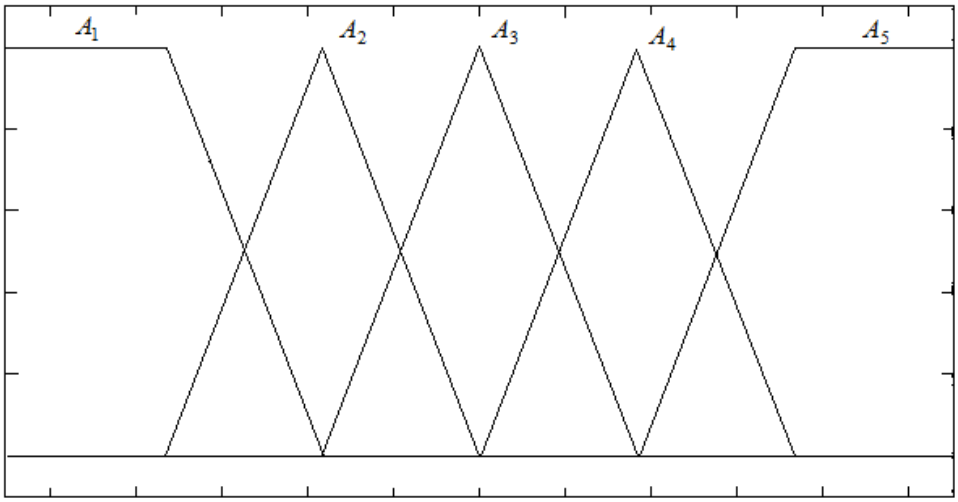

FIGURE 2. Pertinence functions for five levels inputs

The number of levels corresponds to the number of *Fuzzy* sets that are used to classify the input. So, by considering that the input i has N_e^i levels, it means that this input has N_e^i. The pertinence functions for the $N_e^i = 5$ case are shown in Figure 2. Note that there are two trapezoidal levels on the extremes and triangular intermediate levels.

The presented levels are calculated like the following. The input interval is divided into $N_e^i + 1$ intervals of the same size. The triangular functions are built in such a way that its maximum (place where they reach a degree of pertinence equals one) is on the extreme of one of the intervals, and the points where the degree of pertinence is zero are the extremes of the neighboring intervals (see details in Figure 2). Concerning the trapezoidal functions, the first trapezoidal function has value "one" until the end of the first interval and then, it decreases to zero at the end of the second interval.

When the number of input variables (n_e), the number of output variables (n_s), and the number of input and output levels are defined, it is necessary to establish the rules. The *Fuzzy* system rules will follow this model:

$$\begin{aligned} &R_i: \ (\text{If } \tilde{x}_1 \text{ é } X_i^1 \text{ e } \tilde{x}_2 \text{ é } X_i^2 \text{ e } \ldots \text{ e } \tilde{x}_{n_e} \text{ é } X_i^{n_e}) \text{ then} \\ &\tilde{y}_1 \text{ é } Y_i^1, \ \tilde{y}_2 \text{ é } Y_i^2, \ldots, \tilde{y}_{n_s} \text{ é } Y_i^{n_s} \end{aligned} \tag{10}$$

The number of rules of the system can be calculated by the multiplicative principle. Each input variable may assume each one of the linguistic values N_e^i. The number of rules of the system is shown in Equation (11). Each one of these rules must prescribe the output variable values, that is, values should be established for each one of the n_s (one for each of the outputs).

$$\text{Number of Rules} = \prod_{i=1}^{n_e} N_e^i \tag{11}$$

The chromosomes of the genetic algorithm represent this set of rules. The output variable y_j needs a linguistic value Y_i^j for each one of the rules, so, for each output variable, $\prod_{i=1}^{n_e} N_e^i$ integers are needed to represent these linguistic values, and each one of these integers is associated to one of the N_s^j *Fuzzy* sets of the y_j output. Thus, the chromosome for the genetic algorithm should have $\prod_{i=1}^{n_e} N_e^i$ integers for each output, which

makes that the chromosome size is the value presented in Equation (12).

$$\text{Chromosome Size } = n_s \prod_{i=1}^{n_e} N_e^i \tag{12}$$

Consider a system as described in $n_e = n_s = 2$, where each one of the inputs and outputs has two levels. The *Fuzzy* sets for the input x_1 are A_1 and A_2, for the input x_2 they are B_1 and B_2, for the output y_1 they are C_1 and C_2 and for the output y_2 they are D_1 and D_2. For this system, there is a total of four rules $\left(\prod_{i=1}^{n_e} N_e^1 = \prod_{i=1}^{2} N_e^i = N_e^1 \cdot N_e^2 = 2 \cdot 2 = 4\right)$ and an eight-size chromosome $\left(n_s \prod_{i=1}^{n_e} N_e^1 = 2 \cdot \prod_{i=1}^{2} N_e^i = 2 \cdot N_e^1 \cdot N_e^2 = 2 \cdot 2 \cdot 2 = 8\right)$. An example of a chromosome with its generated rules is shown below.

Chromosome: [1 2 1 2 1 2 1 2]

1. If (x_1 é A_1) e (x_2 é B_1) then (y_1 é C_1) e (y_2 é D_1)
2. If (x_1 é A_2) e (x_2 é B_1) then (y_1 é C_2) e (y_2 é D_2)
3. If (x_1 é A_1) e (x_2 é B_2) then (y_1 é C_1) e (y_2 é D_1)
4. If (x_1 é A_2) e (x_2 é B_2) then (y_1 é C_2) e (y_2 é D_2)

The tabular representation of the previous chromosome can be seen in Table 1. The first four bits of the chromosome refer to the rules of the first output variable, and the last four bits of the chromosome are related to the rules of the second output variable.

TABLE 1. Table representation of rules

y_1 **Output Variable**			y_2 **Output Variable**		
x_1/x_2	B_1	B_2	x_1/x_2	B_1	B_2
A_1	C_1	C_1	A_1	D_1	D_1
A_2	C_2	C_2	A_2	D_2	D_2

Based on the above in this section and in the previous sections, it is possible to formulate a general algorithm for the Static Functions methodology. This algorithm can be divided into three phases or stages as described as follows:

a) Phase 01 – Obtainment of a training pattern;
b) Phase 02 – Evolutionary training of the *Fuzzy* system;
c) Phase 03 – Test and simulation of learning model.

Algorithm 1: Static function methodology (without considering control variables).

a) Phase 01 – Obtainment of a training pattern.

1. Given the finite domains of interest $\left[y_j^{\min}(t_0), y_j^{\max}(t_0)\right]^n$ in t_0 for $j = 1, 2, \ldots, n$ of state variables, q random values are generated according to a uniform distribution within these intervals like:

$$p_i = \left[y_1^i(t_0)\ y_2^i(t_0) \ldots y_n^i(t_0)\right]^T, \text{ for } i = 1, 2, \ldots, q \tag{13a}$$

or

$$P = [p_1 : p_2 : \ldots : p_q]_{nxq} \tag{13b}$$

where n is the total number of state variables, and q is the total number of training patterns.

2. By using a high-order integration, all the p_i initial conditions are propagated to $i = 1, 2, \ldots, q$ for the state obtainment in the $t_o + \Delta t$ instant, that is,

$$p_i^{\Delta t} = \left[y_1^i(t_0 + \Delta t)\ y_2^i(t_0 + \Delta t) \ldots y_n^i(t_0 + \Delta t)\right]^T, \text{ for } i = 1, 2, \ldots, q \tag{14a}$$

or

$$P^{\Delta t}\left[p_1^{\Delta t} : p_2^{\Delta t} : \ldots : p_p^{\Delta t}\right]_{nxq} \tag{14b}$$

3. The T_i output vectors or output training patterns of the *Fuzzy* system should be established (direct methodology of mean derivatives) like (see [31]):

$$\begin{aligned} T_i &= \frac{1}{\Delta t} \cdot \left[y_1^i(t_0 + \Delta t) - y_1^i(t_0)\ y_2^i(t_0 + \Delta t) - y_2^i(t_0) \ldots y_n^i(t_0 + \Delta t) - y_n^i(t_0)\right]^T \\ &= \left[\tan_{\Delta t}{}^k\alpha_1^i\ \ \tan_{\Delta t}{}^k\alpha_2^i \ldots \tan_{\Delta t}{}^k\alpha_n^i\right]^T = \tan_{\Delta t}{}^k\alpha^i, \text{ for } i = 1, 2, \ldots, q \end{aligned} \tag{15a}$$

or

$$T = [T_1 : T_2 : \ldots : T_q]_{nxq} \tag{15b}$$

4. The matrices P_{nxq} and T_{nxq} form the training patters that will be evaluated by the hybrid training system by using *Fuzzy* logic and genetic algorithms in stage 02. Note that once specified the value of the Δt integration step, it cannot be modified anymore.
5. The input/output training patterns should be normalized through the expressions $P_{nxq}^N = \frac{1}{\rho_e} \cdot P_{nxq}$ and $T_{nxq}^N = \frac{1}{\rho_s} \cdot T_{nxq}$, where ρ_e and ρ_s are respectively the maximum values in the modulus of the component elements of the matrices P_{nxq} and T_{nxq}. Alternatively, one might have a distinctive normalization factor for each line of the matrices P_{nxq} and T_{nxq}.

Phase 02 – Evolutionary training of *Fuzzy* system.

1. Start from the training matrices P_{nxq}^N and T_{nxq}^N obtained in phase 01.
2. Precisely establish the number of input variables (n_e) and the number of output variables (n_s). Note that in this case, $n_e = n_s = n$, because the *Fuzzy* system inputs are $y_1(t), y_2(t), \ldots, y_n(t)$ and the outputs are $\tan_{\Delta t}{}^k\alpha_1^i, \tan_{\Delta t}{}^k\alpha_2^i, \ldots, \tan_{\Delta t}{}^k\alpha_n^i$ as illustrated in Figure 3.
3. Consider N_e^i for $i = 1, 2, \ldots, n$ the total number of the linguistic values of the input variable i and N_s^j for $j = 1, 2, \ldots, n$ the total number of the linguistic values of the output variable j. In such a way, establish the total number of *Fuzzy* system rules given by NR (Number of Rules) $= \prod_{i=1}^{n_e} N_e^i = \prod_{i=1}^{n} N_e^i$ and the size of chromosome (SC), in order to represent all the rules given by $SC = n_s \prod_{i=1}^{n_e} N_e^i = n \prod_{i=1}^{n} N_e^i$.

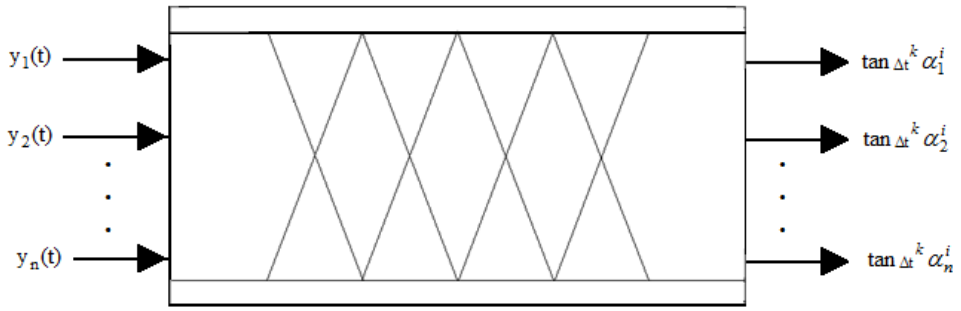

FIGURE 3. Configuration of the input/output variables projected in Mamdani-type *Fuzzy* system

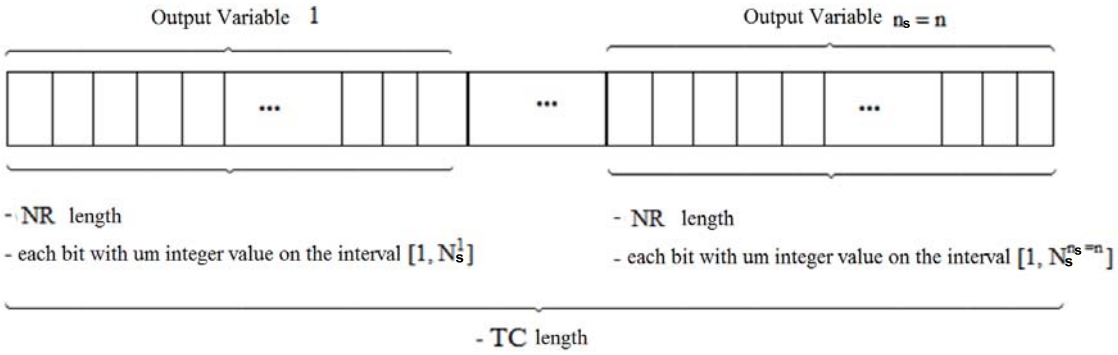

FIGURE 4. Pattern configuration of each chromosome to represent a specific *Fuzzy* set of rules for the system considered in Figure 3

4. Each chromosome of the genetic algorithm must have an SC length and represents a set of specific rules of the *Fuzzy* system in question. Each chromosome bit value is an integer value obtained from N_s^j, for $j = 1, 2, \ldots, n$. For example, where $N_s^j = r$, the integer values allowed for N_s^j should be confined within the range $[1, r]$ (see Figure 4).
5. Do $m = 1$ until *NIP* (Number of Individuals in Population)
   5.1 Randomly generate a chromosome C_m with the configurations as specified in step 4.
   5.2 For the chromosome C_m calculate $EQM_m = \sum_{j=1}^{n} \frac{1}{n} \sum_{i=1}^{q} \frac{1}{q} \left(\tan_{\Delta t}^{N\ k} \bar{\alpha}_j^i - \tan_{\Delta t}^{N\ k} \hat{\alpha}_j^i \right)^2$ where EQM_m is the mean square error of all q training patterns (each training pattern is submitted to a fuzzification and defuzzification process), n is the total number of state variables, $\tan_{\Delta t}^{N\ k} \bar{\alpha}_j^i = \frac{\tan_{\Delta t}{}^k \bar{\alpha}_j^i}{\rho_s}$ is the value of the normalized mean derivative *desired* or goal, and $\tan_{\Delta t}^{N\ k} \hat{\alpha}_j^i = \frac{\tan_{\Delta t}{}^k \hat{\alpha}_j^i}{\rho_s}$ is the value of the normalized mean derivative *estimated* by the *Fuzzy* system, which has a set of rules specified by the chromosome C_m.
   5.3 Estimate the *fitness* function (f_m) of the chromosome C_m, mathematically given by $f_m = \frac{1}{EQM_m}$. Note that it is desired to find an excellent chromosome given by C_m^* that might *minimize* EQM_m value, but it is needed to use the inverted value of EQM_m since the genetic algorithms naturally are maximization algorithms.
   5.4 Alternatively, instead of directly using the *fitness* function value, one may choose the *rank* selection technique. For example, the most adapted individual receives the *rank* 1, the second most adapted individual receives the *rank* 2 and so on. The size of the roulette slice for each of the individuals would be proportional to $1 \Big/ \sqrt{rank}$.

   End m

   NOTE: once that *NIP* is established, it cannot be modified anymore.
6. Start the application of the evolutionary algorithm.
   6.1 Randomly create the chromosomes $C_1, C_2, \ldots, C_{NIP}$ of the initial population, with their $f_1, f_2, \ldots, f_{NIP}$ respective *fitness* functions, as specified in step 5.
   6.2 Do $m = 1$ until *NTG* (Number of Total Generations)

$$\text{Initial Population (IP)} \leftarrow \begin{bmatrix} C_1 \\ \vdots \\ C_{NIP} \end{bmatrix} = \left[C_1^T : C_2^T : \ldots : C_{NIP}^T \right]^T$$

$$\text{Fitness Function (FF)} \leftarrow \begin{bmatrix} f_1 \\ \vdots \\ f_{NIP} \end{bmatrix} = [f_1\ f_2\ \ldots\ f_{NIP}]^T$$

$\text{ARW} = [C_1'^T : \ldots : C_{NIP}'^T] \leftarrow$ *Reproduction through Roulette wheel algorithm (IP, FF)*

$AOM = \left[C_1''^T : \ldots : C_{NIP}''^T\right]^T \leftarrow$ *Apply Mutation Operator (AMO)*

$PI \leftarrow AOM$ (Replacement of the current population for the next population)

End m

Phase 03 – Test and simulation of the learning model.

${}^k y^i \leftarrow$ Vector $p = [y_1(t_0)\ y_2(t_0) \ldots y_n(t_0)]^T$ (Initial Condition)

Do $i = 1$ until H (Simulation Horizon)

${}^k y_N^i \leftarrow \frac{{}^k y^i}{\rho_e}$ (Normalization of the Initial Condition)

$\tan_{\Delta t}^N {}^k\alpha^i \leftarrow f_{SFM}[{}^k y_N^i]$ $f_{\text{SFN}}[.] \equiv$ Normalized *Fuzzy* System

$\tan_{\Delta t} {}^k\alpha^i \leftarrow \rho_s \cdot \tan_{\Delta t}^N {}^k\alpha^i$ (Desnormalization of Mean Derivative)

${}^{k+1}y^i = \tan_{\Delta t} {}^k\alpha^i \cdot \Delta t + {}^k y^i$

${}^k y^i \leftarrow {}^{k+1}y^i$

End i

With the description of the three phases of this algorithm, the description of Static Functions Methodology is ended. In the following section, it is possible to realize that step 4 of Phase 02 can be improved, which leads to an enhancement in the learning of the considered systems. This new methodology was conveniently named Dynamic Functions Methodology.

4.2. **Dynamic Functions Methodology.** Dynamic Functions Methodology is so called because the pertinence functions of the output *Fuzzy* sets are modified and/or dislocated on the Cartesian axis along the generations of the genetic algorithm.

In this new methodology, the linguistic values of the input variable i keep on being any integer number given by N_e^i for $i = 1, 2, \ldots, n$ which must be accepted. The number of rules also keep on being NR (Number of Rules) $= \prod_{i=1}^{n_e} N_e^i = \prod_{i=1}^{n} N_e^i$ for the specific case of $n_e = n_s = n$. Nevertheless, the number of linguistic values of the output variable j necessarily is equal to the number of rules, that is $N_s^j = \prod_{i=1}^{n} N_e^i$ for $j = 1, 2, \ldots, n$. Therefore, the linguistic values of output variables are tied to the NR value. In this new methodology, every rule of *Fuzzy* set of rules has a distinctive and unique trapezium as an output, which can or cannot be symmetrical and may also have its center dislocated in Cartesian axis. Since it takes at least five dimensions to characterize a trapezium in the Cartesian axis (see Figure 5), the chromosome size for this methodology necessarily needs to be equal to $CS = 5 \times n_s \times \prod_{i=1}^{n_e} N_e^i = 5 \times n \times \prod_{i=1}^{n} N_e^i$. It is important not to puzzle the chromosome size (CS) with the *total number of output levels (TNOL) of pertinence functions. The TNOL variable has an* $n \times \prod_{i=1}^{n} N_e^i$ dimension, that is, five times less than CS.

Each one of these output levels must contain an associated *Fuzzy* set. All these sets have a trapezoidal shape. The trapeziums were chosen because a triangle can be considered a degenerate trapezium (if the two points of the lower base coincide), and this is good because it continues to assert the theorem that states the *Fuzzy* systems with triangular pertinence functions are universal approximators of functions.

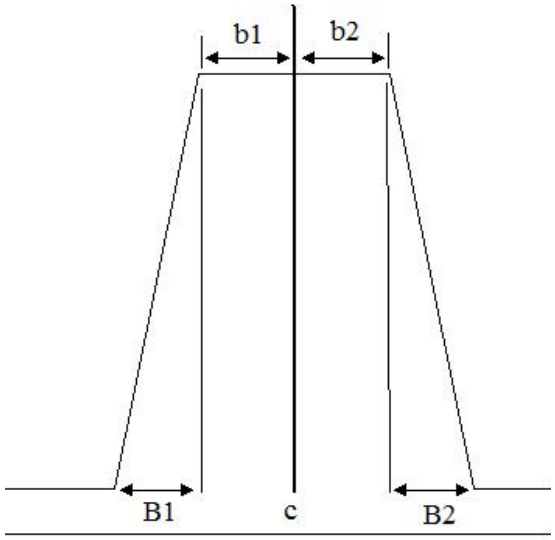

FIGURE 5. Graphical representation of chromosome values

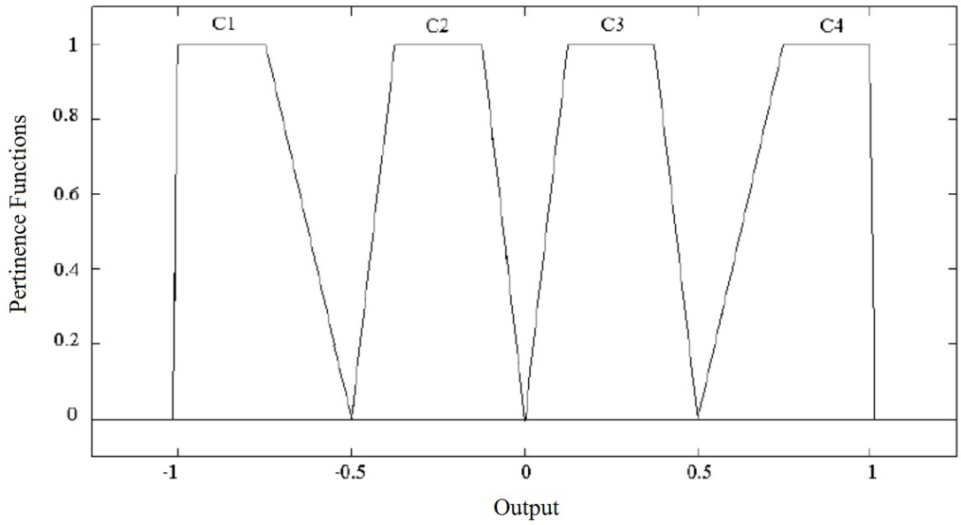

FIGURE 6. Pertinence functions of the output variable for chromosome 1

In order to define a trapezium in each one of these *Fuzzy* sets, five values are used, which are: center (c), b_1, B_1, b_2 and B_2. Center (c) represents a reference to other four values. As it can be seen in Figure 5, b_1 is the size of the smaller base on the left of the center, and $b_1 + B_1$ is the size of the bigger base on the left of the center. Values b_2 and $b_2 + B_2$ are analogous, but for the right side of the center. Figure 5 represents the five values which define the trapezium. Reiterating it, the chromosome of this methodology has a size equal to five times the number of output levels.

For example, for the specific case of Figure 6, the values used to define each chromosome are real numbers in which the center varies in the interval $[-1, 1]$ and b_1, B_1, b_2 and B_2, and varies in the interval $[0, 0.5]$. As an example given, for a system of $n_e = 2$, $N_e^1 = 2$, $N_e^2 = 2$ and $n_s = 1$, the number of output levels is $n_s \times \prod_{i=1}^{n_e} N_e^i = 1 \times N_e^1 \times N_e^2 = 1 \times 2 \times 2 = 4$. Each of these levels requires five values to define their respective trapeziums, causing the chromosome has twenty real values. This system rules, as well as the examples of

chromosomes and the generated systems, are shown below and in Figure 6.

1. If $(x_1$ *é* $A_1)$ e $(x_2$ *é* $B_1)$ then $(y_1$ *é* $C_1)$
2. If $(x_1$ *é* $A_2)$ e $(x_2$ *é* $B_1)$ then $(y_1$ *é* $C_2)$
3. If $(x_1$ *é* $A_1)$ e $(x_2$ *é* $B_2)$ then $(y_1$ *é* $C_3)$
4. If $(x_1$ *é* $A_2)$ e $(x_2$ *é* $B_2)$ then $(y_1$ *é* $C_4)$

If the chromosome which represents this set of rules is defined *by [–1 0 0 0.25 0.25 –0.25 0.125 0.125 0.125 0.125 0.25 0.125 0.125 0.125 0.125 1 0.25 0.25 0 0]*, it generates the output rules as schematized in Figure 6. It is important to note that Dynamic Functions Methodology is only different on step 4 of Phase 02 when compared to Static Functions Methodology. The chromosome size (*CS*) required by the Dynamics Functions Methodology has a factor of 5 times the size of the chromosome of the Static Functions Methodology. All other steps of the three phases of algorithms proposed in the previous section continue unchanged for this new methodology.

It is still useful to note that, at the first sight, the Static Functions Methodology approximates a finite number of functions, because the number of combinations of different rules, however vast, still is finite. Nevertheless, the Dynamic Functions Methodology presents continual changes on the output sets and, therefore, the number of approximated functions becomes infinite.

5. **Numerical Outcomes.**

The methodology outcomes are shown for solving systems of differential equations (ODES). Equations are solved in a specific interest domain in which the extremes of this domain serve for the normalization and denormalization stages of input and output. The equation system used to verify the efficiency of methods is presented as follows. All the presented systems are autonomous, that is, they are of the kind $\dot{x} = f(x)$, in which derivative functions are independent from time.

5.1. **Ordinary differential equations systems (ODES) for test.**

Example 5.1. *Simple Pendulum. This system of equations is quite simple and was chosen to verify whether the generated approximators are able to adapt to this first test. Besides being linear, this system has the advantage of making the solution a linear combination of sine and cosine, which creates a solution within an easily found domain.*

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -x_1 \end{cases} \tag{16}$$

Example 5.2. *Simple Pendulum with air resistance.*

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\frac{g}{l} \cdot sen(x_1) - k \cdot x_2 \end{cases} \tag{17}$$

Example 5.3. *Control loop with relay. Figure 7 displays the blocks diagram of the control system in question. Equation (18) shows the differential equations system that represents the considered system.*

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \begin{cases} 10, & se\ 5 - x \geq 1 \\ -10, & se\ 5 - x \leq 1 \\ 0, & for\ another\ chaos \end{cases} \end{cases} \tag{18}$$

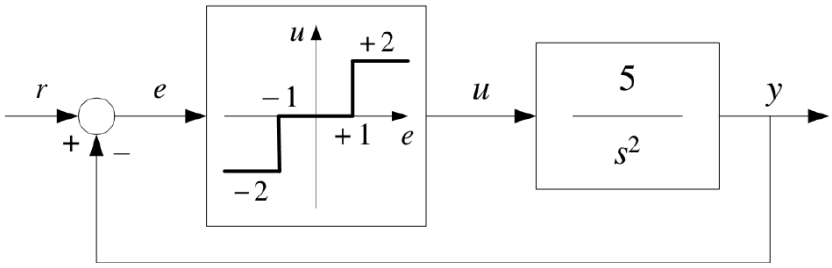

FIGURE 7. Blocks diagram of control loop with relay

TABLE 2. Fuzzy training system data using static methodology and seven input and output levels for the problem of the simple pendulum

Input Number:	2
Output Number:	2
Total of Training Points:	3025
Input and Output Levels (of pertinence functions):	7
Integration Step:	0.01
Population Size:	20
Training Domain:	$[-1.10\ 1.10] \times [-1.10\ 1.10]$

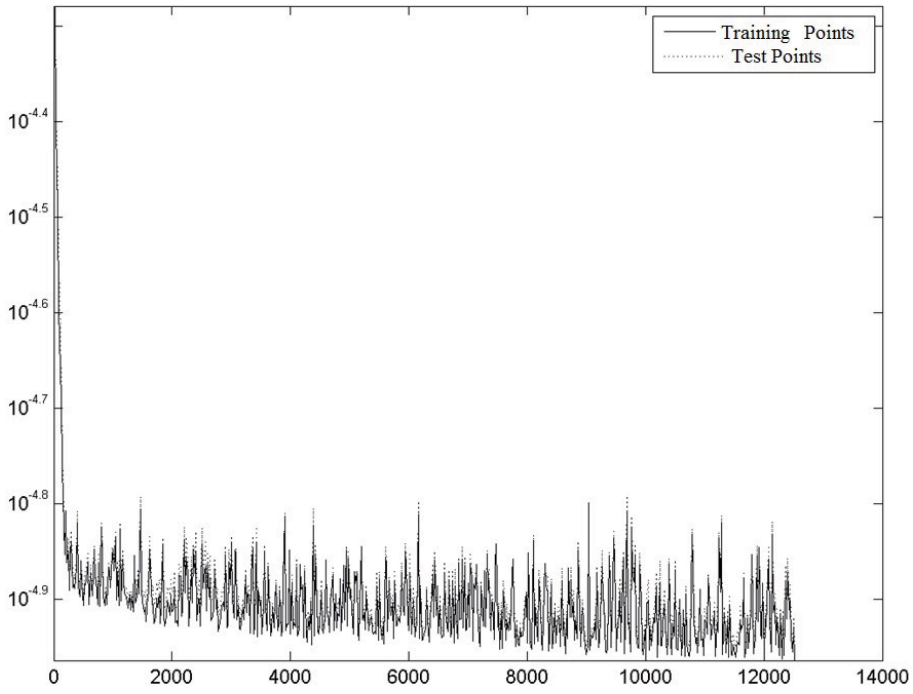

FIGURE 8. Evolution of populations in the training system

5.2. **Static Functions Methodology outcomes.** First of all, the outcomes of Static Functions Methodology are demonstrated for the problems of the simple pendulum and simple pendulum with air resistance. Table 2 reveals the training patterns for the sine function learning. Figure 8 presents the evolution of mean square error along the populations, the continuing line exemplifies the mean square error of the population average

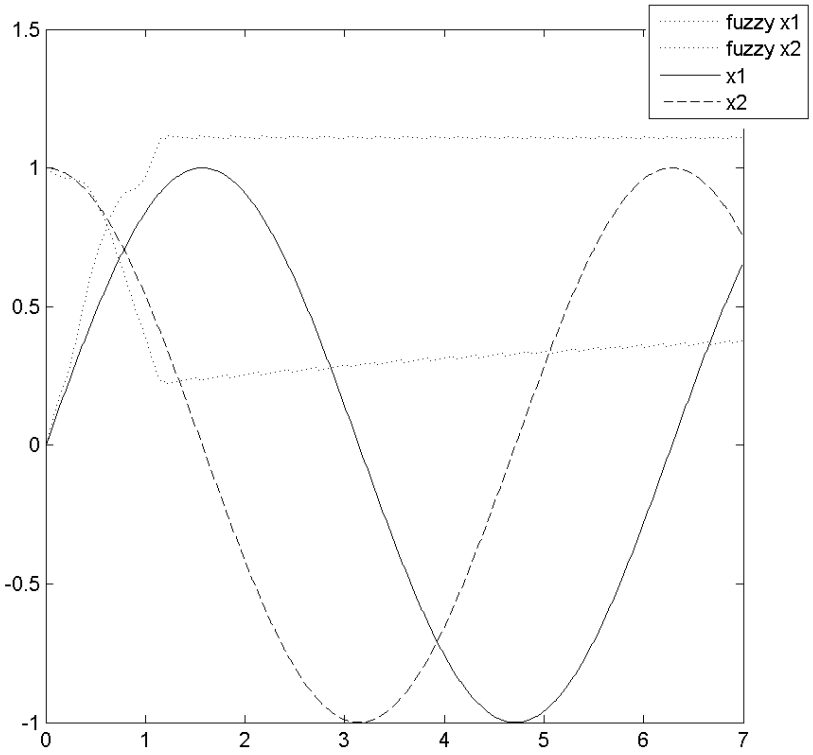

FIGURE 9. Propagation of *Fuzzy* system to the initial condition [0 1]

TABLE 3. Training patterns for simple pendulum with air resistance and eleven input levels

Input Number:	2
Output Number:	2
Total of Training Points:	3025
Input Levels (of pertinence functions):	11
Integration Step:	0.01
Population Size:	20
Training Domain:	$[-1.10\ 1.00] \times [-2.00\ 2.00]$

concerning the training points and the dashed line represents the mean square error relating to test points. Figure 9 demonstrates the propagation of the solution found for the initial condition [0 1].

The following outcomes refer to static function methodology for the problem of simple pendulum with air resistance. The training patterns are displayed in Table 3. Figure 10 presents the evolution of mean square error along the populations. Figure 11 reproduces the propagation of the solution found for the initial condition [0 1].

5.3. **Dynamic Functions Methodology outcomes.** In this section, we show the outcomes obtained for the Dynamic Functions Methodology. Firstly, the outcomes for sine learning are presented. Table 4 reveals the configuration of training patterns for the sine function learning. Figure 12 demonstrates the evolution of mean square error along the populations. Figure 13 shows the solution obtained by this method for the initial condition [0.5 0.5].

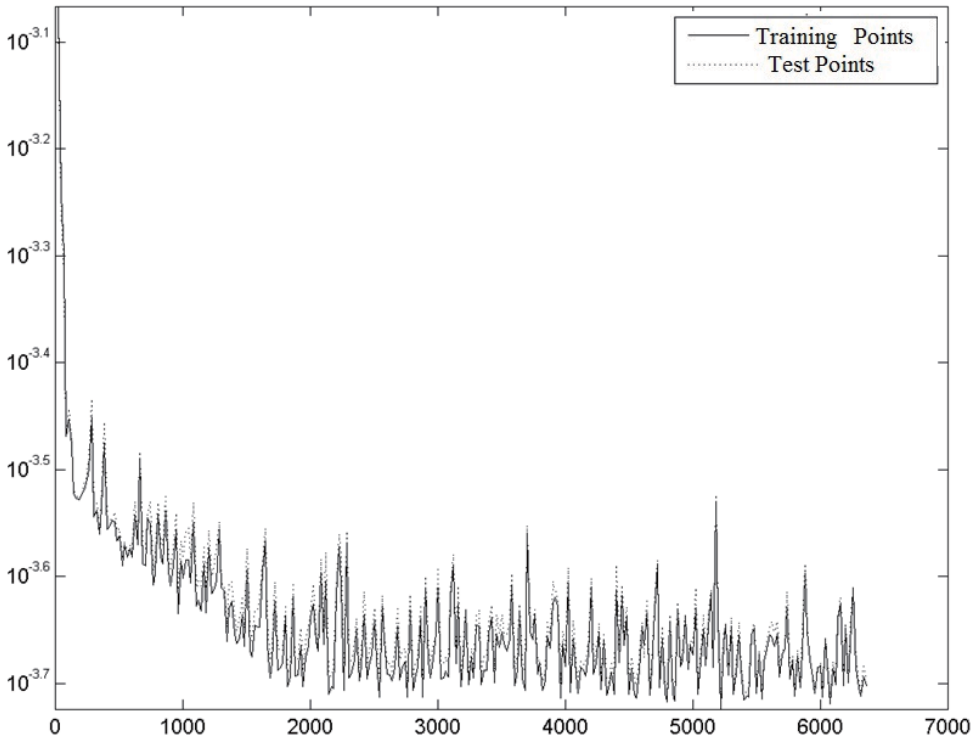

FIGURE 10. Evolution of populations in the training system

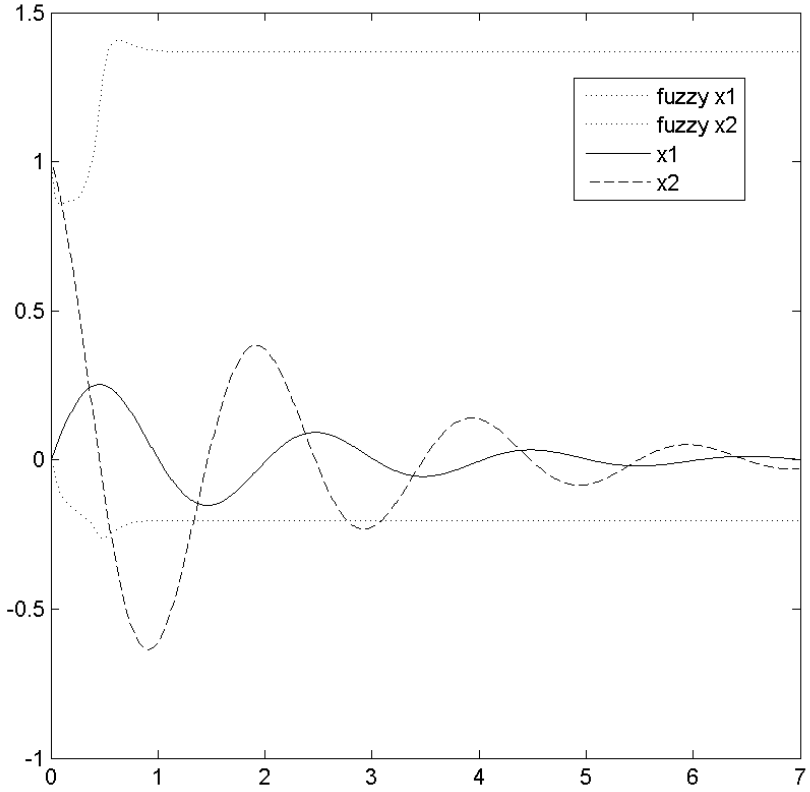

FIGURE 11. Propagation of *Fuzzy* system to the initial condition [0 1]

TABLE 4. Training patterns for sine function learning using the Dynamic Functions Methodology

Input Number:	2
Output Number:	2
Total of Training Points:	3025
Input Levels (of pertinence functions):	5
Integration Step:	0.01
Population Size:	40
Training Domain:	$[-1.50\ 1.00] \times [-1.30\ 1.00]$

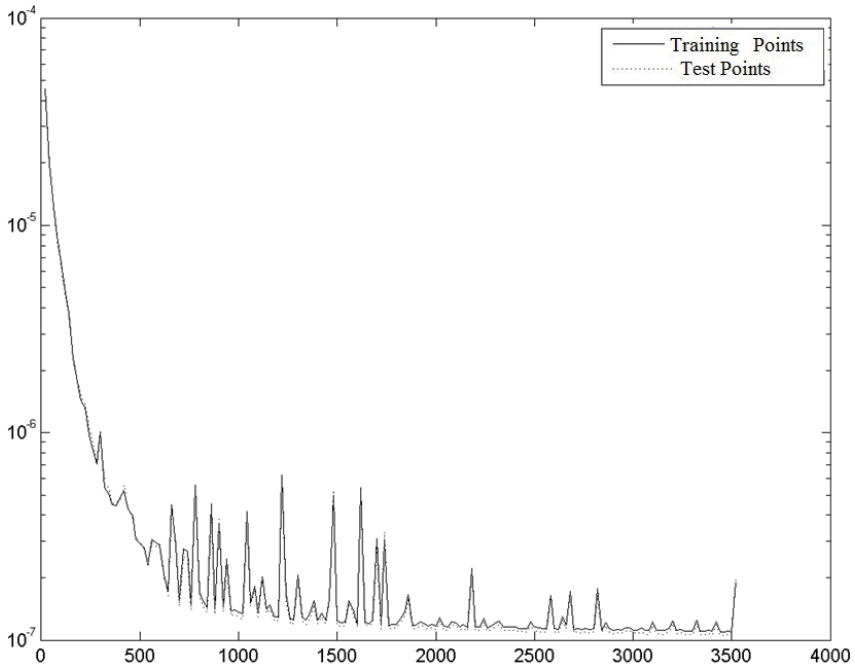

FIGURE 12. Evolution of populations in the training system

Table 5 demonstrates the configuration of the initial parameters for the learning of simple pendulum with air resistance and tree input levels. Figure 14 displays the evolution of mean square error along the populations. Figure 15 shows the propagation of the solution for the initial condition [0 1]. Finally, Figures 16 and 17 exemplify the trapeziums configurations obtained by this method in order to create the pertinence functions for the outputs x_1 and x_2 achieved by the best chromosome of the genetic algorithm. Figure 18 reveals the same problem, but it was resolved by using nine input levels. Note that the outcome has much improved when compared to the solution presented in Figure 15, where only tree input levels were used for pertinence functions. The resulting trapeziums were omitted in Figure 18 due to their great number; in fact, they are $9 \times 9 = 81$ trapeziums for the output pertinence functions.

The outcomes of dynamic function methodology for the loop control with relay are presented in the following. The outcomes for this dynamical system are only presented in relation to the dynamic methodology, since the static methodology did not perform good results for the two previous problems.

Table 6 shows the configuration of initial parameters for the relay control system learning with nine input levels. Figure 19 displays the evolution of mean square error along

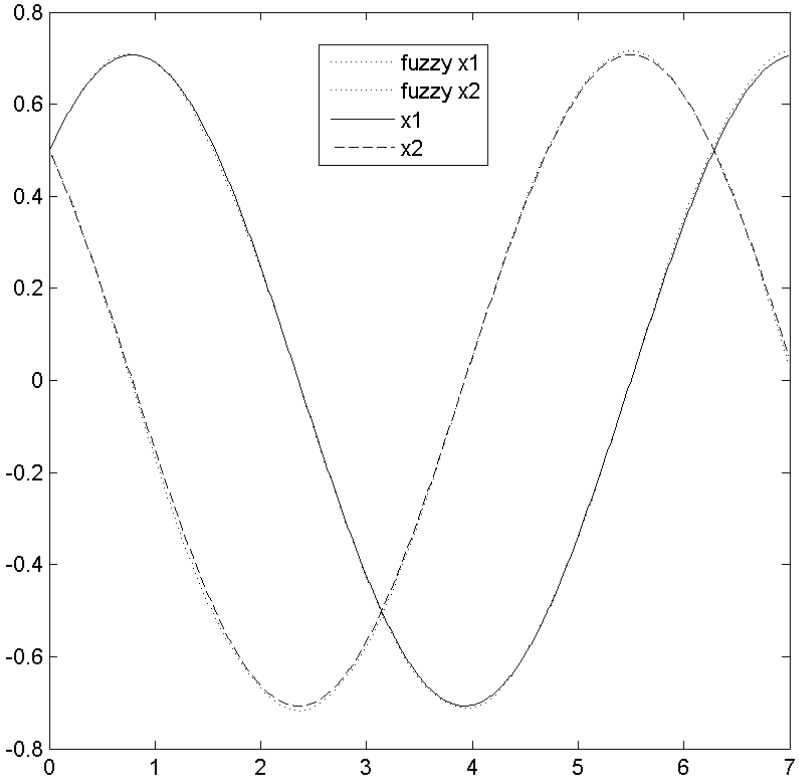

FIGURE 13. Propagation of *Fuzzy* system for the initial condition [0.5 0.5]

TABLE 5. Training patterns for the simple pendulum learning with air resistance by using Dynamic Functions Methodology

Input Number:	2
Output Number:	2
Total of Training Points:	3025
Input Levels (of pertinence functions):	3
Integration Step:	0.01
Population Size:	40
Training Domain:	$[-1.50\ 1.00] \times [-1.30\ 1.00]$

the populations. Figure 20 demonstrates the propagation of the solution for the initial condition [0 1].

6. **Conclusions.** Concerning the practical outcomes, the following conclusions are drawn from this paper. It is a fact that the work has achieved another goal, which is to implement an approximator of *Fuzzy* functions to approximate solutions of ordinary differential equations systems. Two implementations were created with this goal: the Static Functions Methodology and the Dynamic Functions Methodology. As the main contributions one may cite:

1. The use of *Fuzzy* systems as function approximators as a substitutive for neural networks;
2. Automatically creation of complex *Fuzzy* systems;
3. The use of *Fuzzy* systems for dynamic systems behavior learning;

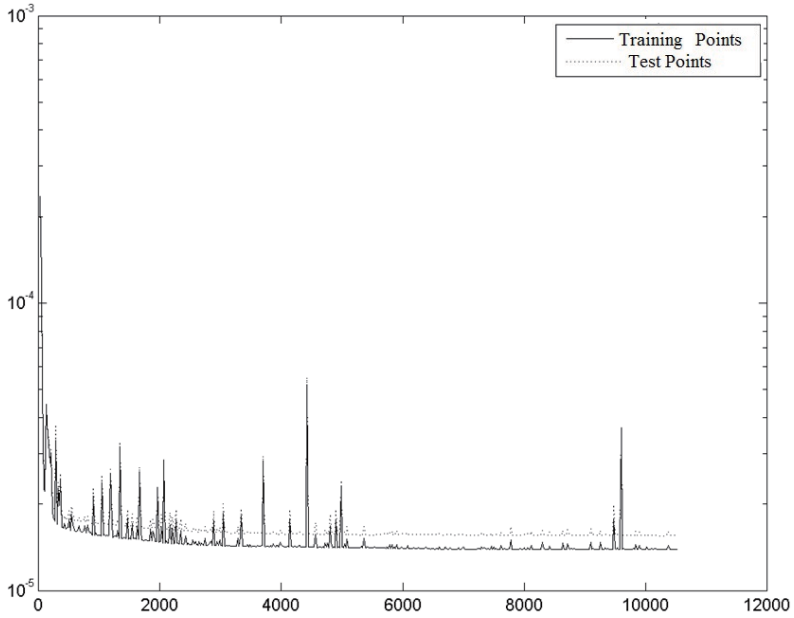

FIGURE 14. Evolution of populations in the training system with nine levels

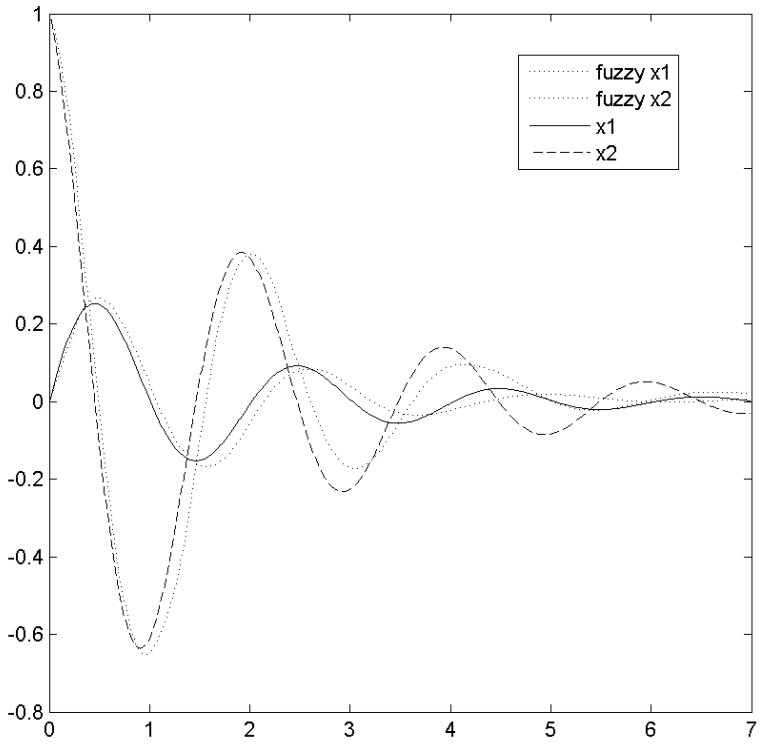

FIGURE 15. Propagation of *Fuzzy* system for the initial condition [0 1] with three input levels

4. The use of genetic algorithms for *Fuzzy* systems training.

Nevertheless, some difficulties were evidenced. The static functions methodology did not reach the desired outcome. When *Fuzzy* systems were fed back, they did not pursue the estimated solutions through Runge-Kutta method. In the cases of test examples, the system did not exceed squared errors of the order of 10^{-5}. This caused the solution was not approximated in the correct way. And besides, this methodology cannot be tested

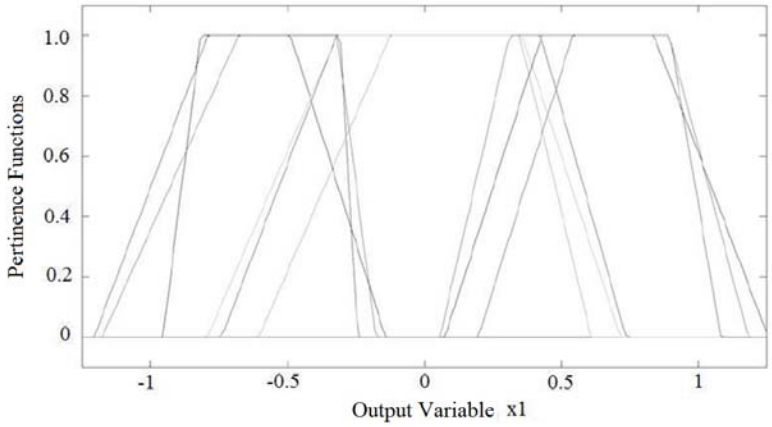

FIGURE 16. Trapeziums configuration of *Fuzzy* system for the output x_1

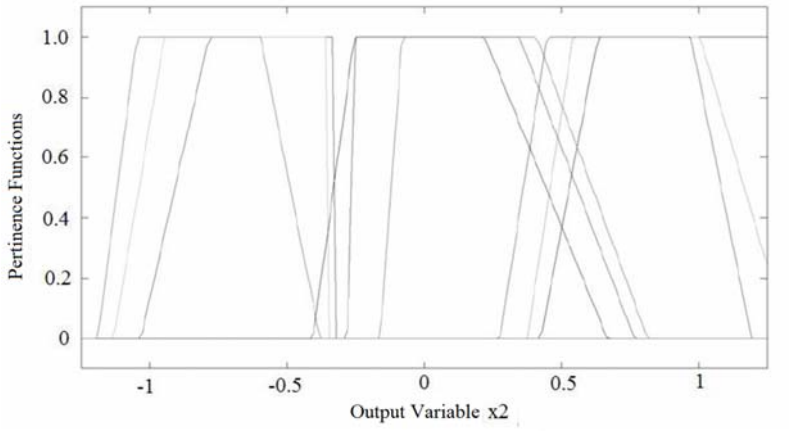

FIGURE 17. Trapeziums configuration of *Fuzzy* system for the output x_2

TABLE 6. Training patterns for control loop learning using the Dynamic Functions Methodology

Input Number:	2
Output Number:	2
Total of Training Points:	3025
Input Levels (of pertinence functions):	9
Integration Step:	0.01
Population Size:	20
Training Domain:	$[-1.00\ 11.00] \times [-10.0\ 10.00]$

for more input levels due to two problems: 1) systems with two inputs have quadratic running time with the number of input levels and 2) the chromosome size quadratically varies with the number of entries levels, too. The first factor makes that the running time of populations increases very quickly and the second causes an increase in the search space, so that the genetic algorithm needs additional generations to achieve a desired error. Therefore, the static functions methodology did not become suitable for the skilful computation time.

The dynamic functions methodology has demonstrated better outcomes by obtaining mean square errors in the order of 10^{-6} and 10^{-7}, and it was able to follow the solution

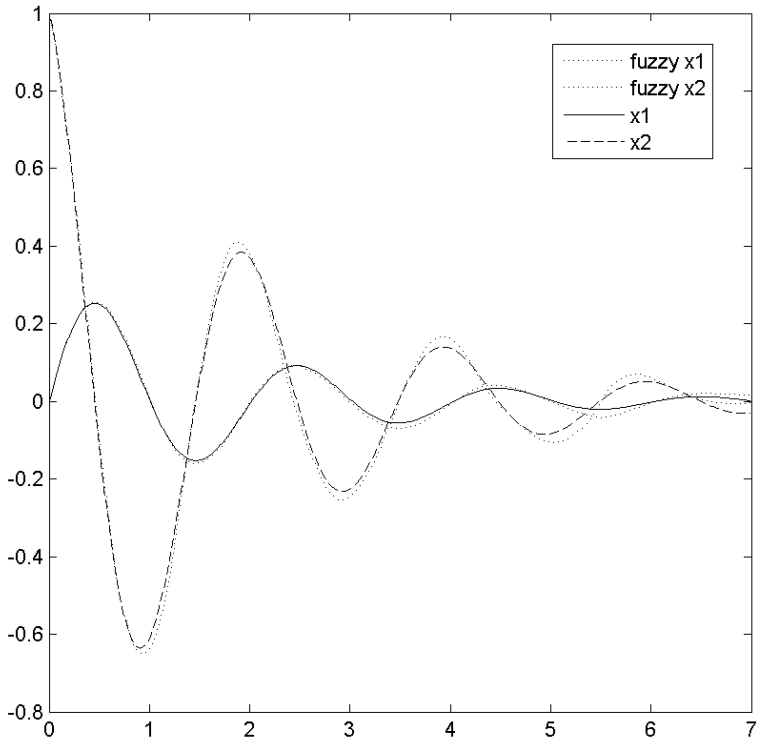

FIGURE 18. Propagation of *Fuzzy* system for the initial condition [0 1] with nine input levels

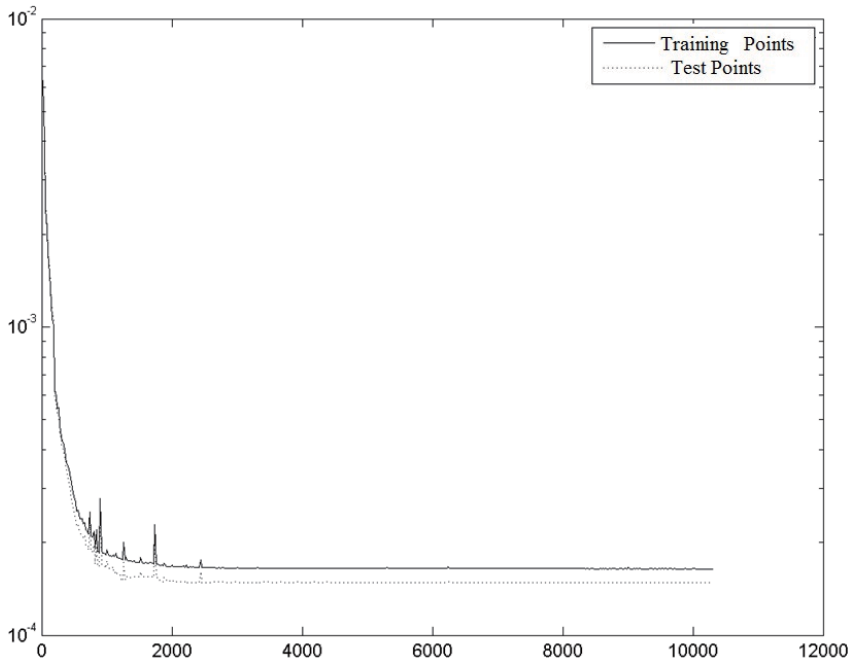

FIGURE 19. Evolution of populations in the training system with nine levels

obtained through Runge-Kutta method. The outcomes have proved satisfactory for systems with up to two inputs. For systems with more number of inputs, the same problems faced by the previous methodology made the Dynamic Functions Methodology unsuitable due to the computation time demanded.

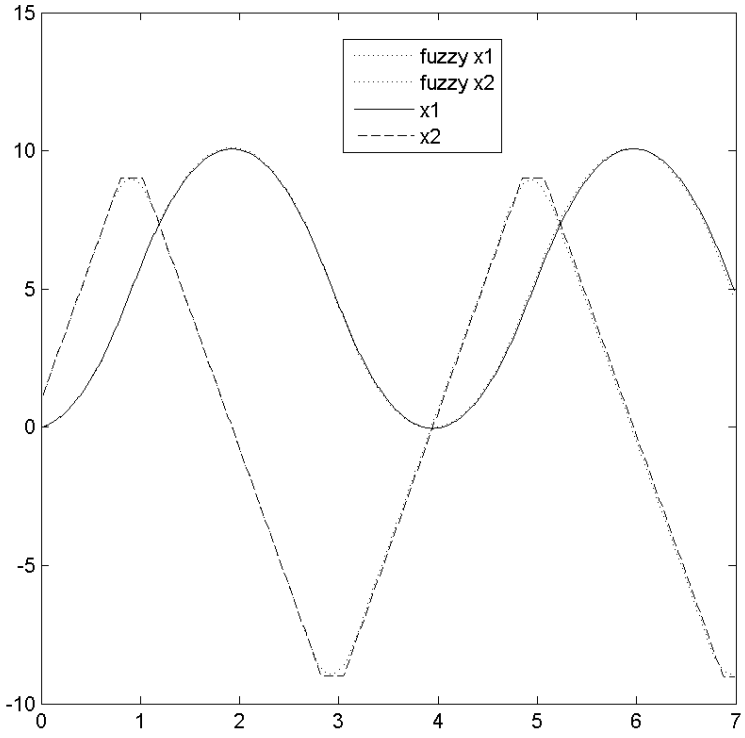

FIGURE 20. Propagation of *Fuzzy* system for the initial condition [0 1]

Using Matlab proved to be a good choice on one hand, on the basis of *Fuzzy* Logic packages and Genetic Algorithms, but a poor choice because the source codes are not available, and therefore it was not possible to evaluate whether the defuzzification might be optimized. Finally, if three inputs *Fuzzy* system is used, the rule matrix "*If-then*" is three-dimensional and, generally, for n inputs we have a matrix of n dimension. This brought several limitations to the methodology presented in this paper for it to be generalized for the general case of n inputs. In addition to this, few publication presents *Fuzzy* systems that were trained from genetic algorithms.

Acknowledgments. Initially, the authors of this article would like to thank for the excellent technical review and good written suggestions made by the reviewers of this paper. We would also like to thank the Technological Institute of Aeronautics (ITA), Casimiro Montenegro Filho Foundation (FCMF) and the 2RP Net Enterprise staffs for sponsoring this research.

REFERENCES

[1] S. Chen, S. A. Billings and P. M. Grant, Non-linear system identification using neural networks, *Int. J. Control*, vol.51, no.6, pp.1191-1214, 1990.

[2] M. Vidyasagar, *Nonlinear Systems Analysis*, Prentice-Hall, New Jersey, Networks Series, Electrical Engineering Series, 1978.

[3] L. Lapidus and J. H. Seinfeld, *Numerical Solution of Ordinary Differential Equations*, Academic Press, New York and London, 1971.

[4] N. E. Cotter, The Stone-Weierstrass and its application to neural networks, *IEEE Trans. Neural Networks*, vol.1, no.4, pp.290-295, 1990.

[5] G. Cybenko, *Continuous Valued Networks with Two Hidden Layers are Sufficient*, Technical Report, Department of Computer Science, Tufts University, 1988.

[6] K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, vol.2, no.5, pp.359-366, 1989.

[7] J. K. Hunt, D. Sbarbaro, R. Zbikowski and P. Gawthrop, Neural networks for control system – A survey, *Automatica*, vol.28, no.6, pp.1083-1112, 1992.
[8] J. S. R. Jang, C.-T. Sun and E. Mizutani, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice-Hall, Inc., 1997.
[9] R. P. Melo and P. M. Tasinaffo, Uma metodologia de modelagem empírica utilizando o integrador neural de múltiplos passos do tipo Adams-Bashforth, *Sociedade Brasileira de Automática (SBA)*, vol.21, no.5, pp.487-509, 2010.
[10] K. S. Narendra and K. Parthasarathy, Identification and control of dynamical systems using neural networks, *IEEE Trans. Neural Networks*, vol.1, no.1, pp.4-27, 1990.
[11] K. S. Narendra and A. U. Levin, Control of nonlinear dynamical systems using neural networks: Controllability and stabilization, *IEEE Trans. Neural Networks*, vol.4, no.2, pp.192-206, 1993.
[12] A. R. Neto, Dynamic systems numerical integrators in neural control schemes, *V Congresso Brasileiro de Redes Neurais*, Rio de Janeiro-RJ, Brazil, 2001.
[13] J. T. Spooner, M. Maggiore, R. Ordónez and K. M. Passino, *Stable Adaptive Control and Estimation for Nonlinear Systems Neural and Fuzzy Approximator Techniques*, Wiley-Interscience, New York, 2002.
[14] P. M. Tasinaffo, *Estruturas de Integração Neural Feedforward Testadas em Problemas de Controle Preditivo*, Ph.D. Thesis, Instituto Nacional de Pesquisas Espaciais -INPE, São José dos Campos, Brazil, 2003.
[15] P. M. Tasinaffo and A. R. Neto, Mean derivatives based neural Euler integrator for nonlinear dynamic systems modeling, *Learning and Nonlinear Models*, vol.3, no.2, pp.98-109, 2005.
[16] P. M. Tasinaffo and A. R. Neto, Predictive control with mean derivative based neural Euler integrator dynamic model, *Sociedade Brasileira de Automática (SBA)*, vol.18, no.1, pp.94-105, 2006.
[17] Y.-J. Wang and C.-T. Lin, Runge-Kutta neural network for identification of dynamical systems in high accuracy, *IEEE Trans. Neural Networks*, vol.9, no.2, pp.294-307, 1998.
[18] P. Shi, Y. Zhang, M. Chadli and R. Agarwal, Mixed H-infinity and passive filtering for discrete fuzzy neural networks with stochastic jumps and time delays, *IEEE Trans. Neural Networks and Learning Systems*, 2015.
[19] R. S. Guimaraes, V. S. Junior and P. M. Tasinaffo, Multipolar-valued fuzzy sets to deal with the cognitive ambiguities, *International Journal of Innovative Computing, Information and Control*, vol.11, no.6, pp.1965-1985, 2015.
[20] F. A. C. Gomide and R. R. Gudwin, Modelagem, controle, sistemas e lógica fuzzy, *Sociedade Brasileira de Automática (SBA)*, vol.4, no.3, 1994.
[21] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, Inc, New Jersey, 1992.
[22] L. Zadeh, Fuzzy sets, *Information and Control*, vol.8, pp.338-353, 1965.
[23] L. Zadeh, Fuzzy logic, *IEEE Computer Society Press*, vol.21, no.4, pp.83-93, 1988.
[24] L. I. Kuncheva, Initializing of an RBF network by a genetic algorithm, *Neurocomputing*, vol.14, pp.273-288, 1997.
[25] M. Mitchell, *An Introduction to Genetic Algorithms*, First MIT Press Paperback Edition, 1998.
[26] B. Kosko, Fuzziness vs. probability, *International Journal of General Systems*, vol.17, no.2, pp.211-240, 1990.
[27] E. H. Mamdani, Application of fuzzy algorithms for control of simple dynamic plant, *Proc. of the Institution of Electrical Engineers*, vol.121, no.12, pp.1585-1588, 1974.
[28] E. H. Mamdani, Applications of fuzzy logic to approximate reasoning using linguistic synthesis, *IEEE Trans. Computers*, vol.c-26, no.12, pp.1182-1191, 1977.
[29] T. Takagi and M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. System, Man and Cybernetics*, vol.SMC-15, no.1, pp.116-132, 1985.
[30] R. S. Guimarães, V. S. Júnior and P. M. Tasinaffo, Implementing fuzzy logic to simulate a process of inference on sensory stimuli of deaf people in an e-learning environment, *Computer Applications in Engineering Education*, vol.24, no.2, pp.320-330, 2016.
[31] P. M. Tasinaffo, R. S. Guimarães, L. A. V. Dias, V. S. Júnior and F. R. M. Cardoso, Discrete and exact general solution for nonlinear autonomous ordinary differential equations, *International Journal of Innovative Computing, Information and Control*, vol.12, no.5, 2016.