Journal of Advanced Computer Science and Technology, 3 (1) (2014) 61-89
©Science Publishing Corporation
www.sciencepubco.com/index.php/JACST
doi: 10.14419/jacst.v3i1.2138
Technical Paper

Modelizing a non-linear system: a computational efficient adaptive neuro-fuzzy system tool based on matlab

G. Bosque[1*], J. Echanobe[2], I. del Campo[2]

[1] *Department of Electronics Technology, University of the Basque Country, Bilbao, Vizcaya, 48013, Spain*
[2] *Department of Electricity and Electronics, University of the Basque Country, Leioa, Vizcaya, 48940, Spain*
**Corresponding author E-mail: guillermo.bosque@ehu.es*

Abstract

In a great diversity of knowledge areas, the variables that are involved in the behavior of a complex system, perform normally, a non-linear system. The search of a function that express those behavior, requires techniques as mathematics optimization techniques or others. The new paradigms introduced in the soft computing, as fuzzy logic, neuronal networks, genetics algorithms and the fusion of them like the neuro-fuzzy systems, and so on, represent a new point of view to deal this kind of problems due to the approximation properties of those systems (universal approximators).

This work shows a methodology to develop a tool based on a neuro-fuzzy system of ANFIS (Adaptive Neuro-Fuzzy Inference System) type with piecewise multilinear (PWM) behaviour (introducing some restrictions on the membership functions -triangular- chosen in the ANFIS system). The obtained tool is named PWM-ANFIS Tool, that allows modelize a n-dimensional system with one output and, also, permits a comparison between the neuro-fuzzy system modelized, a purely PWM-ANFIS model, with a generic ANFIS (Gaussian membership functions) modelized with the same tool. The proposed tool is an efficient tool to deal non-linearly complicated systems.

Keywords: *ANFIS model, Function approximation, Matlab environment, Neuro-Fuzzy CAD tool, Neuro-Fuzzy modelling.*

1. Introduction

The fusion of fuzzy logic and neural networks in Neuro-Fuzzy Systems (NFS) provides a model based on IF-THEN fuzzy rules with the capability of learning from samples. Fuzzy systems provide a framework to represent imprecise information and to reason with this kind of information, while neural networks enhance fuzzy systems with the capability of learning from input-output data; learning is used to adapt parameters of the fuzzy system as membership functions or rules. Some example representative application areas of NFS are: pattern recognition [1, 2], robotics [3, 4], nonlinear system identification [5, 6], adaptive signal processing [7, 8], etc.. In addition, new potential applications can be found in the field of ubiquitous computing and ambient intelligence [9]. These kinds of applications involve reasoning and learning algorithms that have to deal with signals from a large number of distributed sensor nodes [10, 11].

It can briefly be mentioned that one of the main reasons that influenced the success of the neuro-fuzzy systems

is their ability to approximate continuous nonlinear functions. In this area within the neuro-fuzzy networks, the following references are highlighted: [12, 13, 14, 15, 16, 17]

This paper presents a design methodology based on a new Matlab tool to develop computational efficient NFS, PWM-ANFIS Tool[1]. The Matlab tool, constructed by the authors, is the mainstay of the proposed methodology. It is a user friendly environment for modelling, analyzing and simulating computational efficient NFS applications. The proposed methodology is specially suited for real time applications that involve a large number of inputs. It deals with efficient implementations of a class of NFS, the adaptive neuro-fuzzy inference system (ANFIS) [19, 20], that has been widely used to develop NFS in the above application areas. ANFIS is a network representation of different types of fuzzy inference models, endowed with the learning capabilities of neural networks. In the last decade, ANFIS has become very popular, mainly due to the powerful capabilities as universal function approximator that it exhibits, even when simple membership functions like trapezes or triangles are used [12, 21, 22, 23, 24, 25, 26, 16, 27]. The present work focuses specifically on an ANFIS-like model that is functionally equivalent to the Takagi-Sugeno-Kang inference system [28, 29]. With regard to the computational cost, some different restrictions are applied to the system in order to reduce drastically the complexity of its inference mechanism which becomes a Piecewise Multilinear (PWM) function. In what follows will be refered this model as PWM-ANFIS.

The paper is structured as follows: Section 2 overviews the generic ANFIS model and introduces the PWM-ANFIS used in this paper; the computational efficiency is analyzed in order to highlight the advantages of the PWM model. In Section 3 the proposed methodology are described. The tool developed by the authors over Matlab and running on a PC are described in Section 4[2]. Section 5 presents some test examples and the results obtained. Section 6 outlines the main conclusion of this work. Finally, Appendices A and B show the algorithms realization.

2. Generic anfis model and pwm-anfis

In this section, first will be described the proposed PWM-ANFIS. This model is a NFS of the ANFIS type with some restrictions on its membership functions that lead to a very simple and rapid inference mechanism. These restrictions, as will be seen below, hardly affect the learning performance or the approximation capability of the system at all.

2.1. ANFIS model

First let us introduce the basics of the Generic ANFIS model [19], for the case of a zero-order Takagi-Sugeno-Kang inference system [12]. Consider a p-rules n-input, one-output fuzzy system:

$$R_j : IF\ x_1\ is\ A_{1j_1}\ and\ x_2\ is\ A_{2j_2} \ldots and\ x_n\ is\ A_{nj_n}\ THEN\ y\ is\ r_j, \tag{1}$$

where R_j is the jth rule $(1 \leq j \leq p)$, x_i $(1 \leq i \leq n)$ are input variables, y is the output, r_j is a constant consequent, and A_{ij_i} are linguistic labels (antecedents) with each one being associated with a membership function $\mu_{A_{ij_i}}(x_i)$, where $(1 \leq j_i \leq m_i)$ being m_i the number of antecedents of the variable x_i . In a zero-order Takagi-Sugeno-Kang fuzzy model the inference procedure used to derive the conclusion for a specific input $\mathbf{x} = (x_1, x_2, .., x_i, .., x_n)$, $\mathbf{x} \in \Re^n$, consists of two main steps. First the firing strength or weight ω_j of each rule is calculated as:

$$\omega_j = \prod_{i=1}^{n} A_{ij_i}(x_i). \tag{2}$$

After that, the overall inference result, y, is obtained by means of the weighted average of the consequents.

$$y = (\sum_{j=1}^{p} \omega_j \cdot r_j) / \sum_{j=1}^{p} \omega_j). \tag{3}$$

The Eq. (2) and (3) provide a compact representation of the inference model.

[1] A previous work of this tool was presented in [18]

[2] The tool can be discharged from www.ehu.es/guillermo.bosque

To generate a complete rule base, all the possible combinations of the antecedents must be taken into account, therefore, the number of rules p is calculated as:

$$p = \prod_{i=1}^{n} m_i. \tag{4}$$

ANFIS consists of a representation of different types of fuzzy inference models as adaptive networks. To be precise, the above fuzzy model can be viewed as an adaptive network with the following layers (see Fig. 1):

Layer 1. contains $\sum_{i=1}^{n} m_i$ neurons and computes the membership functions:

$$A_{ij_i}(x_i). \tag{5}$$

Layer 2. contains p neurons. Neuron j in this layer generates the firing strength of the $j-th$ rule by computing the algebraic product:

$$\omega_j = \prod_{i=1}^{n} A_{ij_i}(x_i). \tag{6}$$

Layer 3. is an p-neuron normalization layer. This layer performs the normalization of the activation of the rules; the output of the $j-th$ neuron is the ratio of the $j-th$ rules weight to the sum of the weights of all the rules:

$$\bar{\omega}_j = \omega_j / \sum_{k=1}^{p} \omega_k. \tag{7}$$

Layer 4. also contains p neurons. The neuron outputs gives the weights of the corresponding consequents:

$$y_j = \bar{\omega}_j \cdot r_j. \tag{8}$$

Layer 5. contains only one neuron. The neuron output is the weighted sum of the consequents:

$$y = \sum_{j=1}^{p} \bar{\omega}_j \cdot r_j. \tag{9}$$

Wll be useful further on modify the Eq. (9), extracting the sum of rules activation. This equation can be written as:

$$y = \frac{1}{\sum_{k=1}^{p} \omega_k} \sum_{j=1}^{p} \omega_j r_j. \tag{10}$$

To train the above network, a hybrid algorithm has been proposed in [12]. The algorithm is composed of a forward pass which is carried out by a Least Squares Estimator (LSE) process, followed by a backward pass which is carried out by a Back Propagation (BP) algorithm. In this work will be applied a variant of the BP algorithm that is the Gradient Descent Method (GDM). Finally concerning the computational cost of the above network, from Eq. (2) and (3), it can be deduced that the evaluation of an inference requires one division, $m^n(n-1)$ products, $2(m^n-1)$ sums and $m \cdot n$ membership evaluations, where it has been assumed that $m = m_1 = m_2 = \cdots = m_n$. This result will be used below when comparing with the proposed PWM system.

2.2. The proposed pwm-anfis model

In order to reduce the complexity of the above ANFIS model, let us introduce the following restrictions on the antecedents: (i) the membership functions are overlapped by pairs, (ii) they are triangular shaped, and (iii) they are normalized in each input dimension (see Fig. 2). Similar constraints have been successfully used by many designers because of the useful properties of triangular membership functions [30].

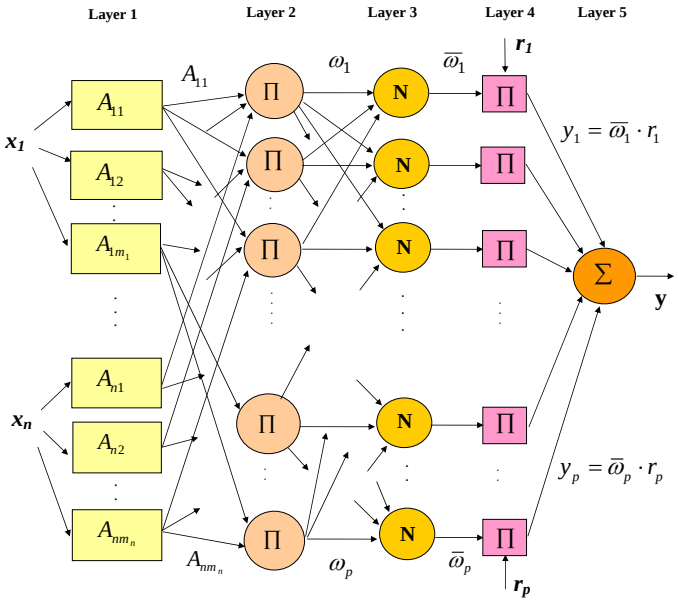


Figure 1: Architecture of a generic n-input ANFIS.

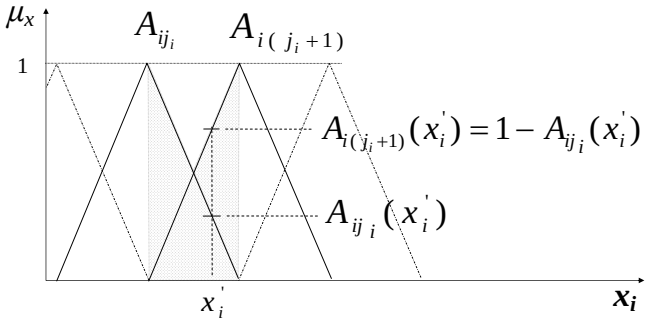


Figure 2: Triangular membership functions verifying constraints (i) to (iii).

In the following will be analyzed the advantages of constraints (i) to (iii) on the simplicity of the layered representation of the fuzzy system. First, let us consider some immediate consequences of these restrictions. The first restriction forces the overlapping degree of the antecedents to be two. Therefore, given an input vector $\mathbf{x} = (x_1, x_2, .., x_i, .., x_n)$ only two antecedents per input dimension provide membership values different from zero (i.e. active antecedents). To be exact, due to (ii) and (iii), only one half of the triangles concerned become active (see shadow region in Fig. 2). Note that there are both a membership value and its complementary value. For better understanding of these advantages, let us introduce a two dimensional example.

2.2.1. Two inputs anfis.

Fig. 3 depicts typical membership functions for a particular case of a 2-input system with m_i triangular antecedents per input ($m_1 = 5, m_2 = 5$) verifying the above constraints. It can also be seen in this figure that the vertex of the 5 triangles delimits 4 intervals per axis. Note that these intervals induce a partition of the input space into $(m_1 - 1) \cdot (m_2 - 1) = 16$ polyhedral (rectangles here) cells or regions.

The input vector $(x_1^{'}, x_2^{'})$ falls into the shaded rectangular region. This region defines 4 active rules, generated by the combination of membership functions $A_{12}(x_1^{'}), A_{13}(x_1^{'}), A_{23}(x_2^{'})$, and $A_{24}(x_2^{'})$, (see Eq. (6)), that is:

$$\omega_8 = A_{12}(x_1^{'}) \cdot A_{23}(x_2^{'})$$
$$\omega_9 = A_{12}(x_1^{'}) \cdot A_{24}(x_2^{'})$$
$$\omega_{13} = A_{13}(x_1^{'}) \cdot A_{23}(x_2^{'})$$
$$\omega_{14} = A_{13}(x_1^{'}) \cdot A_{24}(x_2^{'}).$$

The denominator in the Eq. (10) $\sum_{k=1}^{25} \omega_k$ is now:

$$A_{12}(x_1^{'}) \cdot A_{23}(x_2^{'}) + A_{12}(x_1^{'}) \cdot A_{24}(x_2^{'}) + A_{13}(x_1^{'}) \cdot A_{23}(x_2^{'}) + A_{13}(x_1^{'}) \cdot A_{24}(x_2^{'}).$$

Grouping terms and in virtue of the restriction (iii), having:

$$A_{12}(x_1^{'}) \left(A_{23}(x_2^{'}) + A_{24}(x_2^{'}) \right) + A_{13}(x_1^{'}) \left(A_{23}(x_2^{'}) + A_{24}(x_2^{'}) \right) = A_{12}(x_1^{'}) + A_{13}(x_1^{'}) = 1.$$

In consequence, $\sum_{k=1}^{25} \omega_k = 1$. Now, taking into account the Eq. (10), having:

$$y = \omega_8 \cdot r_8 + \omega_9 \cdot r_9 + \omega_{13} \cdot r_{13} + \omega_{14} \cdot r_{14}.$$

The above example illustrates how the system complexity has been reduced as a consequence of restrictions (i) to (iii). Note that only 4 terms in Eq. (10) are not null instead of the total 25 terms of the Generic ANFIS. In advance, will be considered that the membership functions, of a Generic ANFIS, are Gaussian functions.

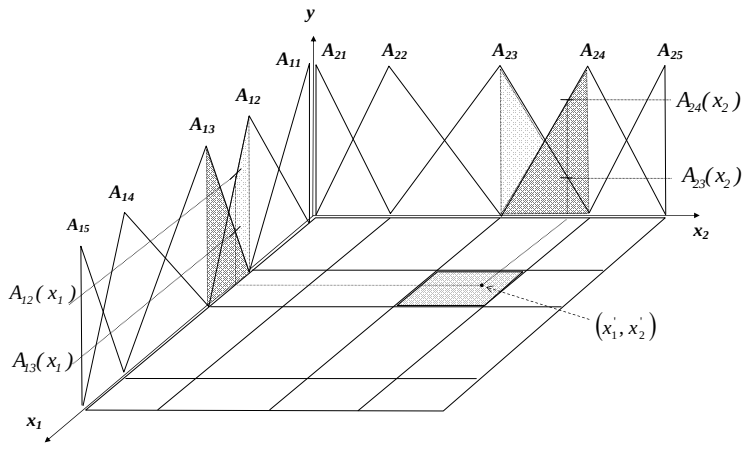


Figure 3: Triangular membership functions for an 2-input pwm-anfis case example.

2.2.2. Multi-variable anfis.

Let us now see the proposed model for n-inputs. Fig. 4 depicts triangular membership functions for a particular case of an n-input system with m_i antecedents per input ($m_1 = 7, m_2 = 5, .., m_n = 4$) verifying the above constraints. Like the above example, the vertex of the m_i triangles delimits $(m_i - 1)$ intervals per axis and these intervals induce a partition of the input space into $\prod_{i=1}^{n} (m_i - 1)$ n-dimensional polyhedral cells or regions. Only one of these cells is involved in the calculus of the system output at each time. The whole system can, therefore, be implemented as a single inference kernel. The parameters of the kernel depend on the concrete cell where the input vector falls (i.e. active cell). Let us illustrate in what way the restrictions imposed on the antecedent membership functions simplify the network architecture.

Layer 1. Every neuron in this layer computes one active membership function. In virtue of (i), each input is concerned with two membership functions per dimension. Therefore, there are only $2n$ active neurons (only one pair of complementary neurons per input) in this layer,

$$A_{12}(x_1),\ A_{13}(x_1),\ A_{23}(x_2),\ A_{24}(x_2), ..,\ A_{n2}(x_n),\ A_{n3}(x_n).$$

Layer 2. Each neuron in this layer generates a multilinear output which represents the firing strength of a rule; each multilinear term consists of the product of n linear terms as in Eq. (6). The rules with non-zero firing

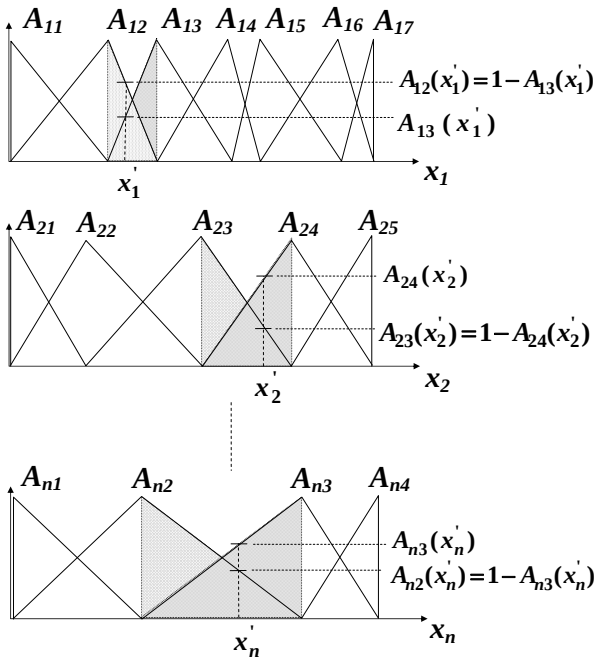


Figure 4: Triangular membership functions for an n-input pwm-anfis case example.

strength are only those associated with the active neurons in layer 1, that is, 2^n active rules. In the following will be used ω_k^* $(1 \leq k \leq 2^n)$ to denote the firing strength of the active rules.

$$\begin{aligned}
&\omega_1^* = A_{12}(x_1) \cdot A_{23}(x_2) \cdot \cdots \cdot A_{n2}(x_n) \\
&\omega_2^* = A_{12}(x_1) \cdot A_{23}(x_2) \cdot \cdots \cdot A_{n3}(x_n) \\
&\vdots \\
&\omega_{2^n-1}^* = A_{13}(x_1) \cdot A_{24}(x_2) \cdot \cdots \cdot A_{n2}(x_n) \\
&\omega_{2^n}^* = A_{13}(x_1) \cdot A_{24}(x_2) \cdot \cdots \cdot A_{n3}(x_n).
\end{aligned} \tag{11}$$

Layer 3. Taking into account constraint (iii), it can easily be proved that the normalization layer disappears because the division is unnecessary [20].

$$\sum_{k=1}^{2^n} \omega_k^* = 1. \tag{12}$$

Layer 4. The neuron outputs produce the weight of the consequents. Will be used r_k^* $(1 \leq k \leq 2^n)$ to denote the consequents associated with the active rules.

$$y_k = \omega_k^* \cdot r_k^*. \tag{13}$$

Layer 5. Finally the output layer is reduced to the sum of 2^n product terms,

$$y = \omega_1^* r_1^* + \omega_2^* r_2^* + ... + \omega_{2^n-1}^* r_{2^n-1}^* + \omega_{2^n}^* r_{2^n}^*. \tag{14}$$

In sum, the main benefits of constraints (i) to (iii) on the general ANFIS architecture are a reduction of the number of neurons per layer (layers 1 and 2) due to the activation of a reduced number of antecedents, the elimination of the normalization layer, and a simplification of the network arithmetic. Fig. 5 shows this reduction.

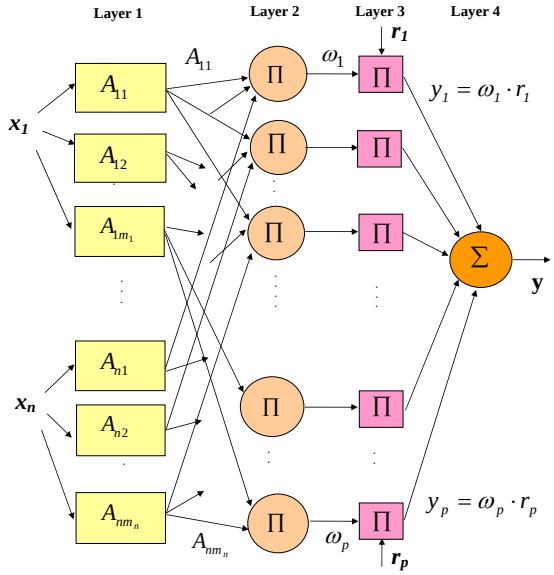


Figure 5: Architecture of a n-input pwm-anfis model.

2.2.3. Computational cost of a pwm-anfis.

With the above restrictions, a total of $(2^n+1)n$ products and 2^n+2n-1 sums is required to perform an inference. This cost is considerably less than that required by the generic ANFIS and it does not depend on the number of membership functions m. Moreover, as n and m increase, the difference between both costs (Generic ANFIS and PWM-ANFIS) becomes extremely large. This fact can be appreciated in Table 1, where the cost for both systems is depicted for some different n and m values; clearly the difference is very good.

Table 1: Comparison of the computational cost. p: products; s: sums; me: membership evaluations; d: divisions. Numbers in parenthesis mean that the operations involved are already included in the sums and/or products

		PWM-ANFIS	**Generic ANFIS**		
			m=4	m=5	m=6
n=2	p	10	31	49	71
	s	7	30	48	70
	me	(4)	8	10	12
	d	-	1	1	1
n=4	p	68	1023	2499	65183
	s	23	510	1248	2590
	me	(8)	16	20	24
	d	-	1	1	1
n=6	p	390	24575	93749	279935
	s	75	8190	31248	93310
	me	(12)	24	30	36
	d	-	1	1	1

2.3. Learning algorithm

Concerning the learning procedure, the advantages of restrictions (i) to (iii) are twofold. Firstly, as has been seen, the PWM-ANFIS limits the activation of the system each time to a single cell or region of the input space. This cellular nature of the feed-forward network (fuzzy inferences) also reduces the computational complexity of the learning algorithm because both LSE and GDM equally require the evaluation of the feed-forward network. Secondly, the constraints imposed on the input partition reduce not only the active set of parameters for a concrete input (parameters of the active cell), but also the total set of antecedent parameters.

Next let us define the parameters over which the learning process will be applied. The PWM-ANFIS network has two kinds of parameters, namely antecedent and consequent, the antecedent parameters are the coordinates that define the triangles (see Fig. 6) and the consequent parameters are the constant consequents of the zero-order Takagi-Sugeno-Kang system (see Eq. (1) and Eq. (10)).

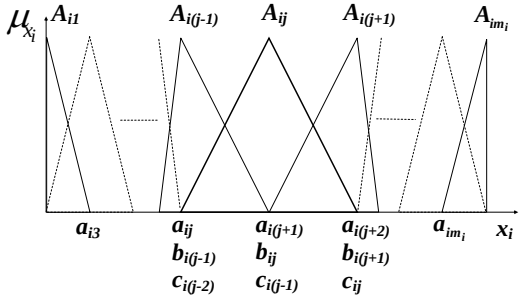


Figure 6: Coordinates of n membership functions and coordinates for the i-th input pwm-anfis case example.

The parameters to be trained are $\{(a_{ij}, b_{ij}, c_{ij}), 1 \leq i \leq n, 1 \leq j \leq m_i\}$ for the antecedents and $\{(r_k), 1 \leq k \leq K\}$ for the consequents. The partition of the input space is completely defined by means of the triangle coordinates (a_{ij}, b_{ij}, c_{ij}) (see Fig. 6), and then, choosing any intermediate triangle A_{ij}, having that $a_{ij} = b_{i(j-1)} = c_{i(j-2)}$. Therefore a partition of m_i triangles is completely defined by means of $m_i - 2$ parameters, because the center of the first and the last triangles of the partition match with the bounds of the input universe. The first bound is $a_{i1} = b_{i1} = a_{i2}$ and last bound is $b_{im_i} = c_{im_i} = c_{i(m_i-1)}$. For example, a 2-input system with a partition of the input space like that depicted in Fig. 3 ($m_1 = m_2 = 5$), requires the adaptation of only 3 antecedent parameters, per variable, (a_{13}, a_{14}, a_{15} and a_{23}, a_{24}, a_{25}) against the 15 parameters, per variable, required to define unconstrained symmetrical or asymmetrical triangles.

Concerning the consequent parameters, the number of crisp consequents to be adapted is equal to the total number of rules (see Eq. (10)) where all possible rules are considered in order to guarantee the completeness of the rule set.

Although a basic back-propagation learning rule can be used to adapt the set of network parameters, the learning process generally becomes too slow. To avoid this problem, the hybrid learning rule proposed in [19], which combines the GDM and the LSE, is used. Each epoch of the hybrid procedure is composed of a forward pass and a backward pass. In the forward pass the consequent parameters are identified by the LSE [14] method and in the backward pass the antecedent parameters are updated by the GDM. Let us briefly present both the LSE and the GDM as applied to the PWM-ANFIS system. Eq. (10) can be written as

$$\mathbf{y} = \omega_1(\mathbf{x}) \cdot r_1 + \omega_2(\mathbf{x}) \cdot r_2 + ... + \omega_p(\mathbf{x}) \cdot r_p, \tag{15}$$

where $\mathbf{x}$ is the vector of input variables. Since $\mathbf{y}$ is linear in the consequent parameter, r_j, and given values of the antecedent parameters, and a set of training data pair $\{(x_k; y_k), 1 \leq k \leq K\}$, each training pattern can be substituted into Eq. (15) to obtain a set of K linear equations.

After the consequent parameters have been identified, the GDM is used to optimize the shape of the m_i triangles in the partition of the $i-th$ input. Consider the error function:

$$E = \frac{1}{2K} \sum_{k=1}^{K} \left(y_k - y_k^{'}\right)^2, \tag{16}$$

also called SSE (Sum Square Estimator), where y_k is the actual output of the network for the $k-th$ training data and $y_k^{'}$ is the desired output. In the off-line operation mode the parameter update is performed after the evaluation of all the training patterns. The learning rule for the triangle coordinates, a_{ij_i}, with $3 \leq j \leq m_i$ (that is $m_i - 2$ parameters), is

$$a_{ij_i}^{new} = a_{ij_i} + \eta_i \cdot \frac{\partial E(a)}{\partial a_{ij_i}}, \tag{17}$$

where a_{ij}^{new} is the parameter a_{ij} updated, and η_i is the learning rate for the parameters of the $i-th$ input. The chain rule has been used to calculate the above derivative (see Appendix A) with:

$$\frac{\partial E(a)}{\partial a_{ij_i}} = \frac{1}{K}\sum_{k=1}^{K}\left[(y_k - y_k^{'})\cdot\left(FactorA_{ij_i}\frac{\partial A_{ij_i}}{\partial a_{ij_i}} + FactorA_{i(j_i-1)}\frac{\partial A_{i(j_i-1)}}{\partial a_{ij_i}} + FactorA_{i(j_i-2)}\frac{\partial A_{i(j_i-2)}}{\partial a_{ij_i}}\right)\right]. \tag{18}$$

The Factors ($FactorA_{ij}$) are formed by sums and products (see Appendix A).

The learning rule Eq. (17), (18) consists of simple arithmetic operations, subtractions and the sum of products that can be efficiently implemented as SW using a microprocessor or a DSP. Note that, in addition to the aforementioned advantages, in the system proposed only one kind of parameter is to be trained in the backward stage, namely, the triangle a_{ij} coordinate, while in the generic model at least two different types of parameters are involved (e.g. center and width). The above hybrid learning rule is suitable for off-line learning and also for on-line learning with minor modifications [14].

2.3.1. Modifications for on-line learning

The learning takes account only of one pair of training points,

$\{(\mathbf{x}_k = \left(x_1^{'}, x_2^{'}, ..., x_n^{'}\right); y_k), 1 \leq k \leq K\}$, and the parameters are updated with each couple. This process is known as Pattern-by-Pattern Learning.

Now referring to the Eq. (14)

$$y_k = \omega_1(\mathbf{x}_k)\cdot r_1 + \omega_2(\mathbf{x}_k)\cdot r_2 + \cdots + \omega_p(\mathbf{x}_k)\cdot r_p = (\omega_1(\mathbf{x}_k)\ \omega_2(\mathbf{x}_k)\ \cdots\ \omega_p(\mathbf{x}_k))\cdot\mathbf{r} = \mathbf{a}_k^T\cdot\mathbf{r}. \tag{19}$$

where $\mathbf{a}_k$ is the array of antecedents for $\mathbf{x}_k$.

The LSE is applied recursively, taking account of the time-varying characteristic of the incoming data. The recursive formula obtains an array $\mathbf{P}_1$ [14]. $\mathbf{P}_1$ is calculated with $\mathbf{P}_0$, $\mathbf{a}$, $\mathbf{a}^T$ and a factor λ (forgetting factor). $\mathbf{P}_1$ is calculated using $\mathbf{P}_0$, which corresponds to the previous pair of training points. $\mathbf{P}_0$ is initialized before begin the learning process and take a value $\mathbf{I}$ (matrix identity).

$$\mathbf{P}_1 = \frac{1}{\lambda}\left(\mathbf{P}_0 - \frac{\mathbf{P}_0\cdot\mathbf{a}\cdot\mathbf{a}^T\cdot\mathbf{P}_0}{\lambda + \mathbf{a}^T\cdot\mathbf{P}_0.\mathbf{a}}\right). \tag{20}$$

With $\mathbf{P}_1$, the new consequents are calculated in an incremental mode. New consequents $\mathbf{r}_{k+1}$ are calculated by adding to $\mathbf{r}_k$ (the array of previous consequents and the index k denote the incremental step) an incremental value [14].

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \mathbf{P}_1\cdot\mathbf{a}\cdot\left(y^{'} - \mathbf{a}^T\cdot\mathbf{r}_k\right). \tag{21}$$

Note that the term $\mathbf{a}^T\cdot\mathbf{r}_k$ corresponds to the actual output $y_k = \mathbf{a}^T\cdot\mathbf{r}_k$. The new output will be:

$$y_{k+1} = \mathbf{a}^T\cdot\mathbf{r}_{k+1}. \tag{22}$$

With respect to the GDM there is a little difference between on-line and off-line learning methods. In the on-line case, the error E seen in the Eq. (16) is only calculated with the current couple, that is, $E = \frac{1}{2}(y_k - y_k^{'})^2$ [14].

3. A design methodology based on the pwm-anfis tool

This section presents the design methodology and the PWM-ANFIS Matlab tool developed by the authors. The proposed methodology provides a compact design flow that encompasses all the steps involved in the development

of NFS. The tool allows the designer to define multivariable systems without limitations either in the number of membership functions or in the number of rules. Triangular and Gaussian membership functions are available to model both PWM-ANFIS systems and generic ANFIS systems, respectively [18].

For purposes of analysis, the tool obtains the *SSE* (see Eq. (16)) between the target function and its corresponding approximations during the learning process. After training it can also provide the Generalized Sum Square Error (*GSSE*), which is the *SSE* but evaluated with a collection of non-training data. These error parameters give measures about the approximation capability of the developed system. In addition, the tool generates the object code for developing both on-line and off-line training applications. The tool has been tested extensively by means of several nonlinear functions [31] and it has been used to develop efficient SW solutions, high performance HW solutions, and hybrid HW/SW approaches [32, 33, 34].

The simulation process takes advantage of the Matlab resources, fundamentally in matrix treatments of complex numerical calculations, to give a high speed response. The most popular debugging facilities of the Matlab environment are also available to refine the system designs. The tool also provides a user-friendly graphic interface to monitor the evolution of learning processes.

In the next subsection will be presented the developed environment, highlight between off-line and on-line learning mode and, finally, the tool interface over a PC.

3.1. The pwm-anfis environment

Fig. 7 shows a block diagram of the PWM-ANFIS Tool. The main block is a Matlab program that gives support for both off-line learning and on-line learning applications.

1. In the off-line learning mode the program accepts the following files: a file that contains the parameters of the process, a file that contains the Input/Output training data pairs, and a file that contains test data pairs (optional).

2. In the on-line learning mode, the tool is able to interface with a plant model in a closed-loop configuration, as can be seen in Fig. 7. In this operation mode the program accepts the file that contains the parameters of the process while the I-O data pairs are provided by the plant.

The parameters of the process are the number of antecedents for each variable, the learning rate, the target error, and the maximum number of iterations.

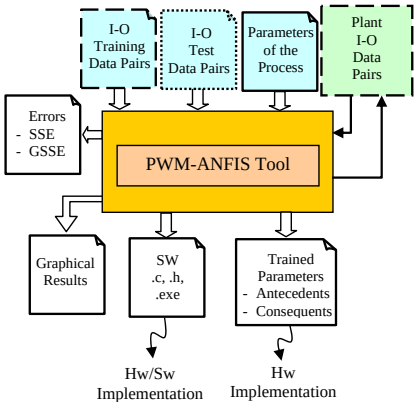


Figure 7: Block diagram of the pwm-anfis environment.

The first step in the development cycle consists in loading the parameters of the process. Then the tool extracts the system size and generates an initial (non-trained) PWM-ANFIS and the training process commences. In the off-line mode, the training data pairs are read from the text file, while in the on-line mode, they are provided by the plant. Meanwhile, the user is able to monitor the evolution of the SSE. The training process finishes when either the specified error is achieved or the maximum number of iterations is reached. If the GSSE option is enabled, the test data file is loaded and the GSSE is calculated. In addition, the graphical interface provides a representation of both the trained function and the target function. Moreover, the membership functions (initial and final triangle partitions) are represented.

As a result of the trained process the PWM-ANFIS Tool generates a text file with the trained parameters (antecedents and consequents) which can be exported to develop different kinds of implementations. In addition, Matlab is able to compile the m-file program (script) to provide C code. Hence, an executable file can be implemented, for instance, on a microprocessor, a digital signal processor (DSP), or a System-on-a-Programmable Chip (SoPC).

3.2. Functional description of the matlab program

The program has been structured into the functional steps shown in Fig. 8.

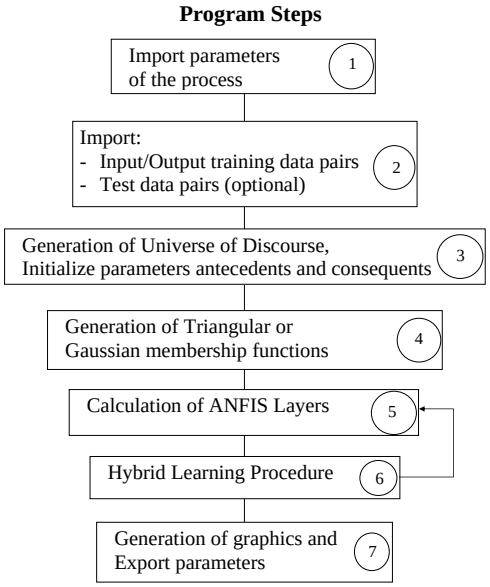


Figure 8: Program steps.

- **Step 1.** Import the Parameters of the Process. The tool imports the file that contains the parameters of the process such as the number of antecedents m_i, the learning rate η for the variables, the error target SSE_t, the maximum number of iterations and the indication about the existence of the non-training data pairs to calculate the $GSSE$.

- **Step 2.** Import the Input/Output samples. In this step the file that contains the Input/Output training data pairs and the file that contains non-training data pairs (optional) are imported. After training, the $GSSE$ is obtained. This error parameter gives measures about the approximation capability of the developed system.

- **Step 3.** Generation of universe of discourse and calculate the parameters of the membership functions. The program extracts the universe of discourse of the input and output variables. In addition it assigns the initial parameters of the antecedents and consequents.

- **Step 4.** Generation of Membership Functions. Generation the array of membership functions, A_{ij_i}, over the universe of discourse. In the case of triangular membership functions the program takes into account the restrictions (i)-(iii) that characterize the PWM-ANFIS developed in this work.

- **Step 5.** Calculation of ANFIS Layers. The tool applies the algorithm, namely Algorithm of Rule Activations (see Appendix B) to implement the Eq. (11) and Eq. (14). This algorithm is divided in two sub-steps. The first one obtains the array of antecedents index that is the current combination of the membership functions, and the second calculates the product of those membership functions to obtain the weight of the rules ω_k^* (see Eq. (11)), the contribution of each rule y_k (see Eq. (13)), and the output value y (see Eq. (14)).

- **Step 6.** Hybrid Learning Procedure. In this step will be used the array of antecedents index obtained in the previous step to calculate several factors ($FactorA_{ij}$ -see Eq. (18) and Appendix A-). The learning procedure

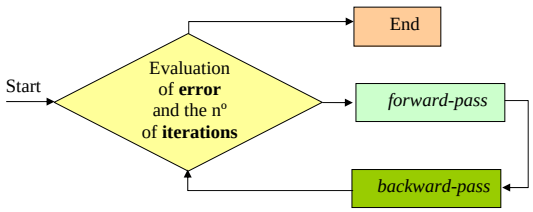


Figure 9: Simplified diagram of the hybrid learning algorithm.

presents some differences between Off-Line and On-Line learning. The procedure finishes by calculating the new output $\mathbf{y}$. The Fig. 9 shows schematically the learning process.

1. **Off-Line - Hybrid Learning.** The learning covers the full set of training points. The process consists of nine functional steps and is repeated for several iterations. The process can finalize in two cases, a) if the number of iterations arrives at the target, b) if the error is less than the target. Fig. 10(a) shows the steps for this.
2. **On-Line - Hybrid Learning.** The learning takes into account only one couple of training points $\left\{\left(\mathbf{x}_k = \left(x_1^{'}, x_2^{'}, ..., x_n^{'}\right); y_k\right), 1 \leq k \leq K\right\}$. The process consists in eleven functional steps. In this tool, the iterations take place once the pairs have been taken into account and finalize with the same conditions as in the off-line process. Fig. 10(b) shows the steps for this algorithm.

- **Step 7.** Generation of Graphics and Export Parameters. Finally, the program displays the results in several graphics (evolution of SSE and the value of $GSEE$) and exports a file with the new parameters (antecedents and consequents) of the trained system.

4. The pwm-anfis tool running on a pc

Taking into account that the Matlab program needs to work with several files to be built by the user; a java-written interface allows to build such files easily and to launch the Matlab program. Firstly, the interface allows to select between two ANFIS models, PWM-ANFIS and Generic ANFIS. The first one works with triangular membership functions as we have seen at length previously, and the second one, with free Gaussian membership functions. The last model make it possible to compare the approximation capability of the PWM-ANFIS versus the Generic ANFIS.

The tool allows working with continuous variables or binary input. The objective function can also be continuous or binary. Has set the limit of at least one input variable must be continuous. Fig. 11 shows the tool interface.

Has been introduced a restriction in this version of the tool, the PWM-ANFIS Tool works in off-line mode.

For methodology of work will be followed the next steps:

4.1. Selecting the model:

The tool lets choose between PWM-ANFIS models and Generic ANFIS model. Fig. 11 shows the selection in the top center panel.

4.1.1. PWM-ANFIS

This model is the result of introducing a number of restrictions in the ANFIS model. The membership functions are triangular, normalized and overlapped by pairs, the Fig. 12(a) shows an example with four and seven membership functions for two input variables. These restrictions allow considerably reduce the computational cost of the modeling process and to simplify implementation hypothetical Hardware (HW) of the approximate network [35, 32, 34].

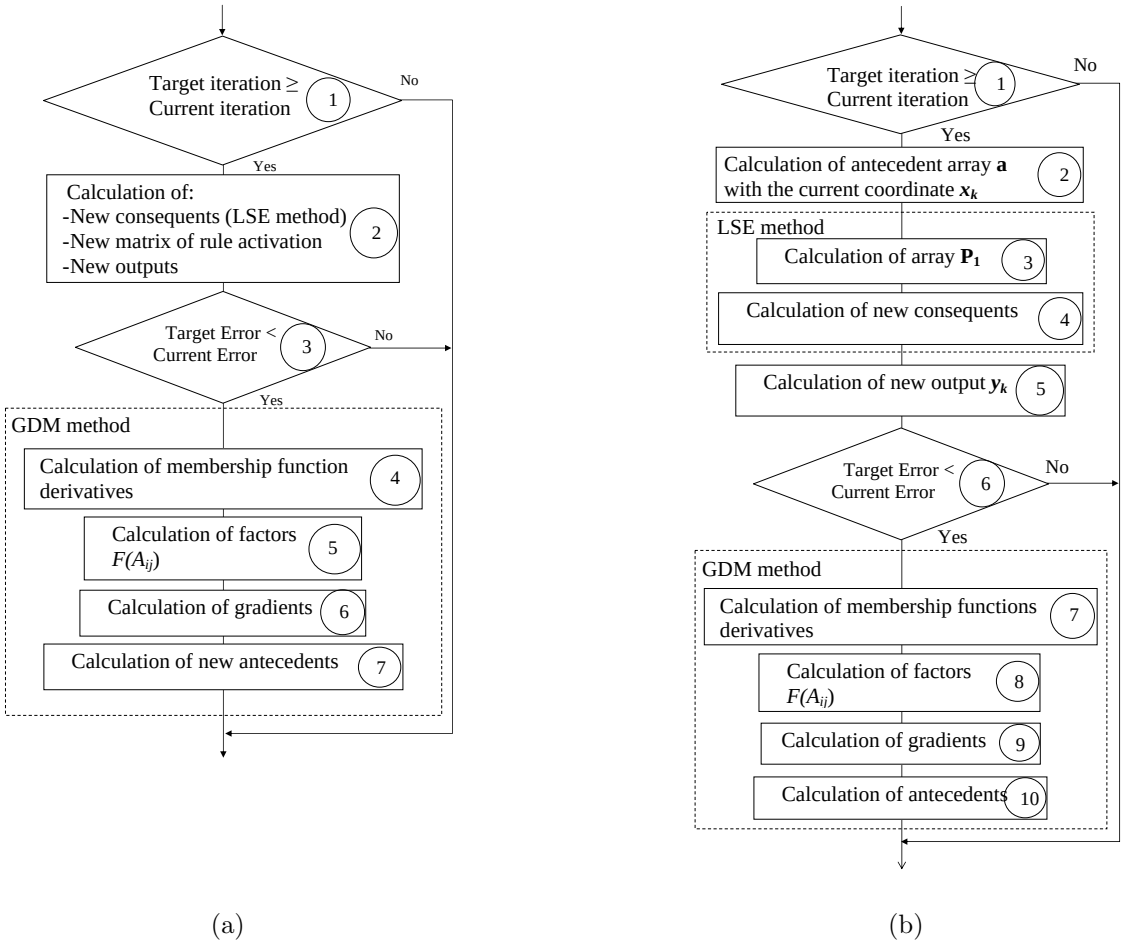


Figure 10: (a) Off-line procedure, (b) On-line procedure.

This choice shown in Fig. 13 the table -PWM-ANFIS Parameters- with specific parameters of this model.
Fig. 14 shows the relationship between the coordinates that define the triangles of the membership functions A_{ij} (input variable i - antecedent j), taking into account the above restrictions. Due to the restrictions will have $a_{ij} = b_{i(j-1)} = c_{i(j-2)}$, being the first boundary of the triangles $a_{i1} = b_{i1} = a_{i2}$ and the last limit $b_{im_i} = c_{im_i} = c_{i(m_i-1)}$.

4.1.2. Generic ANFIS

This model works with Gaussian membership functions free, ie, in the learning process each membership function can move freely. The initial Gaussian are generated equidistant. Fig. 12(b) shows this selection for five and three membership functions.
This choice shown in Fig. 15 the table -Generic ANFIS Parameters- with specific parameters of this model .
Fig. 16 shows the two parameters, σ_{ij} and m_{ij} (input variable i - antecedent j) which define a Gaussian membership function A_{ij}.

4.2. General parameters:

The general parameters are valid for both models (PWM-ANFIS and Generic ANFIS). The panel -General Settings- shows three parameters to adjust: Number of variables, target Error (SSE) and Maximum N of iterations.

1. Number of input variables: The tool can model functions from 2 to 8[3] input variables (x_1, x_2 , . . . , x_8). Using a drop-down list will select the number of variables involved in the function.

[3]This constraint can be modified

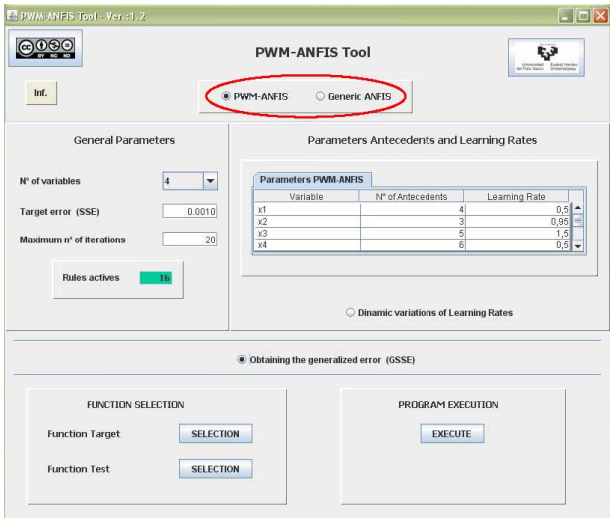
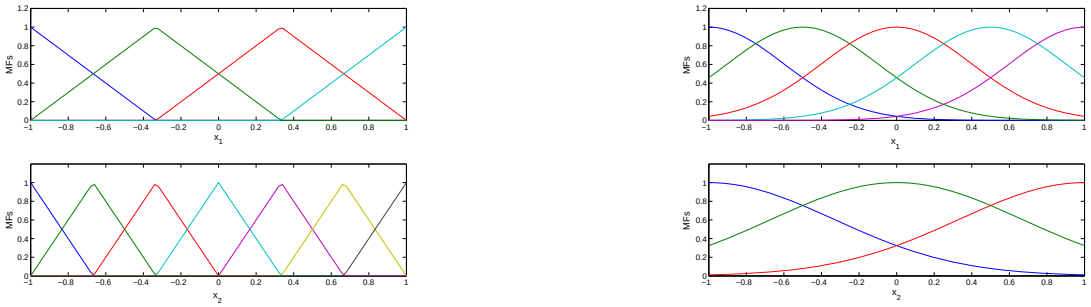


Figure 11: Interface of pwm-anfis tool.



(a) (b)

Figure 12: Membership functions (a) Triangulars, (b) Gaussians.

2. Target error SSE (Sum Square Error): Introduces the minimum error to be achieved during the training network process. The error is calculated according to the expression 16, where y_k represents the current network output for the *k-th* training data and $y_k^{'}$ is the desired output (target) and K is the set of data training pairs $\left\{(\mathrm{x}_k; y_k^{'}), 1 \leq k \leq K\right\}$, being $(\mathbf{x}_k = (x_1, x_2, ..., x_8)\,; y_k^{'})$.

3. Maximum number of iterations of the learning process: The maximum number of iterations to be performed by the learning algorithm to trained the network. If the error SSE is reached before finalizing the maximum iterations, the process will finish.

The change of the model, keeping the same general parameters, is interesting when a comparison of the approximation achieved by the two models is desired. Modelling ends when the error has been reached or SSE has reached the number of iterations introduced. Fig. 13 shows the PWM-ANFIS option and the corresponding parameters table. Fig. 15 shows the ANFIS Generic option and the corresponding parameters table.

4.3. Antecedent parameters and learning rates:

Next will be chosen parameters relating to the antecedents (membership functions) and learning rates for these parameters. Are chosen the number of antecedents per input variable. In the case of PWM-ANFIS model will be equidistant triangles and in the case of the ANFIS Generic model will be equidistant Gaussian membership functions. Learning rates have the following characteristics:

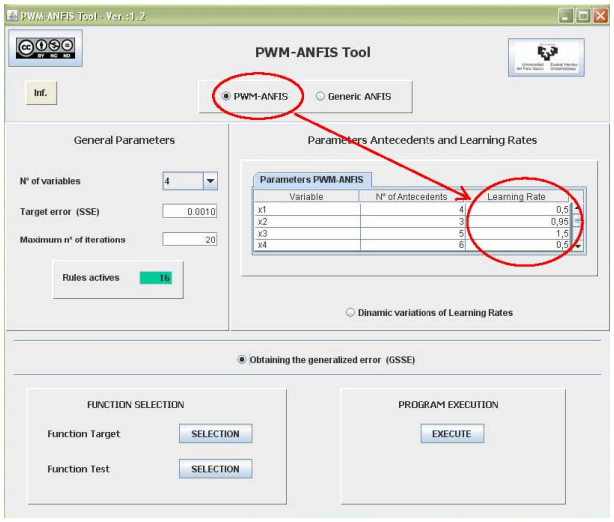


Figure 13: Interface of pwm-anfis tool: PWM-ANFIS.

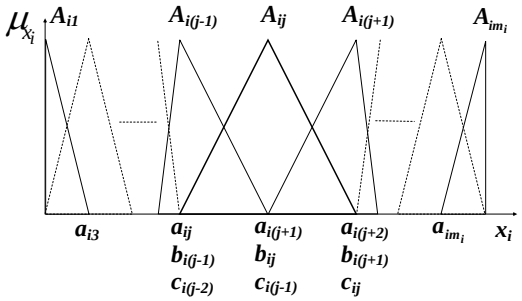


Figure 14: Membership functions and coordinates to the i-th input of a system PWM-ANFIS.

4.3.1. PWM-ANFIS model:

1. **Number of antecedents**: Introduces the number of background triangular membership functions for each variable. The minimum number of antecedents is 3. At runtime, if the tool detects that any variable is binary, assigns 2 antecedents as can be seen in Fig. 17 with a function of 4 input variables, being binary two (x_2 and x_3).

2. **Learning rate**: A learning rate, η, to each variable is assigned, which will mark the evolution of the triangles coordinates. Is introduced an initial estimated value for each variable (see Fig. 13). The rates can be adjusted experimentally by the user to determine which are the best for the learning process. Rates affecting the first coordinate of the triangles (a_{ij}). The minimum allowable rate is 0.0001.

4.3.2. Generic anfis model:

1. **N of antecedents**: Introduces the number of background Gaussian membership functions for each variable. The minimum number of antecedents is 3. If the tool detects that any variable is binary at runtime, assigns 2 antecedents as can be seen in Fig. 18 with a function of 4 input variables, being binary two (x_2 and x_3).

2. **Learning rate**: To be Gaussian membership functions, there are two learning rates for variable, the first applied to the evolution of σ, width of the Gaussian, and the second to changes in m, coordinate of the Gaussian center (see Fig. 15).

   (a) Rates σ: Rates of σ affect the width of the Gaussian.

   (b) Rates m: Rates of m affect the position of the Gaussian.

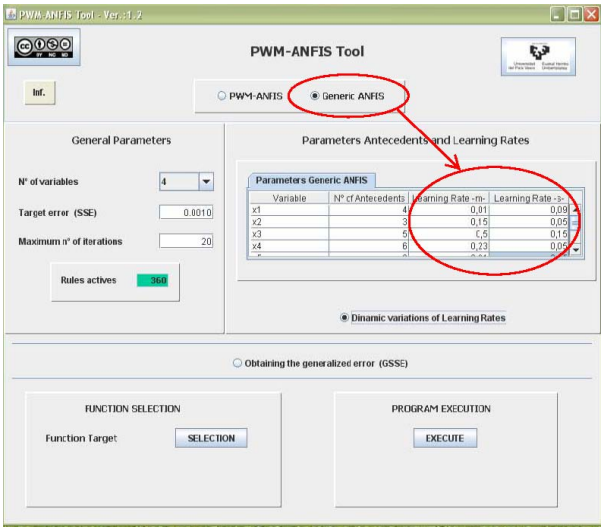


Figure 15: Interface of pwm-anfis tool: generic anfis.

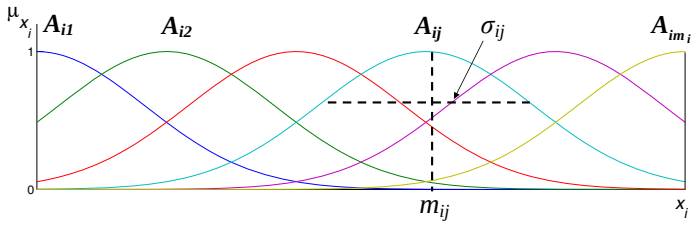


Figure 16: Membership functions and the parameters to the i-th input of a Generic ANFIS.

Introduces an initial estimated value for each variable. Rates can be be adjusted experimentally by the user to determine which are best for the learning process.

In both parameters the minimum rate is: 0.0001.

Will be highlighted that the modeling will be started with low rates of learning, due to the potential instability that can lead to excessively high rates. This is an area that shows highly empirical.

4.3.3. Dynamic variations of the rates:

The tool allows the user to try modeling with fixed learning rates or dynamically variable, ie, in the second case adapted to increase or decrease the error during the iterations according to the new approach (evolution of the error), the rates remain unchanged, increase or decrease.
The dynamic variation of the rates is of interest in trials with a high number of iterations. The executable program examine the rates every 6 iterations, increasing slightly if the error rate is equal *SSE*, increasing the rate more than in the previous case if the error *SSE* has diminished or decreasing the rate slightly if the error *SSE* has increased. Fig. 19 shows the selected option.

4.4. Obtaining the generalized error:

As indicated previously, the tool modeling a nonlinear function by comparison with a target function. Modeling involves a learning process performed during a series of iterations. After each iteration, the result is compared with the target function, that generates an error *SSE* and therefore evolution of the error curve along the iterations scheduled.
At the end of the learning process the tool allows calculate the named generalized error *GSSE* (Generalized Sum Square Error) made a comparison between the approximated network (model) and a test function whose points do not coincide with those of the target function (training points). Fig. 20 shows the choice of calculating the GSSE and the effect of this election shows the select button function test.

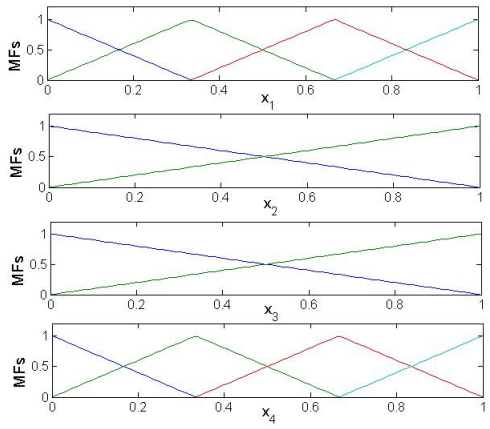


Figure 17: Membership functions of continous and binary triangular variables.

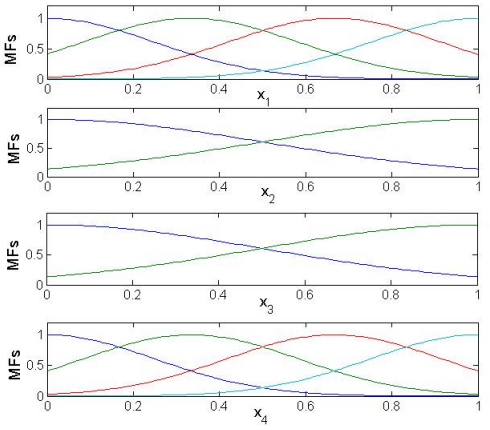


Figure 18: Membership functions of continous and binary Gaussian variables.

4.5. Parameters file created with the parameters chosen

When the parameters has been introduced, the tool generates a parameters file which will be read by the executable program. Each model generates a different file.

1. **PWM-ANFIS model file:** The generated file is named *parameters_tri*. Table 2 (a) shows the generated file for this model. The example shows that the GSSE error has been chosen (1) and has not been chosen the dynamic learning rate (0).

2. **ANFIS Generic model file:** The generated file is named *parameters_gauss*. Table 2 (b) shows the generated file for this model. The example shows that the GSSE error has not been chosen (0) and has been chosen the dynamic learning rate (1).

4.6. Selection of the target Function and the test Function:

The next step consist in choosing the target function and the test function, if the last is available. These functions must have a certain format as will be explained forward. Are files created by the user and which can be termed at the discretion of the user.
Both target and test functions present a matrix structure where each row is a pair $\left\{(\mathrm{x}_k; y_k^{'}), 1 \leq k \leq K\right\}$, being $(\mathbf{x}_k = (x_1, x_2, ..., x_8)\,; y_k^{'})$. The coordinates can be ordered or not, there are only one restriction: the columns must be in order, that is, the first column corresponds to variable x_1, the second column corresponds to the second variable x_2, etc, therefore the minimum number of columns is $(x_1, x_2, y^{'})$ and the maximum is 9 $(x_1, x_2, \ldots, x_8, y^{'})$.

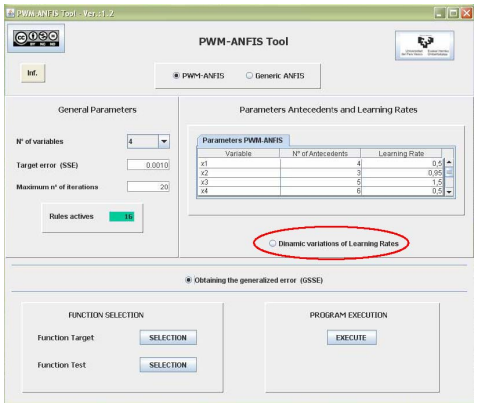


Figure 19: Option of dynamic learning rates.

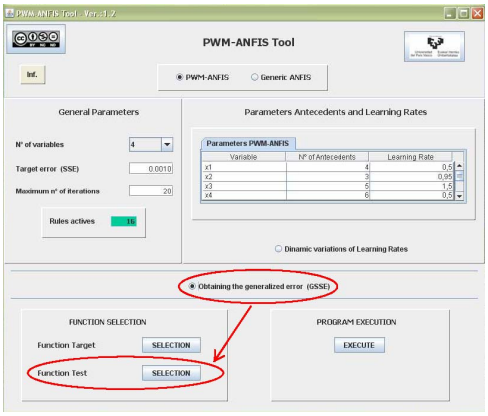


Figure 20: Option of generalized error *GSSE*.

Fig. 21 shows the selection buttons of the target function and the test function. As has been mentioned earlier, the function test can be extracted of the target function, provided that will be available a large number of target points. Table 3 shows the different coordinates associated to the target value by rows and the variables order by columns. The last column corresponds to the target value.

4.7. Program execution

Finally, proceed with the execution of the program that performs the modeling and training the neuro-fuzzy net. The program is an executable of the program realized in Matlab environment.

5. Application of the development methodology to some nfs case examples

5.1. Approximation for off-line training

In this section, the proposed PWM-ANFIS Tool will be used to develop some NFS examples. The examples deal with the approximation of non-linear functions. In particular, in the first, second and third example, some functions that are commonly used as test examples in related works are selected [14, 36]. All these examples clearly illustrate the proposed methodology and they also show how the PWM-ANFIS has good approximation capability and learning performance despite the imposed restrictions.

5.1.1. Two-variables.

In this example the approximation of a polynomial 2-input function is presented. The target function is $y = \frac{sin(x_1)\cdot sin(x_2)}{x_1 \cdot x_2}$. This function is graphically shown in Fig. 22(a) where the universes of discourse have been normalized,

Table 2: Parameters file (a) PWM-ANFIS Model *parameters_tri*, (b) Generic ANFIS Model *parameters_gauss*.

Position	Meaning	Example
0	Max. n of Iterations	25
1	*SSE Target Error*	0.0005
2	*GSSE* Error	1
3	Learning Rates Adjust	0
4	N of antecedents x_1	4
...	...	...
4+n-1	N of antecedentsx_n	5
4+n	Learning Rate x_1	0.75
...	...	...
4+2n-1	Learning Ratex_n	1.2

(a)

Position	Meaning	Example
0	Max. n of Iterations	55
1	*SSE Target Error*	0.003
2	*GSSE* Error	0
3	Learning Rates Adjust	1
4	N of antecedents x_1	4
...	...	...
4+n-1	N of antecedentsx_n	5
4+n	Learning Rate$m$ x_1	0.01
...	...	...
4+2n-1	Learning Rate$m$ x_n	0.02
4+2n	Learning Rateσ x_1	0.04
...	...	
4+3n-1	Learning Rateσ x_n	0.06

(b)

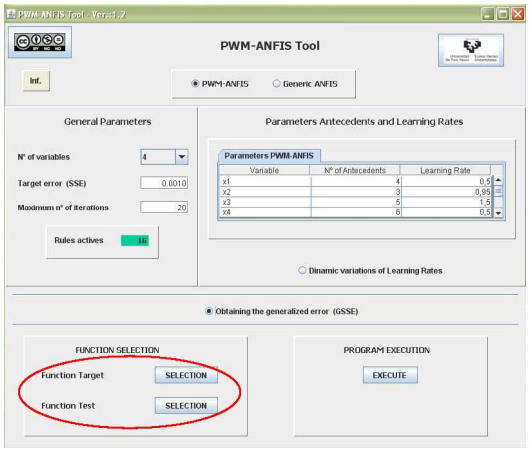


Figure 21: Interface of the pwm-anfis tool: function selection.

$x_1 \in [-1, 1]$ and $x_2 \in [1, 1]$. A collection of training points is obtained by sampling x_1 and x_2 with 41 points respectively; hence, there are a total of 1681 (41^2) training points. The parameters for the system are the following:

1. Number of antecedents for each variable $m_1 = m_2 = 11$
2. Learning rate for each variable $\eta_1 = \eta_2 = 3.5$
3. Maximum number of iterations $i_{max} = 50$
4. Target error $SSE_t = 10^{-4}$

The tool starts by importing the input files from which the size and dimensionality of the NFS are calculated. Then, it builds up the NFS and begins the training process. After the 50th iteration, the process stops and displays the *SSE* curve value (see Fig. 23). Can be seen that the system reaches the error $SSE = 1.96\ 10^{-4}$. The tool also provides a graphical representation of the obtained surface (i.e., the output for the training points) (see Fig. 22(b)). Note the close similarity between the target function and the learned function. Also, can be see the initial and learned membership functions in Fig. 24. The *GSSE* has not been evaluated because the full set of test points has been used in the training process. Table 4 shows the comparison between the PWM-ANFIS and the Generic ANFIS (unrestricted Gaussian membership functions) with 11 Gaussian membership functions per variable. As can be seen in this case, due to the nature of the Gaussian membership functions, the Generic ANFIS outperforms the

Table 3: Array of the target or test functions.

$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	...	$\boldsymbol{x_8}$	$\boldsymbol{y'}$
x_{11}	x_{21}	...	x_{81}	y_1'
x_{11}	x_{21}	...	x_{82}	y_2'
...	...	...	...	...
x_{11}	x_{21}	...	x_{8q}	y_i'
...	...	...	...	...
x_{11}	x_{22}	...	x_{81}	y_j'
...	...	...	...	...
x_{1n}	x_{2m}	...	x_{8q}	y_s'

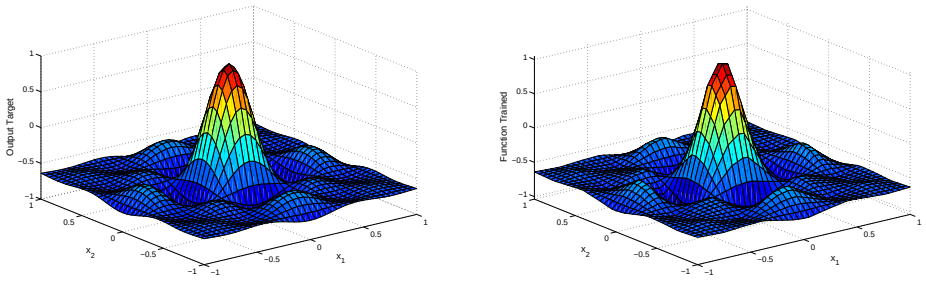


Figure 22: (a) Two-variable target function, (b) Two-variable learned function.

approximation obtained with the PWM-ANFIS in the training points. The results are similar choosing 8 Gaussian membership functions per variable ($GSSE = 2.6 \cdot 10^{-4}$).

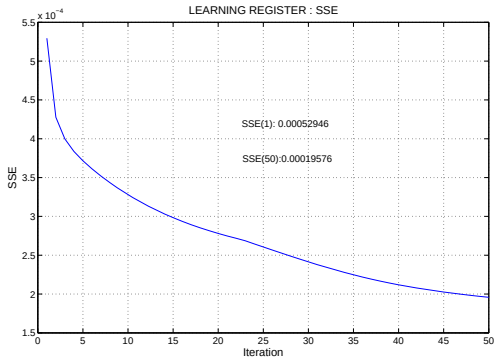


Figure 23: Evolution of the SSE for the two-variable example.

5.1.2. Three-variables.

In this example the approximation of a polynomial 3-input function is presented. The target function is $y = (1 + x_1^{0.5} + x_2^{-1} + x_3^{-1.5})^2$, with x_1, x_2 and x_3 in [1,6]. In this experiment a total of 1331 (11^3) training points and another 1000 intermediate test points have been collected. The input parameters are now the following:

1. Number of antecedents for each variable $m_1 = m_2 = m_3 = 5$
2. Learning rate for each variable $\eta_1 = \eta_2 = \eta_3 = 0.99$
3. Maximum number of iterations $i_{max} = 30$
4. Target error $SSE_t = 0.0001$

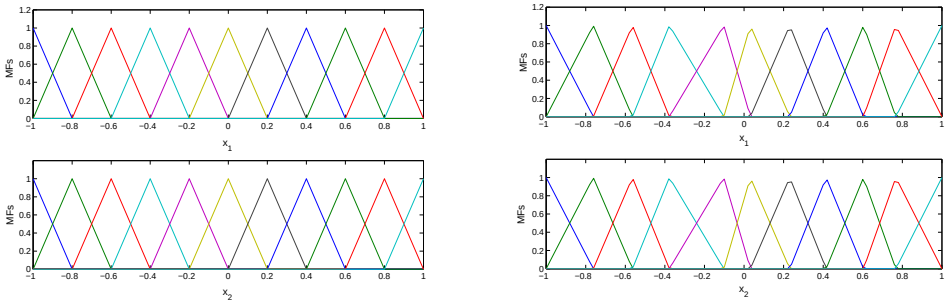


Figure 24: (a) Two-variable initial membership functions, (b) Two-variable learned membership functions.

Fig. 25 shows the obtained surfaces, (a) target surface and (b) learned surface, where x_3 has been taken as a parameter. Fig. 26(a) shows the error values once the 50 iterations have been completed. Can be see again that the system reaches very low error values; i.e., $SSE = 0.0398$ and $GSSE = 0.031$. Table 4 shows the comparison between the PWM-ANFIS and the Generic ANFIS with 5 membership functions per variable.

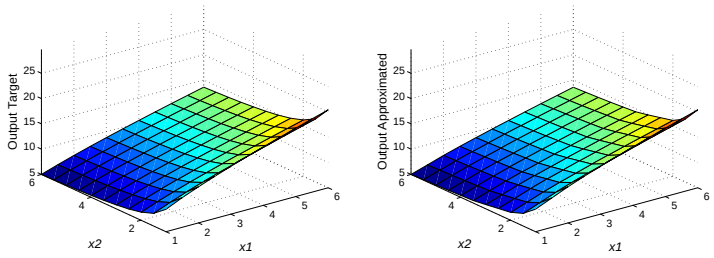


Figure 25: Three-variable function with x_3 as a parameter (a) target function, (b) learned function.

In this case, both the *SSE* and the *GSSE*, are lower if a Generic ANFIS is used. However, despite of the restriction imposed on the present system, the approximation capability of the PWM-ANFIS is not significantly reduced in comparison with the Generic ANFIS; note that both errors are of the same order of magnitude.

Table 4: Comparison of the approximation capability between PWM-ANFIS and Generic ANFIS for examples with two, three and four variables

	PWM-ANFIS		**Generic** ANFIS	
N° Variables	*SSE*	*GSSE*	*SSE*	*GSSE*
2	$1.4\ 10^{-4}$	-	$2.65\ 10^{-6}$	-
3	0.0398	0.031	0.016	0.015
4	0.044	0.072	0.096	0.061

5.1.3. Four-variables.

To analyze the response of the system for a large number of inputs, a 4-input function has also been tested. The function is given by $y = 4(x_1 - 0.5)(x_4 - 0.5)\sin(2\pi(x_2^2 + x_3^2)^{1/2})$, where x_1, x_2, x_3 and x_4 in [-1,1]. The number of training points and test points are respectively 4096 (8^4) and 2041 (7^4). The parameters for the system are the following:

1. Number of antecedents for each variable $m_1 = m_2 = m_3 = m_4 = 5$
2. Learning rate for each variable $\eta_1 = \eta_2 = \eta_3 = \eta_4 = 0.5$
3. Maximum number of iterations $i_{max} = 50$
4. Target error $SSE_t = 0.0001$

Results obtained are depicted in Fig. 26(b). Once again can be see that the values of the errors are very small; i.e., *SSE=0.044* and *GSSE=0.0722*. Fig. 27 shows the initial and learned membership functions. Table 4 shows the comparison between the PWM-ANFIS and the Generic ANFIS with 5 antecedents per variable.

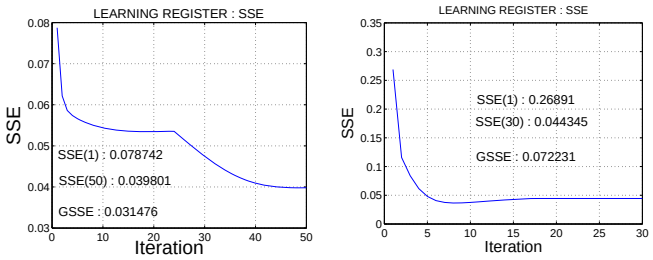


Figure 26: SSE and GSSE errors (a) 3-variable example, (b) 4-variable example.

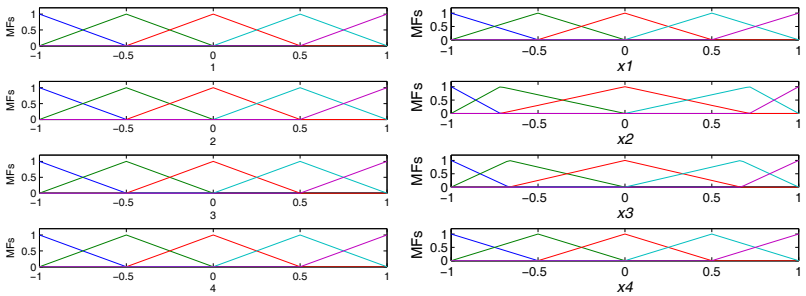


Figure 27: (a) Four-variable initial membership functions, (b) Four-variable learned membership functions.

As in the first example, the PWM-ANFIS reaches a lower *SSE*. However, concerning the non training data points, the *GSSE* is lower if a Generic ANFIS is used. This last result is in agreement with the well known properties of Gaussian membership functions in Neuro-Fuzzy modelling [31]. As in the previous examples both errors (*SSE* and *GSSE*) maintain the same order of magnitude.

5.2. Approximation for on-line training

This example has been performed with the tool developed on Matlab without the Java interface of the PWM-ANFIS Tool as has been presented in the Section 3.

5.2.1. Identification of a plant.

An on-line training experiment has also been carried out [14]. On this occasion, the PWM-ANFIS system has to identify an unknown function of a given plant, from which are taken input-output samples. Let us assume that the plant is governed by the following difference equation: $y'(k+2) = 0.3y'(k+1) + 0.6y'(k) + f(u(k+1))$ where $y^{'}(k)$ and $u(k)$ are respectively the output and input at time step k, and $f(u)$ is the unknown function to be identified. Shall be assumed that $f(u)$ has the following form: $f(u) = 0.6\sin(\pi u) + 0.3\sin(3\pi u) + 0.1\sin(5\pi u)$ therefore will be taken from this equation the collection of input-output data samples for training. Finally the input $u(k)$ is a sinusoid function: $u(k) = \sin(2\pi k/250)$. The number of samples for $u(k)$ is 250.
In order to identify the plant, the new function under consideration is: $y(k+2) = 0.3y(k+1)+0.6y(k)+F(u(k+1))$, where $F(u)$ is the model to be identified by the PWM-ANFIS Tool. This function has a single input, hence the number of rules and consequents is the same as the number of membership functions, and the number of membership functions is chosen by trial and error. For example, good approximation can be obtained with 30 membership functions. The parameters for the system are the following:

1. Number of antecedents for each variable $m_1 = 30$
2. Learning rate for each variable $\eta_1 = 0.01$
3. Forgetting factor $\lambda = 1$

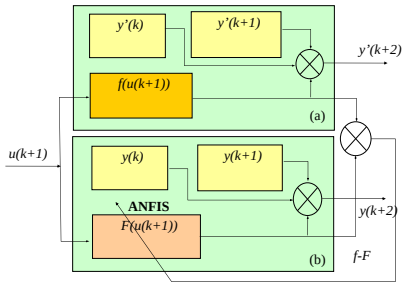


Figure 28: (a) Target Plant and (b) Learned Plant.

4. Maximum number of iterations $i_{max} = 100$

5. Target error $SSE_t = 10^{-12}$

Taking into account that SSE is calculated on each training couple, it is better to have an accumulated error such as the sum of SSE obtained in all iterations, that is to say $Sum\_SSE(sample) = \sum_{sample=1}^{250} \frac{SSE(sample)}{sample}$.

Fig. 29(a) shows the comparisons between the target function and the learned function. Can be see how the learned function follows the target one almost immediately. This result is made clear in Fig. 29(b) where the accumulated SSE between these functions has been depicted.

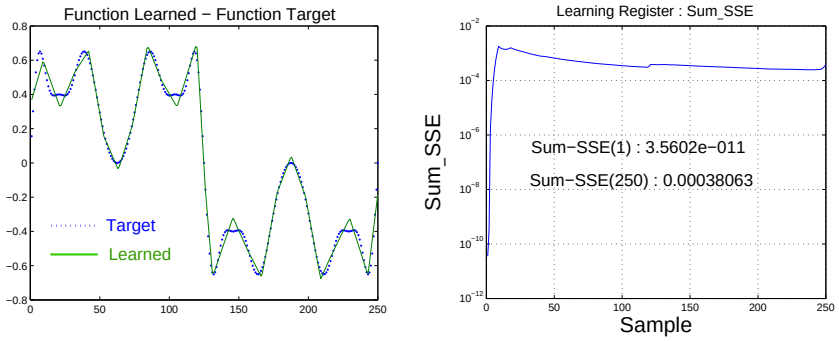


Figure 29: (a) Target and Learned Plant, (b) Learning Register Sum_SSE.

The results obtained for the above examples lead us to conclude that the learning performance of the PWM-ANFIS is satisfactory in comparison with the Generic ANFIS; the results show that Generic ANFIS is hardly a better function approximator but the order of the errors is the same in both models. However, taking into account Table 1 showing the computational cost of both models, the advantages of PWM-ANFIS are obvious.

6. Conclusions

A tool for modelize complex systems has been presented. The tool are based on a class of neuro-fuzzy systems and has been developed on Matlab environment. The class of the neuro-fuzzy systems is an adaptive neuro-fuzzy inference systems (ANFIS) with piecewise multilinear (PWM) behavior. The tool has been named PWM-ANFIS Tool and presents a computational-efficient behaviour tested with several non-linear functions both on-line and off-line learning. The tool presents a very good approximation capabilities and learning performance.

Acknowledgements

The authors would like to thank the Basque Country Government for support of this work under Grants SA-2006/00015 and IT-353-07

References

[1] K. Ray, J. Ghoshal, Neuro fuzzy approach to pattern recognition, Neural Networks 10 (1) (1997) 161–182.

[2] S. K. Pal, S. Mitra, Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing, Wiley-Interscience, 1999.

[3] P. Rusu, E. M. Petriu, T. E. Whalen, A. C. and. H. J. W. Spoelder, Behavior-based neuro-fuzzy controller for mobile robot navigation, IEEE Transaction on Instrumentation and Measurement 52 (4) (2003) 1335–1340.

[4] H. Wongsuwarn, D. Laowattana, Neuro-fuzzy algorithm for a biped robotic system, World Academy of Science, Engineering and Technology 15 (2006) 138–144.

[5] R. Babuska, H. Verbruggen, Neuro-fuzzy methods for nonlinear systems identification, Annual Reviews in Control 27 (2003) 73–85.

[6] P. Panchariya, A. Palit, D. Popovic, A. Sharmal, Nonlinear system identification using takagi-sugeno type neuro-fuzzy model, in: Second IEEE International Conference on Intelligent Systems, IEEE, 2004, pp. 76–81.

[7] C. Li, K.-B. Tsai, Adaptive interference signal processing with intelligent neuro-fuzzy approach, in: ICCOMP'06 Proceedings of the 10th WSEAS international conference on Computers, 2006, pp. 393–398.

[8] S. Chabaa, A. Zeroual, J. Antari, Application of adaptive neuro-fuzzy inference systems for analyzing non-gaussian signal, in: International Conference on Multimedia Computing and Systems. ICMCS '09, 2009, pp. 377 – 380.

[9] G. Acampora, V. Loia, A proposal of ubiquitous fuzzy computing for ambient intelligence, Information Science 178 (2008) 631 – 646.

[10] Y. Kishino, T. Terada, M. Tsukamoto, T. Yoshihisa, K. Hayakawa, A. Kashitani, S. Nishio, A rule-based discovery mechanism of network topology among ubiquitous chips, in: International Conference on Pervasive Services (ICPS 05), 2005, pp. 198 – 207.

[11] Y. Kawahara, M. Minami, H.Morikawa, T. Aoyama, Design and implementation of a sensor network node for ubiquitous computing environment, in: IEEE 58th Vehicular Technology Conference. VTC 2003, Vol. 5, 2003, pp. 3005 – 3009.

[12] J. Buckley, Sugeno type controllers are universal controllers, Fuzzy Sets Systems 53 (1) (1993) 299–303.

[13] J. Castro, Fuzzy logic controllers are universal approximators, IEEE Transactions on Systems, Man, and Cyberbetics 25 (4) (1995) 629–635.

[14] J. Jang, C. Sun, E. Mizutani, Neuro-Fuzzy and Soft Computing, Prentice Hall, 1997.

[15] D. Nauck, R. Kruse, Neuro-fuzzy systems for function approximation, Fuzzy Sets and Systems 101 (1999) 261–271.

[16] L.-X. Wang, C. Wei, Approximation accuracy of some neuro-fuzzy appoaches, IEEE Transactions on Fuzzy Systems 8 (4) (2000) 470–478.

[17] W. Wu, L. Li, J. Yang, Y. Liub, A modified gradient-based neuro-fuzzy learning algorithm and its convergence, Information Sciences 180 (2010) 1630–1642.

[18] G. Bosque, I. del Campo, J. Echanobe, J. Tarela, Modelling and synthesis of computational efficient adaptive neuro-fuzzy systems based on matlab, in: Springer (Ed.), Artificial Neural Networks - ICANN 2008 - LNCS 5164, Vol. 2, 2008, pp. 131–140.

[19] J.-S. Jang, Anfis: Adaptive-network-based fuzzy inference system, IEEE Transactions on Systems, Man, and Cybernetics 23 (1993) 665–685.

[20] J.-S. R. Jang, C.-T. Sun, Neuro-fuzzy modeling and control, Proceedings of the IEEE 83 (3) (1995) 378–406.

[21] S.-K. Oh, W. Pedrycz, B.-J. Park, Relation-based neuro-fuzzy networks with evolutionary data granulation, Mathematical and Computer Modelling 40 (2004) 891–921.

[22] B. Kosko, Fuzzy systems and universal approximators, IEEE Transactions Computing 43 (11) (1994) 1329–1333.

[23] X.-J. Zeng, M. Singh, Approximation theory of fuzzy systems-siso case, IEEE Transactions on Fuzzy Systems 2 (2) (1994) 162–176.

[24] X.-J. Zeng, M. Singh., Approximation theory of fuzzy systems-mimo case, IEEE Transactions on Fuzzy Systems 3 (2) (1995) 219–235.

[25] X.-J. Zeng, M. Singh, Approximation accuracy analys of fuzzy systems as function approximators, IEEE Transactions on Fuzzy Systems 4 (1) (1996) 44–63.

[26] R. Rovatti, Fuzzy piecewise multilinear and piecewise linear systems as universal approximator in sobolev norms, IEEE Transactions on Fuzzy Systems 6 (2) (1998) 235–249.

[27] S. Cao, N. Rees, G. Feng, Mamdani-type fuzzy controllers are universal fuzzy approximators, IEEE Transactions Computer 123 (3) (2001) 359–367.

[28] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modelling and control, IEEE Transactions on Systems, Man and Cybernetics 15 (1985) 116–132.

[29] M. Sugeno, G. Kang, Structure identification of fuzzy model, Fuzzy Sets Systems 28 (1) (1988) 15–33.

[30] W. Pedrycz, Why triangular membership functions?, Fuzzy Sets and Systems 64 (1994) 21–30.

[31] K. Basterretxea, I. del Campo, J. M. Tarela, G. Bosque, An experimental study on non-linear function computation for neural/fuzzy hardware design, IEEE Transactions on Neural Networks 18 (2007) 266–283.

[32] G. Bosque, I. del Campo, J. Echanobe, Efficient hardware/software implementation of a neuro-fuzzy system on a sopc, in: Recent Advanced in Soft Computing (RASC), 2006.

[33] J. Echanobe, I. del Campo, G. Bosque, An adaptive neuro-fuzzy system for efficient implementations, Information Science 178 (9) (May 2008) 2150–2162.

[34] I. del Campo, J. Echanobe, G. Bosque, J. Tarela, Efficient hardware/software implementation of an adaptive neuro-fuzzy system, IEEE Transactions on Fuzzy Systems 16 (3) (Junne 2008) 761–778.

[35] G. Bosque, I. del Campo, J. Echanobe, J. Tarela, Implementacin de un sistema neuro-fuzzy sobre un sopc, in: Congreso Espaol sobre Tecnologas y Lgica Fuzzy (Estylf 2004), European Society for Fuzzy Logic and Technology (EUSFLAT), 2004, pp. 587–592.

[36] S. Lee, C. Ouyang, Neuro-fuzzy system modeling with self-constructing rule generation and hybrid svd-based learning, IEEE Transactions on Fuzzy Systems 11 (3) (2003) 341–353.

APPENDIX

A. CHAIN RULE AND FACTORS

Taking into account the Equation (16), the derivative of this expression is:

$$\frac{\partial E(a)}{\partial a_{ij_i}} = \frac{\partial}{\partial a_{ij_i}} \left(\frac{1}{2K} \sum_{k=1}^{K} (y_k - y_k^{'})^2 \right) = \frac{1}{K} \left(\sum_{k=1}^{K} (y_k - y_k^{'}) \frac{\partial y_k}{\partial a_{ij_i}} \right). \tag{23}$$

Now, it is important to highlight the term $\frac{\partial y_k}{\partial a_{ij_i}}$, taking into account the Equation (15),

$$\frac{\partial y_k}{\partial a_{ij_i}} = r_1 \frac{\partial \omega_1^k}{\partial a_{ij_i}} + r_2 \frac{\partial \omega_2^k}{\partial a_{ij_i}} + \ldots + r_q \frac{\partial \omega_q^k}{\partial a_{ij_i}} + \ldots + r_p \frac{\partial \omega_p^k}{\partial a_{ij_i}}, \tag{24}$$

where the super-index k refers to the *kth*-sample. The term $\frac{\partial \omega_q^k}{\partial a_{ij_i}}$, referring to the Equation (15) is expressed as:

$$\frac{\partial \omega_q^k}{\partial a_{ij_i}} = \frac{\partial}{\partial a_{ij_i}} \prod_{l=1}^{n} A_{lj_l}(x_l^k), \tag{25}$$

where $1 \leq j_l \leq m_l$. The Algorithm of Rule Activations (see appendix B) produces all the combinations of the antecedents, and the value of the index j_l is thus calculated (Fig. 5 shows the combination of all the antecedents). Taking into account the restrictions of the triangles (see Fig. 6) will have (a) $a_{ij_i} = b_{i(j_i-1)}$ and $a_{ij_i} = c_{i(j_i-2)}$, then:

$$\frac{\partial A_{lj_l}(x_i)}{\partial a_{ij_i}} = \text{where } 1 \leq l \leq n \begin{cases} 0, & \text{if } l \neq i \\ 0, & \text{if } j_l \neq j_i \text{ or } j_l \neq j_i - 1 \text{ or } \\ & j_l \neq j_i - 2 \\ \frac{\partial A_{ij_i}(x_i)}{\partial a_{ij_i}}, & \text{if } l = i \text{ and } j_l = j_i \\ \frac{\partial A_{i(j_i-1)}(x_i)}{\partial a_{ij_i}}, & \text{if } l = i \text{ and } j_l = j_i - 1 \text{ (a)} \\ \frac{\partial A_{i(j_i-2)}(x_i)}{\partial a_{ij_i}}, & \text{if } l = i \text{ and } j_l = j_i - 2 \text{ (b)}, \end{cases} \tag{26}$$

where

$$\begin{aligned} \frac{\partial A_{ij_i}(x_i)}{\partial a_{ij_i}} &= \frac{1}{a_{i(j_i+1)} - a_{ij_i}} \left(\frac{x_i - a_{i(j_i+1)}}{a_{i(j_i+1)} - a_{ij_i}} \right) \\ \frac{\partial A_{i(j_i-1)}(x_i)}{\partial a_{ij_i}} &= \frac{1}{a_{i(j_i+1)} - a_{ij_i}} \left(\frac{a_{i(j_i+1)} - x_i}{a_{i(j_i+1)} - a_{ij_i}} \right) \\ \frac{\partial A_{i(j_i-2)}(x_i)}{\partial a_{ij_i}} &= \frac{1}{a_{ij_i} - a_{i(j_i-1)}} \left(\frac{x_i - a_{i(j_i-1)}}{a_{ij_i} - a_{i(j_i-1)}} \right). \end{aligned} \tag{27}$$

Thus will have three terms for each a_{ij_i}.

$$\frac{\partial A_{ij_i}(x_i)}{\partial a_{ij_i}}, \frac{\partial A_{i(j_i-1)}(x_i)}{\partial a_{ij_i}}, \frac{\partial A_{i(j_i-2)}(x_i)}{\partial a_{ij_i}} \tag{28}$$

Applying these terms to the Equation (25)

$$\frac{\partial \omega_q}{\partial a_{ij_i}} = \begin{cases} \left(\prod_{s_i \neq j_i} A_{is_i}(x_i) \right) \frac{\partial A_{ij_i}(x_i)}{\partial a_{ij_i}} \\ \left(\prod_{s_i \neq j_i - 1} A_{is_i}(x_i) \right) \frac{\partial A_{i(j_i-1)}(x_i)}{\partial a_{ij_i}} \\ \left(\prod_{s_i \neq j_i - 2} A_{is_i}(x_i) \right) \frac{\partial A_{i(j_i-2)}(x_i)}{\partial a_{ij_i}}, \end{cases} \tag{29}$$

where q that affects to the three terms, is obtained in the algorithm expressed in m-code, as will be seen below in this appendix. The variable q of this m-code has the same interpretation that in the equations. (24), (25), (29). The algorithm has three sections,

1. Factors for x_1: where q concerns to the parameters a_{1j_1}
2. Factors for x_i, being $2 < i < n$: where q concerns to the parameters a_{2j_2}, $j \in [1, m_2]$, $\cdots a_{(n-1)j_{(n-1)}}$, $j \in [1, m_{(n-1)}]$
3. Factors for x_n: where q concern to the parameters a_{nj_n}, $j \in [1, m_n]$

Each product of the antecedents, in the Equation (29), has *n-1* terms. Applying this process in the Equation (24), and grouping the different products shall be obtained the final expression:

$$\frac{\partial y_k}{\partial a_{ij_i}} = F(A_{ij_i})\frac{\partial A_{ij_i}}{\partial a_{ij_i}} + F(A_{i(j_i-1)})\frac{\partial A_{i(j_i-1)}}{\partial a_{ij_i}} + F(A_{i(j_i-2)})\frac{\partial A_{i(j_i-2)}}{\partial a_{ij_i}}. \tag{30}$$

The grouping of the different products has been made with reference to the next algorithm. Firstly taking into account the structure of the combination of all the antecedents will have the next definitions:

1. Number of Rules (NR): $NR = \prod_{i=1}^{n} m_i$
2. Number of Consecutive Elements (NCE) for a variable i and an antecedent j_i:
   (a) If $i \neq n$, $NCE(i) = \prod_{k=i+1}^{n} m_k$
   (b) if $i = n$, $NCE(n) = 1$
3. Number of Total Elements (NTE) for a variable i and an antecedent j_i:
   (a) If $i = 1$, $NTE(i) = NCE(i)$
   (b) If $i \neq 1$, $NTE\,(i) = \prod_{i=1}^{i-1} m_i \cdot \prod_{i+1}^{n} m_i$, could be also expressed $NTE\,(i) = \frac{NR}{m_i}$
4. Number of Blocks of Elements (NB) for an antecedent:
   (a) If $i = 1$, $NB(i) = 1$
   (b) If $i \neq 1$, $NB\,(i) = \prod_{k=1}^{i-1} m_k$, could be also expressed $NB\,(i) = \frac{NTE(i)}{NCE(i)}$
5. Gap between Blocks of Consecutive Elements ($GBCE$):
   (a) If $i = 1$, $GBCE(1) = 1$
   (b) If $i \neq 1$, $GBCE(i) = \prod_{k=1}^{n} m_k$, could be also express $GBCE(i) = \frac{NR}{NB(i)}$

From now on the antecedent A_{ij_i} will be expressed in matrix form $A(i, antecedent)$, where i is the variable and $antecedent$ is the number of antecedents for this variable, m_i will be $m(i)$. The variable $structure\_antecedents$ contains the combination of the antecedents for a rule and is obtained in the Algorithm of Rule Activations (Appendix A.2). The algorithm will be expressed in m-code of Matlab.

1. Factors for x_1,

```
k = 1; nce = NCE(1);
for j = 1 : m(1) % For 1 to the number of antecedents of x1
    sum = 0;
    for q = k : nce
        semi_antecedent = 1;
        for i = 2 : n
            antecedent = structure_antecedents(q, i);
            semi_antecedent = semi_antecedent * A(i, antecedent);
        end
        sum = sum + r(q) * semi_antecedent;
    end
    k = nce + 1;
    nce = nce + NCE(1);
    FactorA(1, j) = sum;
end
```

2. Factors for x_i, where $2 < i < n$,

```
for i = 2 : n − 1 % Variables between 1 and n
    k = 1; nce = NCE(i);
    for j = 1 : m(i) % For 1 to the number of antecedents of x_i
        sum = 0;
        for nb = 1 : NB(i)
            semi_antecedent = 1;
            for q = k : nce
                for var = 1 : i − 1
                    semi_antecedent = semi_antecedent ∗ A(i, antecedent);
                end
                for var = i + 1 : n
                antecedent = structure_antecedents(q, i);
                    semi_antecedent = semi_antecedent ∗ A(i, antecedent);
                end
                sum = sum + r(q) ∗ semi_antecedent;
            end
            k = k + GBCE(i);
            nce = k + NCE(i) − 1;
        end
        FactorA(i, j) = sum;
        k = NCE(i) ∗ j + 1;
        nce = k + NCE(i);
    end
end
```

3. Factors for x_n,

```
k = 1; p = 1; % Internal Variables
for j = 1 : m(n) % For 1 to the number of antecedents of x_n
    sum = 0;
    for q = j : m(n) : NR − (m(n) − p)
        semi_antecedent = 1;
        for i = 1 : n − 1
            antecedent = structure_antecedents(q, n);
            semi_antecedent = semi_antecedent ∗ A(i, antecedent);
        end
        sum = sum + r(q) ∗ semi_antecedent;
    end
    FactorA(n, j) = sum;
    p = p + 1;
end
```

B. ALGORITHM OF RULE ACTIVATIONS

This algorithm obtains an array, named *structure_antecedents*, which contains all the combinations of the antecedents of the variables that conform all the rule activations (see Eq. (11)). With the index *index_antecedents* the algorithm obtains the *structure_antecedents*. The algorithm is written in m-code of Matlab.

```
for i = 1 to n
    index_antecedents(i) = 1; % Index that cover all antecedents of each variable x_i
    round(i) = 0;  % Variable that shows if all the antecedents of a variable x_i have been covered
end
```

```
p = 1; % Internal Variable
structure_antecedents = [];

while round(1) == 0
    for j = 1 : m(n) % m(n) is the last antecedent of the variable x_i
        structure_antecedents(p,:) = index_antecedents;
        structure_antecedents(p,n) = structure_antecedents(p,n) + j - 1;
        p = p + 1;
    end
    for i = n - 1 : -1 : 1
        if i == n - 1
          if index_antecedents(i) < m(i)
            index_antecedents(i) = index_antecedents(i) + 1;
          else
            index_antecedents(i) = 1;
            round(i) = 1;
          end
        else
            if round(i + 1) == 1
              round(i + 1) = 0;
              if index_antecedents(i) < m(i)
                index_antecedents(i) = index_antecedents(i) + 1;
              else
                index_antecedents(i) = 1;
                round(i) = 1;
              end
            end
        end
    end
end
```