TRABAJO ESPECIAL DE GRADO

DESARROLLO E IMPLEMENTACIÓN DE UN MODELO DE INFERENCIA HIDROLÓGICO MEDIANTE REDES ANFIS Y PAQUETES DE SOFTWARE LIBRE

Presentado ante la ilustre
Universidad Central de Venezuela
por el Br. Andrés G. Avendaño C.
para optar al Título de
Ingeniero Electricista

Caracas, 2009

TRABAJO ESPECIAL DE GRADO

DESARROLLO E IMPLEMENTACIÓN DE UN MODELO DE INFERENCIA HIDROLÓGICO MEDIANTE REDES ANFIS Y PAQUETES DE SOFTWARE LIBRE

PROFESOR GUÍA: Profa. Tamara Pérez
TUTOR INDUSTRIAL: Prof. Iván Saavedra

Presentado ante la ilustre
Universidad Central de Venezuela
por el Br. Andrés G. Avendaño C.
para optar al Titulo de
Ingeniero Electricista

Caracas, 2009

CONSTANCIA DE APROBACIÓN

Caracas, 2009

Los abajo firmantes, miembros del Jurado designado por el Consejo de Escuela de Ingeniería Eléctrica, para evaluar el Trabajo Especial de Grado presentado por el Bachiller Andrés G. Avendaño C., titulado:

"DESARROLLO E IMPLEMENTACIÓN DE UN MODELO DE INFERENCIA HIDROLÓGICO MEDIANTE REDES ANFIS Y PAQUETES DE SOFTWARE LIBRE"

Consideran que el mismo cumple con los requisitos exigidos por el plan de estudios conducente al Título de Ingeniero Electricista en la mención de Electrónica, Computación y Control sin que ello signifique que se hacen solidarios con las ideas expuestas por el autor, lo declaran APROBADO.

Prof. Panayotis Tremante
Jurado

Prof. William La Cruz
Jurado

Prof. Tamara Pérez
Profesor guía.

DEDICATORIA

A mis padres y abuelas. Sin cuya ayuda y compresión este trabajo no hubiera sido posible

A mis compañeros del Núcleo Cagua y la Escuela de Ingeniería Eléctrica con los cuales comparto muchos recuerdos gratos.

RECONOCIMIENTOS Y AGRADECIMIENTOS

Quiero dar un pequeño reconocimiento a todas aquellas personas cuya influencia a contribuido de una u otra manera con mi desarrollo académico y personal, por tal motivo expreso mi total gratitud a:

- Mis profesores, entre ellos: Jorge Retamozo, Isabel Díaz, Rebeca Pradere, William La Cruz, Rafael Arrabarruena, Rafael Rivero, Pedro Pinto, Walter Senessi, Jorge Bernadas y especialmente a la profesora Tamara Pérez, que me introdujo a este fascinante trabajo de grado.

- Mis amigos y compañeros de estudio: Martha Rodríguez, Deisy Marcano, Carina De Sousa, Andrés Segovia, Esther Ferro, Adriana Hurtado, Jolly Pérez, Ramón Méndez, Cesar Gutiérrez, Ricardo Parra, Roberto Olivares, Carlos Espluga, Héctor Mota, Balmore García, Aníbal Montaño, Zolandia Silva, Victor Devia, Dany Abreu, Jackson Ramírez, Servando Álvarez, Carlos Di Yorio, Victor González, Andrés Iriarte entre muchos otros.

Al profesor Iván Saavedra por facilitar el tema, los profesores Henry Flores y Judith Fernandez por facilitar, las series de estudio, al Prof. Luis Fernandez por el tiempo dedicado al montaje del servidor web.

A Leonardo Yepez por su aporte desinteresado en el desarrollo del entonorno web, las comunidades de los foros “tek-tips” (http://www.tek-tips.com) y “Scilab” (http://groups.google.co.ve/group/comp.soft-sys.math.scilab/topics).

Avendaño C. , Andrés G.

DESARROLLO E IMPLEMENTACIÓN DE UN MODELO DE INFERENCIA HIDROLÓGICO MEDIANTE REDES ANFIS Y PAQUETES DE SOFTWARE LIBRE

Profesor Guía: Tamara Pérez A. Tutor industrial: Prof. Iván Saavedra. Tesis. Caracas U.C.V. Facultad de Ingeniería. Escuela de Ingeniería Eléctrica. Ingeniero Electricista 2009. Opción: Electrónica. Institución: Instituto de Mecánica de Fluidos. 89h + anexos.

Palabras Claves: Modelo ANFIS; Sistema de inferencia difuso; Modelos hidrológicos; Prevención de desastres; Alerta temprana; Software libre.

Resumen. Se estudia la arquitectura ANFIS, definiendo y estableciendo los pasos para su utilización como modelo hidrológico, desarrollando el modelo de inferencia ANFIS con paquetes de código abierto entrenándolo y validándolo con series conocidas, para finalmente visualizar sus resultados y datos de entrenamiento en un entorno gráfico amigable. El proyecto PROCEDA cuenta con estaciones de medición automática y un amplio registro de series históricas, sin embargo, carece de herramientas de pronóstico y divulgación, considerando que los modelos de inferencia pueden representar características altamente no lineales presentes en las series hidrológicas, se desarrolla e implementa una herramienta en Scilab que utiliza el modelo ANFIS como herramienta de pronóstico y un entorno web para la visualización de los datos de entrenamiento y los resultados del modelo, adicionalmente se hacen propuestas para la automatización de la infraestructura actual.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

SIGLAS Y ABREVIATURAS

ANFIS	Adaptative-Network-Based Fuzzy Inference Systems.
ANSI	American National Standars Institute
AR	Autoregressive Model, Modelo Autorregresivo.
FIS	Fuzzy Inference Systems, Sistema de inferencia difuso.
FSF	Free Software Foundation, Fundación de Software Libre.
GCC	GNU Compiler Collection, Colección de Compiladores GNU.
GNU	GNU's Not Unix, GNU no es UNIX.
GPL	General Public License, Licencia al Público General
HTML	Hyper Text Markup Language
IMF	Instituto de Mecánica de Fluidos.
LSE	Least Squares Estimate, Estimación por Mínimos Cuadrados.
OSI	Open Source Initiative, Iniciativa de Fuente Abierta.
PROCEDA	Proyecto de Cuenca Experimental del Macizo Ávila.
RBF	Radial Basis Function, Funciones de Base Radial
RMSE	Root Mean Squared Error, Error Cuadrático Medio.

INTRODUCCIÓN

El presente trabajo nace en la Universidad Central de Venezuela, específicamente en el Instituto de Mecánica de Fluidos como parte del proyecto PROCEDA (Proyecto de Cuenca Experimental del Macizo Ávila) con la intención de utilizar las series de datos existentes de la Cuenca Experimental de Galipán en la falda Norte del Ávila (Caracas – Venezuela), para el desarrollo de un modelo de pronóstico que permita incorporar un sistema de alerta temprana a la infraestructura ya existente.

Los modelos de pronóstico hidrológicos frecuentemente dependen tanto de las entradas presentes como las anteriores, considerando ésto es posible estimar la salida del sistema siempre y cuando se tengan suficientes datos históricos de su comportamiento. El ajuste de los parámetros de un modelo de inferencia es realizado por un experto, sin embargo, estos modelos son tan cambiantes que una supervisión constante podría resultar muy complicada.

El modelo ANFIS (Adaptative-Network-Based Fuzzy Inferece System), busca minimizar este problema imitando la función del experto con fundamentos de la lógica difusa y redes neuronales. Por supuesto no existe un método estándar para transformar la experiencia o el conocimiento humano en un modelo preciso, pero es posible ajustar automáticamente los parámetros de un modelo de inferencia para minimizar el error a su salida.

No se encontraron precedentes del uso de herramientas de pronóstico automatizado para el proyecto PROCEDA, por tal motivo el presente trabajo de grado propone el desarrollo e implementación de una herramienta de pronóstico basado en

el modelo de inferencia ANFIS, la propuesta para tal fin es presentada en 4 capítulos.

En el capítulo 1 se cubren los fundamentos teóricos para entender el funcionamiento del modelo de inferencia ANFIS, así como una descripción de los paquetes de software libre usados para su implementación, en el capítulo 2 se exponen las consideraciones de diseño de las herramientas a desarrollar y como seria su posible implementación dentro de la estructura actual del proyecto PROCEDA, el capítulo 3 cubre las características del software desarrollado para satisfacer las condiciones de diseño, finalmente en el capítulo 4 se ilustran posibles aplicaciones de la herramienta en el pronóstico y la simulación de series altamente no lineales.

CAPÍTULO I

PLANTEAMIENTO DEL PROBLEMA

Se desea desarrollar e implementar un modelo de inferencia ANFIS para la predicción de variables hidrológicas, cuyos resultados se muestran en un entorno gráfico amigable, adicionalmente se desea un entorno que permita ver los resultados del modelo por Internet y permita establecer niveles de alerta para la prevención de desastres originados por factores hidrológicos. El desarrollo del modelo y el entorno gráfico se hará con paquetes de código abierto de modo que puedan modificarse y adaptarse a muchas otras aplicaciones.

OBJETIVO GENERAL

Desarrollar un modelo de inferencia basado en datos hidrológicos con herramientas de computación emergente y el uso de paquetes de código abierto.

OBJETIVOS ESPECÍFICOS

- Estudiar las variables asociadas con el desarrollo de modelos hidrológicos.
- Investigar las distintas metodologías usadas para el desarrollo de series estocásticas utilizadas en el desarrollo de modelos hidrológicos.
- Investigar las diferentes topologías o arquitecturas de modelos ANFIS.
- Proponer la automatización de la adquisición y transmisión de los datos provenientes de las estaciones hidrológicas remotas.
- Determinar el software de código abierto que permita la implementación.
- Diseñar e implementar un modelo hidrológico utilizando ANFIS.
- Validar los resultados obtenidos del modelo desarrollado.
- Analizar el comportamiento de otras series con respecto a las series obtenidas.
- Desarrollar un entorno gráfico que permita la organización y visualización de los datos (viejos y nuevos) y los resultados de los modelos, en línea (Internet).

1. MARCO TEÓRICO

Los modelos Autorregresivos (anexo A.5.1) comúnmente usados en la hidrológica son modelos lineales que no resultan del todo adecuados para resolver problemas con variables pobremente definidas, con alta incertidumbre y características altamente no lineales. Por el contrario, existen varias técnicas de computación emergente que pueden reproducir estas características mediante la emulación de aspectos intrínsecos del pensamiento humano, sin emplear métodos precisos de análisis cualitativos (que por lo general son computacionalmente muy costosos).

El modelo de inferencia ANFIS (Adaptative-Network-Based Fuzzy Inference System) [1] propuesto por Jyh-Shing Roger Jang en 1993, utiliza elementos de la lógica difusa y las redes neuronales adaptativas para autoajustar sus parámetros con cada iteración y así reducir el error cuadrático medio.

En esta sección se trata de dar una visión general de las técnicas de computación emergente necesarias para el correcto funcionamiento del modelo de inferencia ANFIS, así como otras herramientas necesarias para la comprensión del presente trabajo de grado.

1.1. Lógica Difusa

La Lógica Difusa (también llamada lógica borrosa) fue introducida inicialmente por Lotfi Asker Zadeh en 1965, su concepto central gira entorno a la existencia de los llamados “conjuntos difusos” los cuales son representados por una ecuación matemática frecuentemente conocida como función de pertenencia.

Mediante la observación de sistemas complejos, Lotfi se dio cuenta de lo que más tarde llamó principio de incompatibilidad: “Conforme la complejidad de un sistema aumenta, nuestra capacidad para ser precisos y construir instrucciones sobre su comportamiento disminuye hasta el umbral más allá del cual, la precisión y el significado son características excluyentes” [2], bajo este principio se concluye que el pensamiento humano no corresponde a variables numéricas sino lingüísticas. Si, por ejemplo, tenemos un grupo de personas con edades comprendidas entre 0 y 30 años, fácilmente podríamos separarlas en tres grupos de acuerdo a su edad: Niños, Adolescentes, Adultos.

Si entre el grupo tenemos a una persona de 12 años, podríamos decir que está culminando la niñez y comenzando la adolescencia, es decir, la persona pertenece en cierto grado a dos grupos, una afirmación como esta puede resultar muy engorrosa en la lógica normal, pero con la lógica difusa puede ser muy fácil, gracias al uso de los conjuntos difusos y las funciones de pertenencia.

1.1.1 Conjuntos Difusos

Un conjunto difuso se define como un conjunto de valores caracterizados por una función de pertenencia $\mu(x)$ establecida en un universo de discurso U, a diferencia de los conjuntos clásicos los conjuntos difusos no tienen una frontera estrictamente definida y en consecuencia la variable x puede tener valores de pertenencia graduales en el rango $\mu(x) \in [0,1]$.

Un ejemplo presentado por Lotfi para ilustrar el concepto de conjunto difuso, sería el conjunto de “hombres altos” [3]. Según la lógica clásica el conjunto “hombres altos” es un conjunto al que pertenecerían los hombres con una estatura mayor a una constante “X”.

Sea *A = Hombres Altos* y *X=valor constante;*

Si Y ≥ X, Entonces Y ∈ A

Si por ejemplo *X*=1.80 metros, entonces todos los hombres con una altura inferior a este valor quedarían fuera del conjunto "*A*". Así tendríamos que un hombre que mide 1.81 metros de estatura pertenecería al conjunto "*A*" (hombres altos), y en cambio un hombre que mida 1.79 metros de altura ya no pertenecería a ese conjunto. Sin embargo, no parece muy lógico decir que un hombre es alto y otro no cuando su altura difiere en tan solo dos centímetros, la Figura 1 ilustra los conceptos de la lógica difusa respecto a la lógica clásica.

El enfoque de la lógica difusa considera que el conjunto "hombres altos" es un conjunto que no tiene una frontera clara para pertenecer o no pertenecer a él mediante una función. Dicha función es conocida como "función de pertenencia", que define la transición de "alto" a "no alto" asignando a cada valor de altura un grado de pertenencia al conjunto entre 0 y 1.

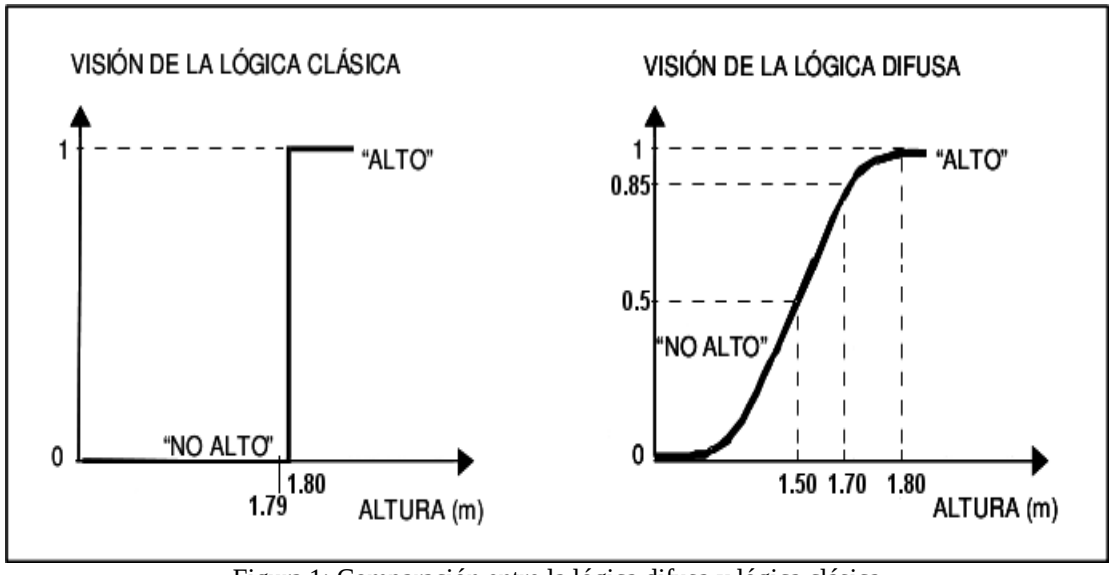


Figura 1: Comparación entre la lógica difusa y lógica clásica

Así por ejemplo para una función de pertenencia dada, un hombre que mida 1.70 m podría pertenecer al conjunto difuso “hombres altos” con un grado 0.85 de pertenencia, uno que mida 1.80 m un grado 1, y uno que mida 1.50 m un grado de pertenencia de 0.5. Visto desde esta perspectiva se puede considerar a la lógica clásica como un caso límite de la lógica difusa.

1.1.1.1 Funciones de Pertenencia

Las funciones de pertenencia son ecuaciones matemáticas que definen los conjuntos difusos. Sobre un universo de discurso denotado por X, la función de pertenencia de un conjunto difuso A, es generalmente denotada $\mu_A(x)$, asocia a cada elemento de $x \in X$ valores numéricos en un intervalo unitario.

$$\mu_A(x) : X \rightarrow [0,1] \tag{1}$$

De acuerdo a la expresión (1) un valor de pertenencia $\mu_A(x)=0$, corresponde a un caso donde x no pertenece al conjunto difuso, un valor $\mu_A(x) \in (0,1)$ corresponde a un nivel de pertenencia intermedio donde x pertenece en cierto grado al conjunto X sin pertenecer a él en su totalidad y un $\mu_A(x)=1$ corresponde a la total pertenencia de x al conjunto X.

A manera de ejemplo; una función de pertenencia en forma de campana puede ser definida como:

$$\mu_A(x) = \frac{1}{1+\left(\frac{x-c}{a}\right)^{2b}} \tag{2}$$

Donde los parámetros a, b y c determinan la longitud, pendiente y centro de la función de pertenencia, (Ver Figura 2).

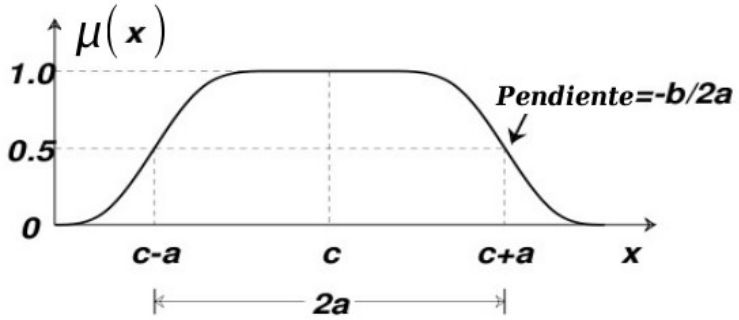
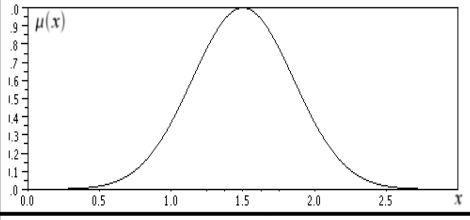
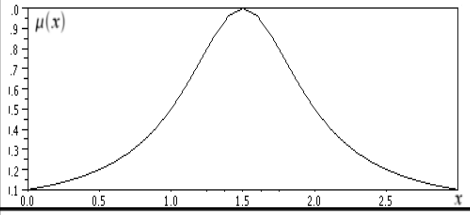
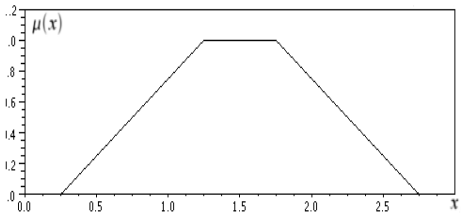
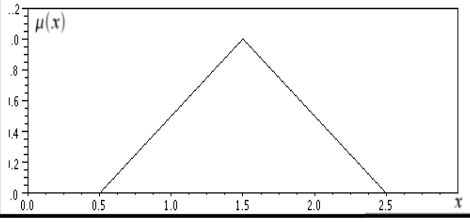


Figura 2: Interpretación gráfica de la ecuación (2) [1]

Tabla 1: Algunas funciones de pertenencia.

Nombre	Forma	Ecuación
Campana 1		$\mu(x)=e^{-\left(\frac{x-c}{a}\right)^{2b}}$
Campana 2		$\mu(x)=\frac{1}{1+\left(\frac{x-c}{a}\right)^{2b}}$
Trapezoidal		$\mu(x)=\begin{cases} 0 & si \quad x\geq z \quad ó \quad x\leq y \\ 1 & si \quad x<c+\frac{b}{2} \quad y \quad x>c-\frac{b}{2} \\ a\cdot(z-x) & si \quad x>c+\frac{b}{2} \\ a\cdot(x-y) & si \quad x<c-\frac{b}{2} \end{cases}$ donde: $z=c+b/2+1/a, \quad y=c-b/2-1/a$
Triangular		$\mu(x)=\begin{cases} 0 & si \quad x\geq z \quad ó \quad x\leq y \\ 1 & si \quad x=c \\ a\cdot(z-x) & si \quad x>c \\ a\cdot(x-y) & si \quad x<c \end{cases}$ donde: $z=c+1/a, \quad y=c-1/a$

En la práctica las funciones de pertenencia pueden adoptar varias formas y en consecuencia el número de parámetros que las definen pueden variar con respecto a la ecuación usada. La Tabla 1 muestra algunas funciones de pertenencia, observe que la función triangular corresponde a un caso especial de la función trapezoidal donde *b=0*.

1.1.1.2 Operaciones con Conjuntos Difusos

Algunas operaciones de los conjuntos ordinarios también son definidos en los conjuntos difusos, de modo que utilizan la misma notación [4], estos son:

a) Sean *A* y *B* dos conjuntos difusos definidos sobre *X*. La unión de estos conjuntos difusos denotada por $A \cup B$, es otro conjunto difuso *C* de *X*, tal que,

$$\mu_C(x) = max[\mu_A(x), \mu_B(x)] = \mu_A(x) \vee \mu_B(x), \forall x \in X \qquad (3)$$

donde el operador $\vee$ en la ecuación (3) denota el operador max (máximo).

En algunas bibliografías la operación Unión Estándar es también llamada operación OR (ver Figura 3 (a)).

b) Sean *A* y *B* conjuntos difusos definidos sobre *X*. La intersección de estos conjuntos difusos denotada por $A \cap B$, es otro conjunto difuso *D* de *X* tal que,

$$\mu_D(x) = min[\mu_A(x), \mu_B(x)] = \mu_A(x) \wedge \mu(x)_B, \forall x \in X \qquad (4)$$

donde el operador $\wedge$ en la ecuación (4) denota al operador min (mínimo).

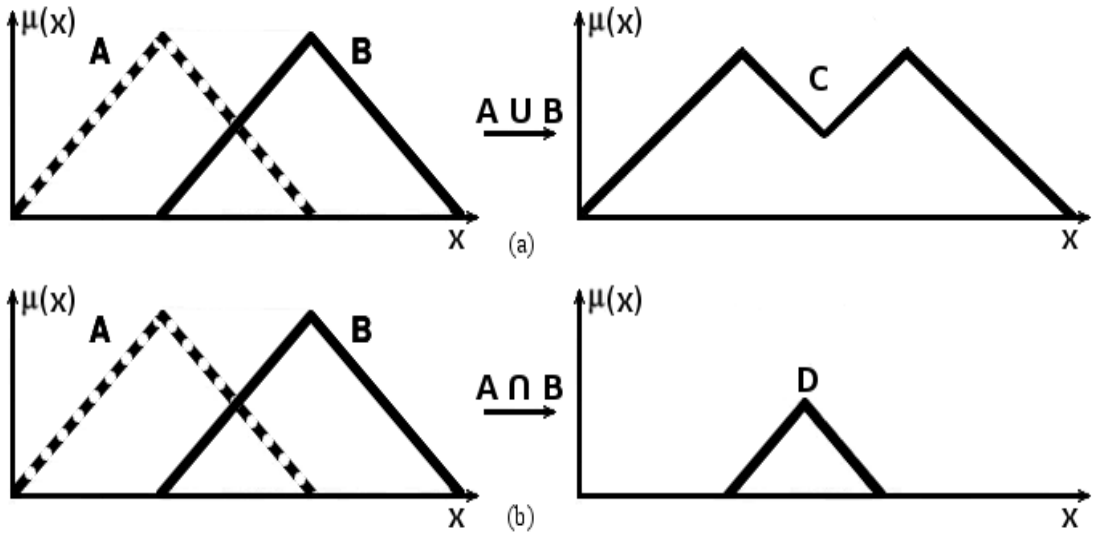


Figura 3: Operaciones Difusas: (a)Unión, (b) Intersección

En algunas bibliografías la operación Intersección Estándar es también llamada operación AND (ver Figura 3 (b)).

c) Una relación difusa entre el par (x, y) se define como un conjunto difuso sobre el espacio de producto cartesiano $X \times Y$. Si $X_1, X_2, \ldots, X_n$ son una colección de conjuntos, una relación difusa es un conjunto difuso definido sobre el espacio de su producto cartesiano $X_1 \times X_2 \times \ldots \times X_n$.

d) Sean A y B conjuntos difusos definidos sobre X y Y respectivamente. Su producto cruz o producto cartesiano $A \times B$ es una relación difusa R sobre el conjunto $X \times Y$, donde

$$\mu_R(x, y) = min[\mu_A(x), \mu_B(x)] \tag{5}$$

e) El complemento de un conjunto difuso A es denotado como $\bar{A}$ y es un conjunto difuso en el universo U cuya función de pertenencia se define como [22]:

$$\mu_{\bar{A}}(x) = 1 - \mu_{A}(x) \quad (6)$$

1.1.2 Reglas de Decisión Si-Entonces (if then rules)

Las reglas de decisión basan su funcionamiento en la idea de realizar una deducción mediante métodos computacionales. La deducción es posible mediante el establecimiento de reglas de la forma "Si *A* Entonces *B*" (If A Then B), donde "*A*" y "*B*" son variables lingüísticas, estas reglas son frecuentemente usadas para emular el razonamiento humano en la toma de decisiones.

Comúnmente se conoce a las variables lingüísticas pertenecientes al conjunto "*A*" como parte antecedente y las variables lingüísticas pertenecientes al conjunto "*B*" como parte consecuente. Por ejemplo:

Si la presión es alta **Entonces** el volumen es bajo

donde "presión" y "volumen" son variables lingüísticas, "alta" y "bajo" son valores lingüísticos caracterizados por funciones de pertenencia.

Otra forma de reglas Si-Entonces es la propuesta por Takagi, Sugeno y Kang [5,6], la cual involucra el uso de conjuntos difusos solo en la parte antecedente, pero definiendo la parte consecuente como una ecuación no difusa de la variable de entrada. Por ejemplo:

Si la velocidad es alta **Entonces** la fuerza = $k*(\text{velocidad})^2$

donde nuevamente "alta" en la parte antecedente es un valor lingüístico caracterizado

por una función de pertenencia, mientras que la parte consecuente es descrita por la ecuación no difusa de la entrada, “velocidad”.

Ambos tipos de reglas han sido ampliamente estudiadas y usadas en modelaje y control de sistemas. En base a los cuantificadores de la parte antecedente, cada regla Si-Entonces puede ser vista como una descripción local del sistema en consideración.

1.1.3 Razonamiento Difuso

El razonamiento difuso, también conocido como razonamiento aproximado, es un procedimiento de inferencia que atribuye una conclusión a partir de un conjunto de reglas Si-Entonces.

Según Jang [7] el razonamiento difuso en general se puede resumir en los siguientes cuatro pasos:

1. Cálculo de los Grados de Pertenencia

   Evalúa la entrada en los conjuntos difusos de la parte antecedente para encontrar los grados de pertenencia con respecto a cada una de sus funciones de pertenencia (este paso es comúnmente conocido como fusificación*)

2. Cálculo de los Pesos

   Combina los grados de pertenencia de la parte antecedente utilizando los operadores difusos AND u OR ($\cup$ o $\cap$) para determinar la fuerza de disparo (también conocido como peso) de cada regla.

* Nota: En el texto se usaran las terminologías “Fusificación” y “Desfusificación” provenientes del término “Fuzzy” en inglés, que significa borroso o difuso, esto es debido a que no existen traducciones exactas de dichos términos en español.

3. Composición de la Parte Consecuente

Genera las composiciones de la parte consecuente de cada regla en base a la fuerza de disparo.

4. Salida

Relaciona todas las composiciones de la parte consecuente para obtener una salida nítida (este paso es comúnmente conocido como desfusificación).

1.1.4 Sistemas de Inferencia Difusos (FIS) [1]

Los sistemas de inferencia difusos, se forman de una infraestructura computacional basada en los conceptos de los conjuntos difusos, las reglas Si-Entonces, y el razonamiento difuso. Se les han encontrado muchas aplicaciones en un gran número de campos tales como, el control automatizado, la clasificación de los datos, análisis de decisiones, sistemas expertos, predicción de series temporales, robótica y reconocimiento de patrones.

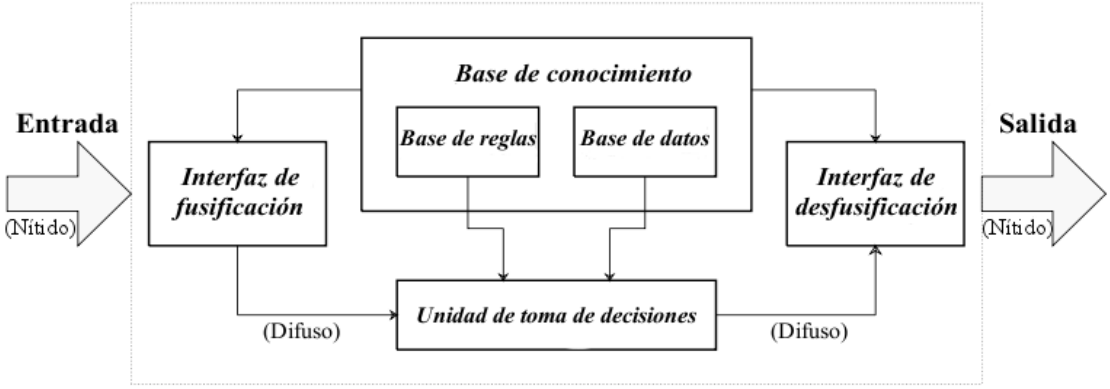


Figura 4: Sistema de Inferencia difuso [1]

Como se muestra en la Figura 4, un sistema de inferencia difuso se compone de 5 bloques bien definidos:

- Una base de reglas, la cual contiene las reglas Si-Entonces.

- Una base de datos que define los conjuntos difusos y las funciones de pertenencia usadas por las reglas Si-Entonces.

- Una unidad de toma de decisiones que realiza las operaciones de inferencia sobre las reglas.

- Una interfaz de fusificación que transforma las entradas nítidas del sistema en grados de pertenencia difusa.

- Una interfaz de desfusificación que transforma los resultados difusos de la inferencia en una salida nítida.

Normalmente la base de reglas y la base de datos son conjuntamente llamadas base del conocimiento.

El principal problema de los sistemas de inferencia difusos es que para su correcto funcionamiento es necesario que sus parámetros antecedentes y consecuentes sean previamente ajustados, bien sea por un experto o, en ausencia de éste, por ensayo y error.

Dependiendo del tipo de razonamiento difuso y las reglas Si-Entonces usadas, la mayoría de los sistemas de inferencia difusos puede clasificarse como:

1.1.4.1 Tipo Mamdani [7]

El sistema de inferencia Mamdani (1975) fue el primer sistema de inferencia difuso del que se tiene conocimiento, fue diseñado para controlar la velocidad de una

maquina de vapor, cuando los métodos de control convencionales fallaron en arrojar resultados satisfactorios, el conjunto de reglas lingüísticas usadas fue obtenida directamente de experimentados operadores humanos.

En la aplicación de Mamdani, dos sistemas de inferencia difusos fueron usados como dos controladores para generar la entrada de calor de la caldera y la apertura de válvula del cilindro del motor, ambos controladores operaban independientemente.

En general la inferencia tipo Mamdani, se caracteriza por generar una salida totalmente difusa mediante la aplicación del operador max sobre las composiciones, la Figura 5 resume las 5 desfusificaciones comúnmente usadas para obtener una salida nítida del modelo de este tipo de inferencia.

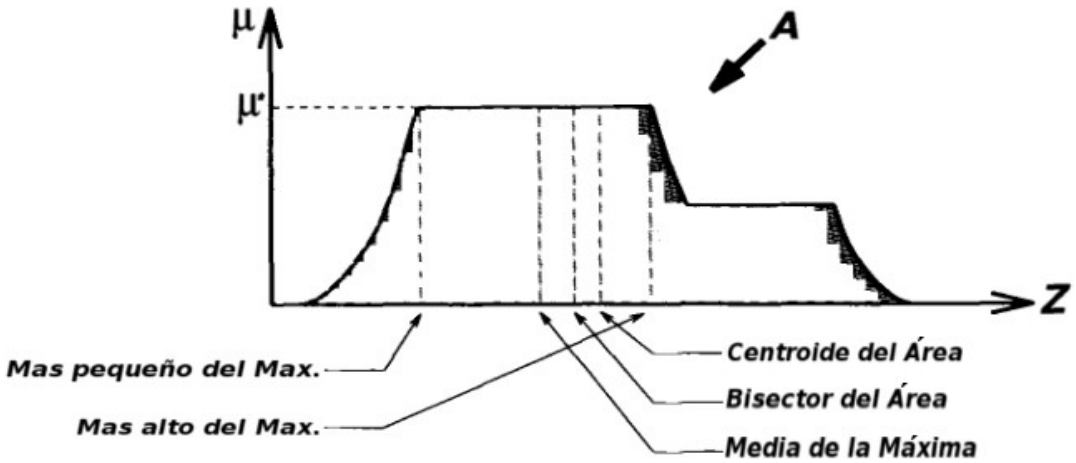


Figura 5: Métodos de desfusificación para razonamiento Mamdani [7]

1.1.4.2 Tipo Tsukamoto [7]

Propuesto en 1979, la parte consecuente de cada regla Si-Entonces es representada por una función de pertenencia monotónica. Como resultado la salida inferida de cada regla es un valor real inducido por la fuerza de disparo de la regla.

Como cada regla infiere una salida real, el modelo difuso de Tsukamoto relaciona la salida de cada regla mediante el método de promedio pesado y en consecuencia evita el paso de desfusificación (que puede consumir mucho tiempo). Sin embargo, este modelo no se usa con mucha frecuencia debido a que no resulta tan transparente como el modelo Mamdani o el modelo Sugeno (presentado más adelante).

1.1.4.3 Tipo Sugeno [7]

Fue propuesto en 1985, como un esfuerzo para desarrollar una aproximación sistemática a la generación de las reglas Si-Entonces a partir de los pares de muestras en un sistema dado. Generalmente se expresa de la siguiente forma:

$$\text{Si } x \text{ es } A \text{ y } y \text{ es } B \text{ entonces } f(x,y)=px+qy+r$$

Donde A y B son conjuntos difusos de la parte antecedente mientras $f(x,y)$ es la combinación lineal las variables de entrada y términos constantes. Cuando $f(x,y)$ es un polinomio de primer orden, el sistema de inferencia resultante es llamado modelo Sugeno de primer orden, cuando f es una constante entonces, tenemos un modelo Sugeno de orden cero, que también puede ser visto como una caso especial del modelo Mandani, en donde cada regla de la parte consecuente es un singleton difuso, ó un caso especial del modelo Tsukamoto, en el cual cada regla consecuente es especificada por una función escalón centrada en la constante.

Como las salidas de cada regla son nítidas, la salida total es obtenida mediante promedios pesados. En la práctica, el promedio de pesos es algunas veces remplazado con el operador de suma de pesos, con el fin de reducir aún más necesidades computacionales, en especial durante el entrenamiento del sistema de

inferencia difuso, sin embargo esta simplificación podría producir la pérdida del significado lingüístico de la función de pertenencia a menos que la suma de los pesos sea cercana a la unidad. En la Figura 6 se realiza una comparación entre los tres tipos de razonamiento difuso.

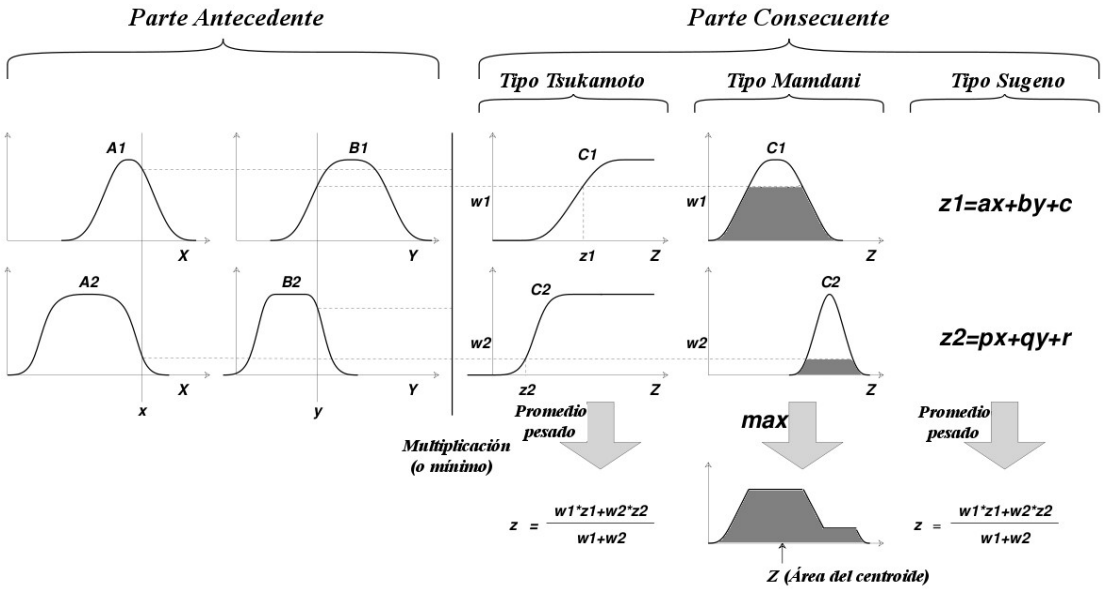


Figura 6: Tipos de razonamiento difuso comúnmente usados [7]

Gracias a la gran versatilidad de funciones posibles en la parte antecedente y su salida no difusa, el modelo Sugeno es por mucho, el candidato más popular para el modelado difuso en sistemas con pares de muestras de datos.

1.2. Redes Neuronales Artificiales

Las redes neuronales artificiales surgen del esfuerzo de entender y modelar matemáticamente los procesos del pensamiento y las funciones del cerebro humano, el primer modelo fue propuesto por McCulloch y Pitts en 1940 dando origen a los modelos conexionistas y definiendo formalmente la neurona en 1943 como una máquina binaria con varias entradas y salidas. En 1949 Hebb, define dos conceptos muy importantes basándose en investigaciones psicofisiológicas:

1) El aprendizaje se localiza en las sinapsis o conexiones entre las neuronas.

2) La información se representa en el cerebro mediante un conjunto de neuronas activas o inactivas.

Las hipótesis de Hebb, se sintetizaron en lo que más tarde sería conocido como la regla de aprendizaje del Hebb, que sigue siendo usada en los modelos actuales. Esta regla nos dice que los cambios en los pesos de las sinapsis se basan en la interacción entre las neuronas pre y postsinápticas.

En la actualidad, las redes neuronales se usan en todo tipo de aplicaciones entre las que se puede citar: reconocimiento de patrones, voz y vídeo, compresión de imágenes, estudio y predicción de sucesos muy complejos aplicaciones de apoyo a la medicina y en general todo tipo de aplicaciones que necesiten el análisis de grandes cantidades de datos, etc. [8].

Las redes neuronales artificiales buscan imitar el comportamiento de las redes neuronales biológicas definiendo a cada neurona como una unidad de procesamiento compuesta de:

- Entradas (dendritas) x_j, que pueden ser las salidas (axones) de otras neuronas
- Pesos (sinapsis) W_{ij}, asociados a las entradas que son la conexión que entra a la neurona "*i*" y sale a la neurona "*j*".
- Una regla de propagación: utiliza las entradas x_j y los pesos

sinápticos W_{ij} para realizar algún tipo de operación que permita obtener el valor del potencial postsináptico (valor que es función de las entradas y los pesos y que se utiliza en último término para realizar el procesamiento). Comúnmente se realiza las sumatoria de las entradas tomando en cuenta el peso de cada una [9].

Resultando así el término net_i de la ecuación (7) en una suma ponderada,

aunque otras operaciones también son posibles.

$$net_i = \sum_j W_{ij} . X_j \qquad (7)$$

En la Figura 7 se resume la estructura de una neurona artificial.

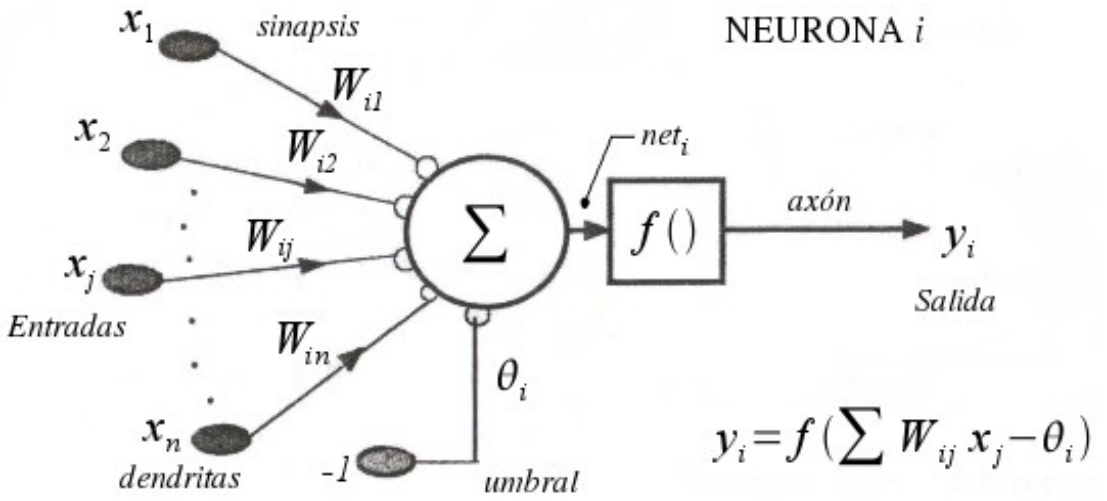
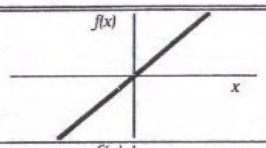
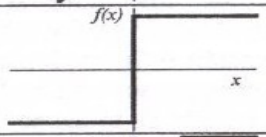
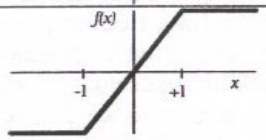
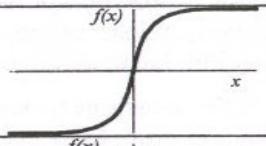
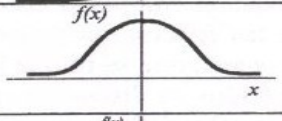
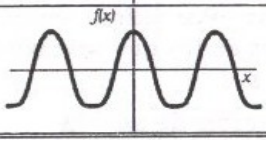


Figura 7: Estructura de una neurona artificial [9]

- Una función de activación f() cuyo objetivo es ajustar (filtrar) la salida de la regla de propagación para que opere en el dominio asignado por la neurona. Dependiendo de las características de la red, se usa una función de activación u otra. La tabla 2 muestra algunas de las funciones de

activación más usadas [9].

Tabla 2: Algunas funciones de activación [9]

	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
Lineal a tramos	$y = \begin{cases} -1, & si\ x < -l \\ x, & si\ +l \le x \le -l \\ +1, & si\ x > +l \end{cases}$	$[-1, +1]$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0, +1]$ $[-1, +1]$	
Gaussiana	$y = Ae^{-Bx^2}$	$[0,+1]$	
Sinusoidal	$y = A\,\mathrm{sen}(\omega x + \varphi)$	$[-1,+1]$	

1.2.1 Arquitectura de las Redes Neuronales Artificiales

Se puede ver una red neuronal como una estructura guiada y ponderada donde cada uno de sus nodos son neuronas artificiales y los arcos que unen los nodos son las conexiones sinápticas. Dichos arcos son unidireccionales de modo que la información se propaga en un único sentido, desde una neurona presináptica (neurona origen) a una neurona postsináptica (neurona destino).

Por otro lado, la conexión de los arcos es ponderada, es decir, las conexiones tienen asociado un valor (el peso), que indica la importancia de esa conexión con respecto al resto de las conexiones. Si dicho peso es positivo la conexión se dice que es excitadora, mientras que si es negativa se dice que es inhibidora [9].

La distribución de las neuronas en una red neuronal se realiza formando niveles o capas. Dependiendo de su ubicación dentro de la red se puede definir como:

- Capa de entrada, cuya única función es recibir la información del exterior.

- Capa oculta, que realiza operaciones sobre la información y permite la comunicación con las otras capas.

- Capa de salida, que envía la información al exterior dando la respuesta de la red neuronal.

Según algunos autores, las capas de entrada y de salida no son válidas para producir procesamiento y son usadas solo como sensores. Sin embargo la capa de salida frecuentemente realiza operaciones matemáticas (sumatoria, productoria, etc.) que también forman parte del procesamiento.

1.2.2 Clasificación de las Redes Neuronales

La Figura 8 muestra la clasificación de algunas redes neuronales en base al número de capas de la red, esta clasificación es comúnmente aceptada en la literatura, sin embargo, no resalta las características principales de cada red.

En este texto se clasificaran las redes neuronales en base a sus mecanismos de aprendizaje ya que se considera que dicha clasificación se ajusta mejor a los fundamentos teóricos que se describirán más adelante.

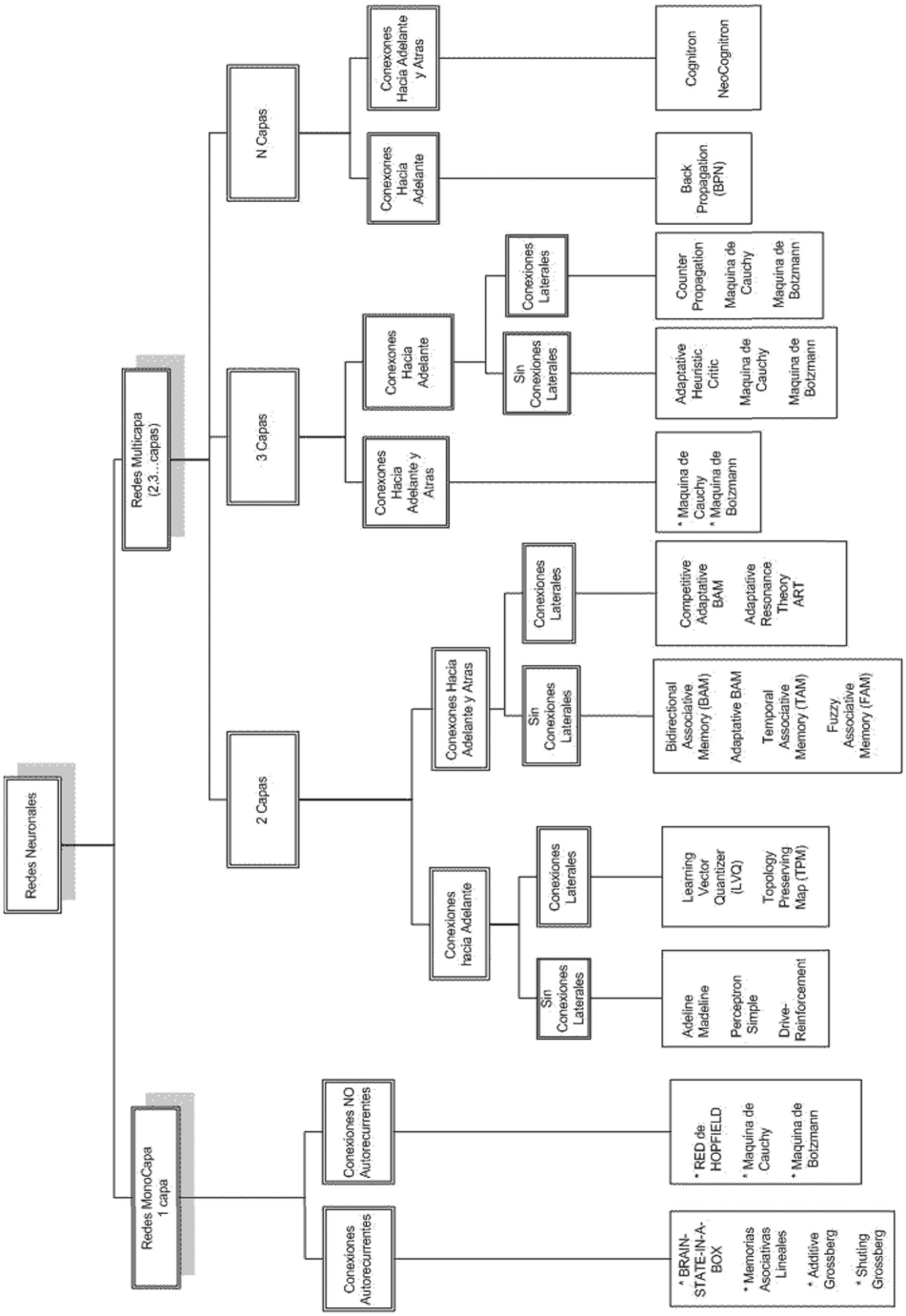


Figura 8: Algunas redes neuronales conocidas [8]

1.2.2.1 Aprendizaje Supervisado

En este tipo de aprendizaje se proporciona a la red neuronal una serie de

ejemplos consistentes en unos patrones de entrada, junto con la salida que debería dar la red. El proceso de entrenamiento consiste en el ajuste de los pesos para que la salida de la red sea lo más parecida posible a la salida deseada, es por ello que en cada iteración se use alguna función que nos dé cuenta del error o el grado de acierto que esta cometiendo la red. Algunas redes que usan este aprendizaje supervisado son el perceptrón, el perceptrón multicapa y la red de Hopfield.

1.2.2.2 Aprendizaje No Supervisado

En este tipo de aprendizaje se presenta a la red una serie de ejemplos pero no se presenta la respuesta deseada. En consecuencia la red busca reconocer regularidades en el conjunto de entradas y estimar una función densidad de probabilidad que describa la distribución de patrones en el espacio de entrada.

Entre los distintos tipos de aprendizaje no supervisado podemos distinguir, el aprendizaje por componentes principales y el aprendizaje competitivo.

1.2.2.2.1 Aprendizaje por Componentes Principales

Se basa en hallar las características principales de los componentes comunes de muchos patrones de entrada, para ello un pequeño número de neuronas coopera en la representación del patrón de entrada.

1.2.2.2.2 Aprendizaje Competitivo

En el aprendizaje competitivo, las neuronas pugnan entre sí, para representar a una clase o patrón de entrada.

La neurona seleccionada, es aquella cuyos pesos incidentes se asemejan más al patrón de entrada. El aprendizaje consiste en reforzar las conexiones de la unidad

ganadora y debilitar las otras, para que los pesos de la unidad ganadora se asemejen cada vez más al patrón de entrada.

1.2.2.3 Aprendizaje Híbrido

Es una mezcla de los anteriores. Unas capas de la red tienen un aprendizaje supervisado y otras capas de la red tienen un aprendizaje de tipo no supervisado.

Este tipo de entrenamiento se encuentra presente en las funciones de base radial (RBF en inglés).

1.2.2.4 Aprendizaje Reforzado

Es un aprendizaje con características del supervisado donde la información proporcionada a la red es mínima, no se proporciona una salida deseada, pero se le indica a la red en cierta medida el error que comete, aunque sea por un error global.

Este tipo de aprendizaje se basa en la noción de condicionamiento por refuerzo, es decir, se aprenden las conductas reforzadas positivamente y las conductas castigadas o reforzadas negativamente.

1.2.3 Redes Adaptativas

Las redes adaptativas pueden definirse como una infraestructura unificada de casi todos los tipos de redes neuronales con capacidad de aprendizaje supervisado, consisten de redes estructurales compuestas por nodos conectados mediante enlaces direccionales. Todos o parte de los nodos son adaptativos lo que significa que las salidas de esos nodos dependen de parámetros variables pertenecientes a esos mismos nodos. Las reglas de aprendizaje especifican cómo esos parámetros deben ser actualizados para disminuir la medición del error, que por lo general consiste en una

expresión matemática encargada de evaluar la discrepancia entre la salida presente en la red y la salida deseada.

La regla básica de aprendizaje de las redes adaptativas se basa en el método del gradiente descendente y la regla de la cadena inicialmente propuesta por Werbos en los años 70 [10], este método también es conocido como propagación hacia atrás (backpropagation) gracias al trabajo de Rumelhart [11] en 1986. Para una descripción más detallada de este método ver el Anexo A.1.

Los parámetros de una red adaptativa se distribuyen dentro de sus nodos, de modo que cada nodo tiene un conjunto de parámetros locales. Si un nodo posee un conjunto vacío de parámetros, dicho nodo se representa con un círculo y se denomina nodo ajustado. Si por otra parte el nodo tiene un conjunto de parámetros no vacíos se representa mediante un cuadrado y se denomina nodo adaptativo, en la Figura 9 podemos apreciar un ejemplo.

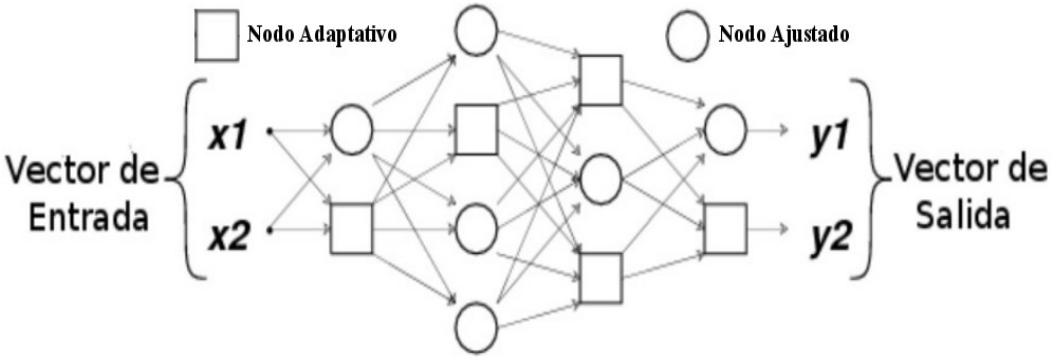


Figura 9: Ejemplo de una red adaptativa [7]

Normalmente las redes adaptativas son redes multicapa, donde su conexión puede ser hacia adelante o realimentada. La escogencia de los nodos adaptativos, depende de la relación entrada-salida deseada.

1.3. Modelo de Inferencia ANFIS

Propuesto en 1993 por Jyh-Shing Roger Jang [1], es un sistema de inferencia difuso implementado en la infraestructura de una red adaptativa, para actualizar los parámetros de sus nodos adaptativos se utiliza el método de aprendizaje híbrido del mismo autor (ver Anexo A.2). El modelo ANFIS puede ajustar los conjuntos difusos en base al conocimiento humano integrado, mediante reglas Si-Entonces y la relación entrada(s) - salida del sistema.

1.3.1 Arquitectura ANFIS

Las redes ANFIS se constituyen de redes adaptativas con conexión hacia adelante, su arquitectura consiste de varias capas con sus respectivas funciones nodales. Considerando la arquitectura de la Figura 10 (b):

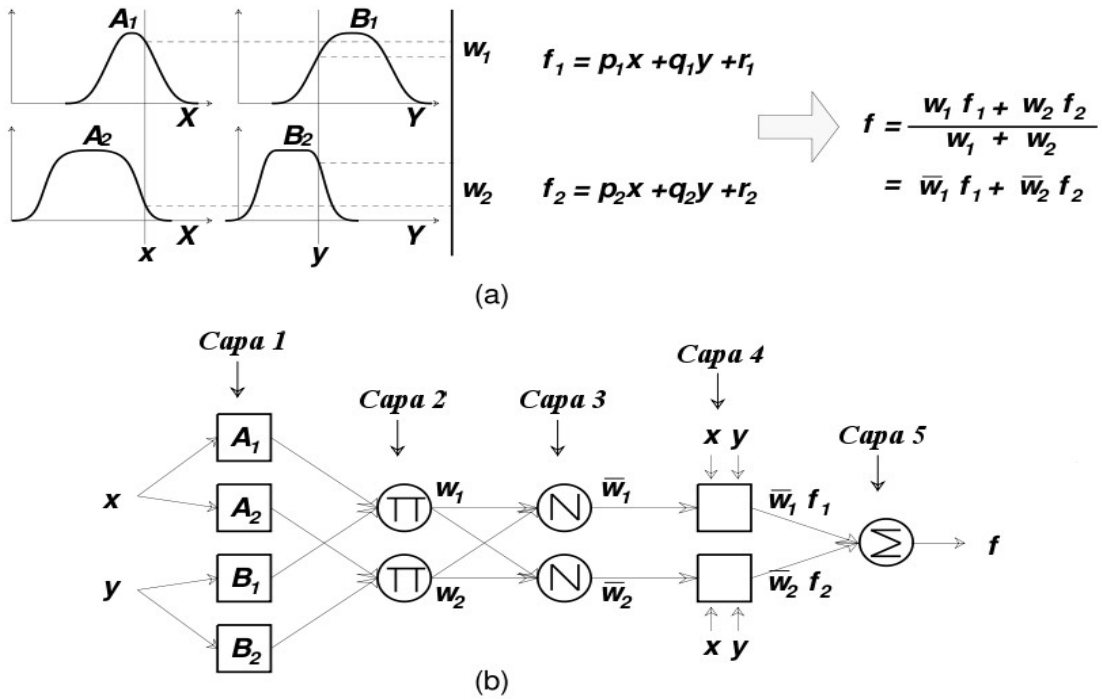


Figura 10: (a) Razonamiento tipo Sugeno; (b) Arquitectura ANFIS equivalente para un razonamiento Sugeno. [1]

donde el sistema de inferencia difuso bajo consideración utiliza un razonamiento tipo

Sugeno de primer orden de dos entradas *x* y *y*, con dos funciones de pertenencia por entrada (FP) y una salida *f*, un conjunto de reglas común sería de la forma.

Si *x* es A_1 y *y* es B_1, entonces $f_1=p_1x+q_1y+r_1$,
Si *x* es A_2 y *y* es B_2, entonces $f_2=p_2x+q_2y+r_2$.

Para un modelo ANFIS las funciones nodales de una misma capa pertenecen a la misma familia de funciones de la manera que se describe a continuación.

- Capa 1.- Todo nodo *i* en esta capa es un nodo adaptativo con una función de activación.

$$\begin{aligned} O_{1,i} &= \mu_{A_i}(x) \quad , para \;\; i=1,2...k/2 \\ O_{1,i} &= \mu_{B_{i-k/2}}(y), para \;\; i=k/2+1,...k \end{aligned} \tag{8}$$

  donde k=(*número de FP*)∗(*número de entradas*), *x* o *y* es la entrada correspondiente al nodo *i* y A_i o B_{i-2} es una variable lingüística asociada al nodo. En otras palabras $O_{1,i}$ corresponde al grado de pertenencia de la entrada *x* o *y* con respecto a la función de pertenencia μ_A o μ_B.

  Los parámetros adaptativos de esta capa corresponden a los parámetros usados para la construcción de la función de pertenencia, también conocidos como parámetros antecedentes.

- Capa 2.- Cada nodo de esta capa es ajustado y su función de salida es el producto de la combinación lineal de las salidas de la etapa anterior.

$$O_{2,i} = w_i = \mu_{A_j}(x)\,\mu_{B_g}(y) \tag{9}$$

donde $i=[1,k]$; $j=[1,k/2]$;$g=[1,k/2]$; $k=$(*número de FP)* $^{(número\ de\ entradas)}$, la salida de cada nodo representa la fuerza de disparo (peso) de cada regla. En general se puede usar cualquier operador que produzca la operación "AND".

- Capa 3.- Cada nodo en esta capa es ajustado y su función de salida normaliza el peso del *i*-ésimo nodo con respecto a la sumatoria de todos los pesos.

$$O_{3,i}=\overline{w_i}=\frac{w_i}{\sum_{j=1}^{k} w_j}, i=1,2,\ldots k \tag{10}$$

donde *k=(número de FP)* $^{(número\ de\ entradas)}$

- Capa 4.- Cada nodo *i* en esta capa es adaptativo con una función

$$O_{4,i}=\overline{w_i}\, f_i=\overline{w_i}(p_i x+q_i y+r_i), \tag{11}$$

donde p_i, q_i y r_i son el conjuntos de parámetros del nodo también conocidos como parámetros consecuentes

- Capa 5.- Esta capa está compuesta por un solo nodo que realiza la suma total de todas las señales entrantes de la etapa anterior.

$$O_{5,1}=\sum_i \overline{w_i} f_i \tag{12}$$

Las capas anteriores describen la estructura de una red adaptativa funcionalmente equivalente a un modelo difuso tipo Sugeno de primer orden, sin embargo se debe destacar que esta estructura no es única, de modo que se pueden condensar varias funciones en una misma capa, siempre que los nodos de cada capa puedan ejecutar funciones significativas modularmente.

Se puede adaptar el modelo a un sistema Tsukamoto cambiando la red adaptativa como se muestra en la Figura 11(b), como consecuencia la salida del sistema viene dada por la salida de cada regla (f_i, donde *i* puede tomar valores entre *1* y el *número de reglas* $^{número\ de\ entradas}$) y la fuerza de disparo (peso).

Para el caso Mamdani se puede construir un modelo ANFIS con algunas restricciones dependientes de la composición usada. Sin embargo el modelo resultante es mucho más complicado que los obtenidos por Sugeno o Tsukamoto y no implica mejores capacidades de aprendizaje ni mejores resultados.

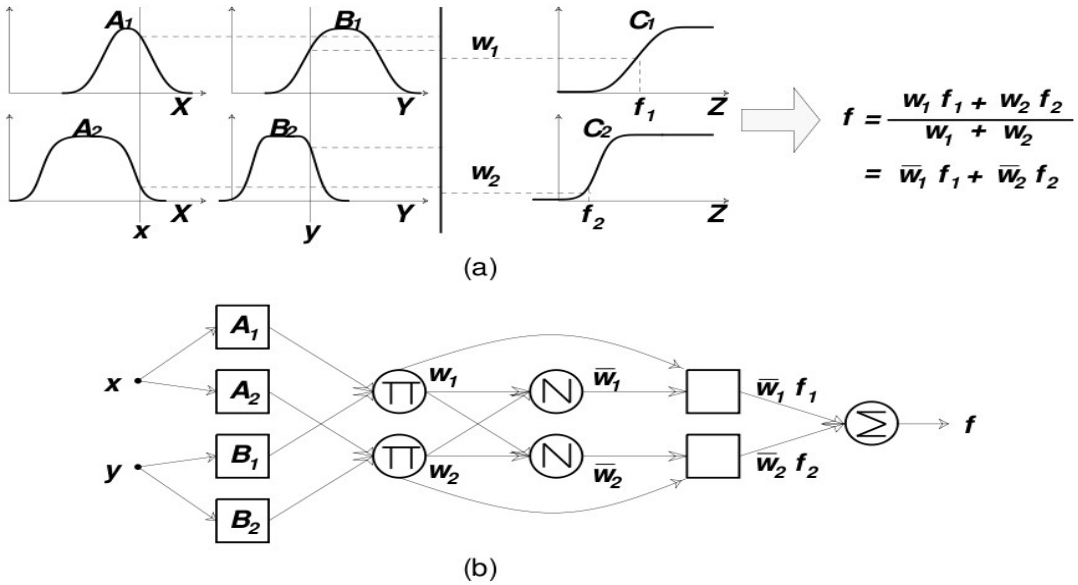


Figura 11: (a) Razonamiento tipo Tsukamoto; (b) Arquitectura ANFIS equivalente para un razonamiento Tsukamoto. [1]

1.3.2 Entrenamiento del Modelo ANFIS [1]

Este consiste en la actualización de los nodos adaptativos de la red mediante el método de aprendizaje híbrido descrito por Roger Jang [1]. La tabla 3 resume el entrenamiento del modelo ANFIS para cada iteración.

En el paso adelante los parámetros antecedentes están ajustados y los parámetros consecuentes se calculan por estimación de mínimos cuadrados.

Tabla 3: Dos pasos en el aprendizaje híbrido para el modelo ANFIS [1].

——	Paso hacia Adelante	Paso hacia Atrás
Parámetros antecedentes	Ajustados	Gradiente Descendente
Parámetros consecuentes	Estimación por mínimos cuadrados	Ajustados
Señales	Salidas Nodales	Tasa de error

En el paso hacia atrás los parámetros consecuentes (previamente ajustados por el paso adelante), están ajustados y los parámetros antecedentes se determinan mediante el método del gradiente descendente (también llamado “propagación hacia atrás”, ver Anexo A.1 para más información).

Estos pasos se repiten hasta que el número de iteraciones sea alcanzado o hasta que el error cuadrático medio sea menor al deseado.

La principal ventaja de este método es que los parámetros consecuentes siempre son óptimos bajo la condición de que los parámetros antecedentes estén ajustados, en este sentido el método de aprendizaje híbrido converge mucho más rápido debido a que reduce los parámetros buscados por el método de propagación hacia atrás puro.

Existen otros métodos de aprendizaje propuestos en la bibliografía, que utilizan variantes del método del gradiente descendente u otras técnicas de optimización tales como el gradiente conjugado, propagación hacia atrás de segundo orden, propagación rápida, algoritmos genéticos (método que no utiliza derivadas) etc.

1.4. Paquetes de Software

En este proyecto se utilizara el término “Software libre” en aquellas herramientas cuya licencia permita a cualquier persona o institución descargar y manipular el código fuente, dichas licencias pueden variar de país en país, pero la mayoría son compatibles con alguna de las establecidas por la Free Software Foundation (FSF) [16] o la Open Source Initiative (OSI) [17].

1.4.1 Scilab

Es un paquete libre (bajo la licencia CeCILL [14]) de software científico para la computación numérica, provista con un poderoso entorno computacional ideado para aplicaciones de ciencias e ingeniería. Desde 1994 Scilab es distribuido gratuitamente con su código fuente desde internet y actualmente es usado en entornos educativos e industriales alrededor del mundo. Scilab incluye cientos de funciones matemáticas con la posibilidad de agregar programas interactivos de varios lenguajes (C, C++, FORTRAN, etc.), también posee sofisticadas estructuras de datos (incluyendo, listas, polinomios, funciones racionales, sistemas lineales, etc.), un intérprete y un lenguaje de programación de alto nivel.

En el presente trabajo se utilizó la versión 5.1.1 de Scilab la cual puede obtenerse para múltiples plataformas (Linux, MacOSX, Windows, etc.) desde su página principal [15].

1.4.2 GCC

GNU Compiler Collection (colección de compiladores GNU), son un conjunto de compiladores creados por el proyecto GNU, distribuido por la Free Software Foundation (FSF) [16] bajo la licencia GPL [18]. Estos compiladores se consideran estándar para los sistemas operativos derivados de UNIX (Linux, BSD, etc.) de código abierto y algunos propietarios como Mac OS X.

En el presente trabajo se usó la versión gcc-4.3.2 para la compilación de los programas hechos en lenguaje C ANSI (American National Standard Institute).

1.4.3 PHP

Es un lenguaje interpretativo (similar a C o Perl) de propósito general, diseñado especialmente para desarrollo web y puede ejecutarse de forma embebida dentro de código HTML [19].

Entre sus principales características podemos destacar su soporte a la programación orientada a objetos, su compatibilidad con MySQL, XML, Sqlite y SOAP, entre otros.

En el presente trabajo se usará PHP 5.2.6 para el desarrollo de la interfaz web.

CAPÍTULO II

2. DISEÑO DE HERRAMIENTAS DE ENTRENAMIENTO PRONÓSTICO Y DIVULGACIÓN

Actualmente el proyecto PROCEDA cuenta con 9 estaciones hidrometeorológicas de medición automática, cada una con su respectivo registro de series históricas. El estudio y divulgación de dichas series es realizado por un operador de manera manual, lo cual limita su acceso a la comunidad científica e imposibilita su utilización en sistemas de prevención y alerta temprana. Bajo esta consideración en el presente capítulo se propone la creación e implementación de una herramienta de pronóstico que permita al operador evaluar los datos rápidamente y una herramienta web que permita la divulgación de los pronósticos y las series usadas por estos. La Figura 12 muestra el diagrama de bloques propuesto en base a la infraestructura disponible por el proyecto PROCEDA.

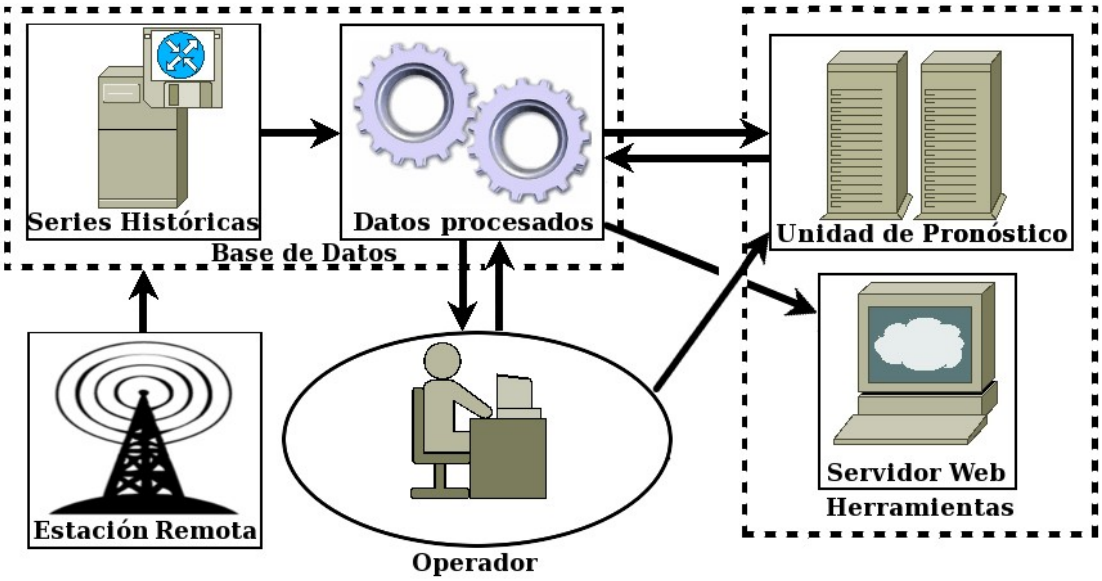


Figura 12: Diagrama de bloques general del sistema

2.1. Estaciones Remotas

PROCEDA cuenta con 2 estaciones hidrométricas para mediciones de nivel y caudal de cursos de agua y 7 estaciones pluviométricas para mediciones de precipitación y otras variables climatológicas, cada estación posee un modulo celular que es usado por el operador para la obtención remota de los datos hidrológicos.

Idealmente las series históricas deberían actualizarse en vivo pero dado la lentitud de los datos y los costos asociados, la implementación en línea se considera poco viable.

2.2. Base de Datos

De momento, el proyecto no cuenta con un manejador de base datos operativo, de modo que el arreglo y procesamiento de los datos debe realizarse manualmente, la implementación y puesta a punto de la base de datos escapa del alcance de esta tesis pero su eventual uso es obligatorio para el correcto funcionamiento de las herramientas a desarrollar.

Considerando que los datos de las estaciones remotas son almacenados en formato ".csv" cumpliendo con la normativa ANSI SQL las herramientas a desarrollar serán implementadas en Scilab y PHP 5, también compatibles con el estándar ANSI SQL, se recomienda construir la base de datos bajo una plataforma MySQL o compatible.

La base de datos deberá ordenar cronológicamente los datos de las series históricas correspondientes a una misma estación, permitiendo la reestructuración de las series para distintos intervalos de tiempo.

El sistema de alerta (propuesto más adelante) requiere el uso de series

horarias de precipitación para el cálculo de las precipitaciones ponderadas en un tiempo de vida medio de 1,5 y 72 horas (*Pw72* y *Pw1,5)*, cumpliendo con el método del comité [23] descrito en el anexo A.4, la base de datos deberá calcular las precipitaciones acumuladas y la alerta correspondiente en base a los registros históricos y los últimos datos de las estaciones remotas, dichos cálculos deberán incorporarse a la base de datos existente para ser usados por la herramienta de pronóstico, almacenando sus resultados y permitiendo la visualización y descarga de series mediante la herramienta web.

2.3. Diseño de Herramientas de Entrenamiento y Pronóstico

2.3.1 Herramienta de Entrenamiento del ANFIS

Consiste de un entorno gráfico que facilita al usuario la configuración del modelo sin que éste comprenda todos los procesos involucrados, posee un entorno intuitivo desarrollado en Scilab, que ejecuta el modelo ANFIS mediante librerías dinámicas hechas en ANSI-C.

La interfaz desarrollada permite realizar de manera sencilla los procesos mostrados en la Figura 13:

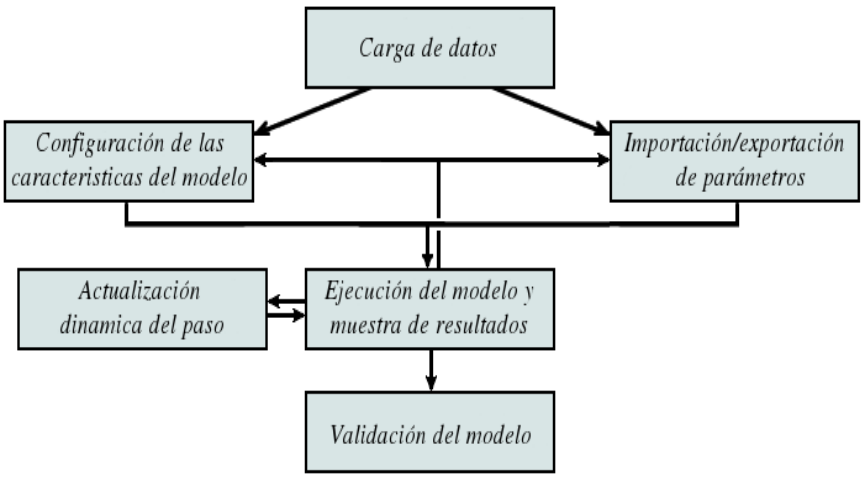


Figura 13: Procesos implementados dentro de la herramienta de entrenamiento ANFIS

2.3.1.1 Carga de Datos

Los datos a cargar corresponden a los llamados datos de entrenamiento y datos de validación, el usuario deberá suministrar las direcciones (rutas; el camino donde se encuentran los archivos) de cada uno de estos.

Los datos de entrenamiento son obligatorios para el aprendizaje del modelo, a partir de ellos el programa será capaz de determinar el número de muestras *"p"* de la serie y el número de entradas *"k"*, basándonos en criterios previamente establecidos, se considerará que la última columna de la matriz de muestras será la salida del sistema.

Los datos de validación no son necesarios para el aprendizaje del modelo pero serán de utilidad para comprobar el correcto funcionamiento de éste, los datos de validación deben tener las mismas variables de entrada que las muestras de entrenamiento y en el mismo orden, la salida por otra parte es opcional.

El toolbox MySQL de Scilab [21] permite hacer consultas a una base de datos almacenando su respuesta en un arreglo, se considera una buena opción para obtener los datos de entrenamiento y/o validación directamente de la base de datos.

2.3.1.2 Configuración de las Características del Modelo

El usuario deberá seleccionar un tipo de función de pertenencia *"FP"* y suministrar un número entero *"L"* que indicará el número de funciones de pertenencia que se usarán por cada entrada, algunos FIS (sección 1.1.4) permiten distintos números de funciones de pertenencia por entrada pero el modelo desarrollado contempla el mismo número para cada entrada.

Se contempla el uso de los 4 tipos de funciones de pertenencia descritos en la tabla 1 de la sección 1.1.1.1:

- *FP=1* correspondiente a la ecuación Campana 1
- *FP=2* corresponde a la ecuación Campana 2
- *FP=3* corresponde a la ecuación Trapezoidal
- *FP=4* corresponde a la ecuación Triangular

Ya asignados los valores *FP* y *L*, el programa generará la configuración inicial de los parámetros antecedentes, en base a los siguientes criterios:

- Los centros deben empezar en el valor más bajo del universo de discurso "*U*" y culminar en su valor más alto, el centro de una función y el centro de la función siguiente debe satisfacer la ecuación

$$c_2 - c_1 = \frac{Máximo\ E(:,k) - Mínimo\ E(:,k)}{L-1}, \qquad (13)$$

donde *L* y *k* son constantes.

- Los parámetros *a* y *b* (solo "*a*" para la función triangular) deben ser tales que una función de pertenencia y la función siguiente solo se intercepten en un punto del universo de discurso "*U*", al evaluar dicho punto en cualquiera de las dos funciones el resultado debe ser igual a 0,5.

Por ejemplo: Sea una matriz de entrada "*E(p,k)*", se evalúan los componentes de la primera entrada (*k=1*) el universo de discurso "*U*" mediante *U*=[mínimo valor de la entrada, máximo valor de la entrada].

Para *L=3*, *FP=2* con *U=[0,10]* (ver Figura 14).

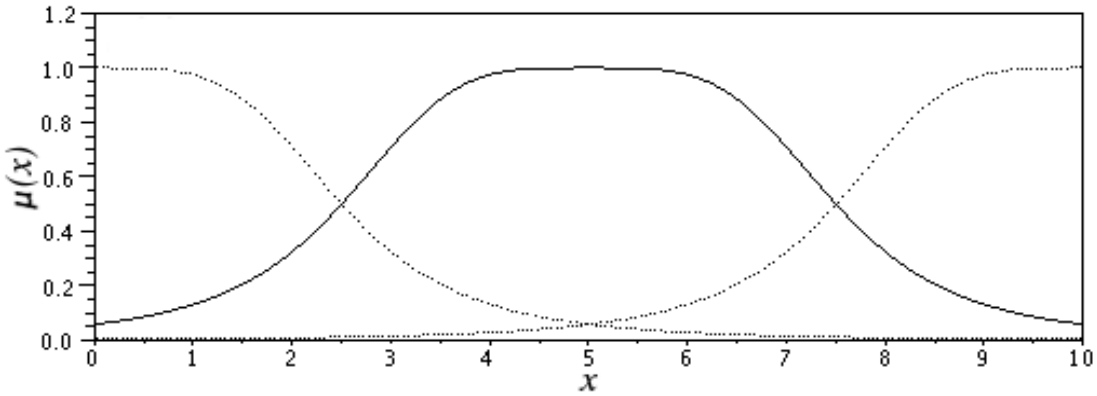


Figura 14: Configuración inicial típica para los parámetros *L=3*, *FP=2* con *U=[0,10]*

El mismo procedimiento debe repetirse en todas las entradas "*k*" de la matriz de muestras.

Al final se obtendrá una matriz con todos los parámetros antecedentes iníciales del sistema. Considerando el ejemplo de la Figura 14 donde *L=3*, *FP=2* con *U=[0,10]* con una entrada adicional tendríamos la matriz:

$$\begin{matrix} & \overbrace{\qquad\qquad}^{k=1} & & & \overbrace{\qquad\qquad}^{k=2} & & \\ \begin{bmatrix} a \\ b \\ c \end{bmatrix} = & \begin{bmatrix} 2.5 & 2.5 & 2.5 & 0.55 & 0.55 & 0.55 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & 5 & 10 & 1 & 2.1 & 3.2 \end{bmatrix} & & & & & \end{matrix} \qquad (14)$$

entrada 1 entrada 2

Como puede observarse el universo de la segunda entrada seria *U=[1, 3.2]*, y *L=3* indica el uso de tres funciones de pertenencia por cada entrada *(k)*, donde cada columna representa los parámetros *a*, *b* y *c* de dichas funciones de pertenencia.

2.3.1.3 Ejecución del modelo y muestra de resultados

El usuario debe seleccionar entre el aprendizaje híbrido fuera de línea *(tpo=1)* o el aprendizaje en línea *(tpo=2)*, el número de iteraciones máximas *(Itr), el paso inicial (ka)* y el error cuadrático medio requerido *(err)*.

Si al momento de ejecutar el modelo, falta algún parámetro *(L, FP, Itr, ka err, tpo)* por definir, entonces el modelo utilizará su(s) valor(es) por defecto (*L=3; FP=2; Itr=10; ka=0.01; err=0; tpo=1).*

Al terminar la ejecución del modelo, el programa mostrará la evolución del error con respecto al número de iteraciones, así como los parámetros antecedentes y consecuentes que arrojaron el error cuadrático medio más pequeño. En el capítulo III se indica como observar estos parámetros, bien sea por la interfaz o por la consola de Scilab.

Si se ejecuta el entrenamiento híbrido fuera de línea *(tpo=1)*, el modelo se ejecuta solo una vez y los resultados solo pueden observarse cuando finalice el entrenamiento.

Si se ejecuta el entrenamiento híbrido en línea *(tpo=2)*, el modelo se ejecuta tantas veces como muestras tenga la matriz de entrenamiento, observe que para este caso no se necesita realizar más de una iteración por muestra de modo que *Itr y err son* despreciables, la actualización de los parámetros ocurre con cada muestra de datos.

El entrenamiento en línea generalmente es menos eficiente que su contraparte fuera de línea debido a que requiere un mayor número de funciones de

pertenencia para llegar a los mismos resultados, en este sentido el entrenamiento fuera de línea se considera la mejor opción a implementar en nuestro modelo de inferencia hidrológico.

2.3.1.4 Actualización Dinámica del Paso

El paso (*ka*) utilizado en el método del gradiente descendente (expresión A1.9 del Anexo A1) puede influenciar la velocidad de convergencia del método. Si *ka* es pequeño, el método sera cercano al camino del gradiente pero su convergencia sera lenta ya que el gradiente deberá ser calculado muchas veces, por otra parte si el paso es grande la convergencia será en principio más rápida pero el algoritmo será propenso a oscilaciones. Basado en estas observaciones, Roger Jang [1] propone la utilización de dos reglas heurísticas:

1. Si la medición del error conlleva 4 reducciones consecutivas, *ka* debe incrementarse en un 10%.
2. Si la medición del error conlleva 2 combinaciones consecutivas de un incremento y una reducción, *ka* debe decrementarse en un 10%.

El valor inicial de *ka* puede escogerse de manera arbitraria dentro de un rango razonable (comúnmente entre 0 y 1), como valor inicial por defecto se tomó un paso de *ka=0.01*.

Aparte de los parámetros antecedentes y consecuentes, *ka* es el único valor suministrado por el usuario, que se actualiza al finalizar el entrenamiento del modelo, esto con el fin de tomar dicho valor como paso inicial del próximo entrenamiento si éste fuese requerido por el usuario.

2.3.1.5 Importación/Exportación de Parámetros

Se considera la posibilidad guardar en un documento de texto las características del modelo, de manera que sus resultados puedan reproducirse en distintas plataformas computacionales.

La exportación de parámetros se logra al pulsar el botón llamado “exportación”, al iniciar el programa dicho botón se encuentra deshabilitado y solo se activa después de realizar por lo menos una iteración de entrenamiento.

La importación de parámetros se logra al pulsar el botón llamado “importación”, al iniciar el programa dicho botón se encontrará deshabilitado y solo se activa después de cargar los datos de entrenamiento.

Los archivos de texto generados están estructurados por:

- *k* (entradas).
- *L* (número de funciones de pertenencia por entrada).
- *FP* (tipo de función de pertenencia).
- *ka* (paso).
- *abc* (parámetros antecedentes).
- *pqr* (parámetros consecuentes).

Las matrices de parámetros *abc* y *pqr son previamente re*ordenadas para que ocupen una sola columna.

2.3.1.6 Validación del Modelo

Una vez culminado el entrenamiento se puede verificar su correcto

funcionamiento, usando la matriz de validación previamente cargada, las entradas de la matriz serán evaluadas en el modelo con los parámetros antecedentes y consecuentes obtenidos del entrenamiento, si las muestras de validación contienen la salida del sistema se mostrará una gráfica con la salida de la matriz de validación y la estimación del modelo junto con el error cuadrático medio, de no poseer salida solo se mostrará una gráfica con la estimación del modelo.

2.3.1.7 Minimización del Tiempo de Cómputo y Portabilidad del Modelo

En algunos casos puede pasar que el número de entradas, las muestras y/o el número de funciones de pertenencia por entrada sean muy grandes, de modo que el tiempo de cómputo necesario podría ser demasiado alto, un entorno gráfico consumiría recursos del sistema y haría el procesamiento de estos datos más lento, bajo esta consideración se construyó un programa ejecutable sin interfaz gráfica (solo opera en modo consola), a diferencia del entorno gráfico principal, este programa solo fue diseñado para realizar el entrenamiento fuera de línea, las series de entrenamiento y la configuración del modelo son suministradas por archivos de texto, los resultados del mismo son guardados de igual manera, los archivos de texto son compatibles con los usados en el entorno gráfico y pueden ser integrados a este mediante las opciones de importación/exportación.

También se considera viable transmitir las características del modelo junto con las series históricas a una unidad con alta capacidad de cómputo como un Cluster, Grid, etc., donde dicha unidad podría funcionar solo con el programa ejecutable y los archivos antes mencionados.

2.3.2 Herramienta de Pronóstico

Se establecieron dos formas de utilizar la validación del modelo ANFIS de la sección 2.3.1.6, como herramienta de pronóstico:

1. La primera realiza su entrenamiento con las series temporales de las estaciones remotas, considerando los valores conocidos de una serie de precipitación hasta un punto $x=t$ para predecir el valor de un punto futuro $x=t+P$. El método propuesto para este tipo de predicción consiste en crear un conjunto de D puntos de la serie espaciados en Δ partes, es decir, de la forma:

$$[Prec(t-(D-1)\Delta),\ldots Prec(t-\Delta),\ Prec(t),\ Prec(t+P)], \qquad (15)$$

Adicionalmente, en algunas estaciones es posible asignar como entradas otras variables climatológicas (como humedad, presión, temperatura, etc.).

2. La segunda utiliza la validación del modelo ANFIS para establecer un nivel de alerta asociado a la ocurrencia de un alud torrencial, para ello usa las precipitaciones acumuladas a un tiempo medio de vida de 1,5 horas (*Pw1,5)* y 72 horas (*Pw72)*, acorde con el método del comité descrito en el anexo A.4.

Acorde a lo establecido por T. Gascón [23], la selección de eventos extremos de precipitación se realiza analizando series continuas que no tengan más de 24 horas (1 día), que hayan alcanzado un acumulado mayor 80 mm y que dentro de este incluyan por lo menos una lluvia diaria superior o igual a 20 mm.

El modelo ANFIS requiere un entrenamiento con la mayor cantidad de casos posibles, por ello se recomienda el uso series horarias de larga duración, que satisfagan lo establecido en el párrafo anterior.

En la Figura 15 se muestran los niveles de alerta asociados a la posibilidad de ocurrencia de un alud torrencial, fueron definidos mediante líneas paralelas a la línea crítica a intervalos igualmente espaciados, la efectividad de este criterio no ha sido comprobada, de modo que su desempeño debe ser evaluado y, de ser necesario, corregido por un experto.

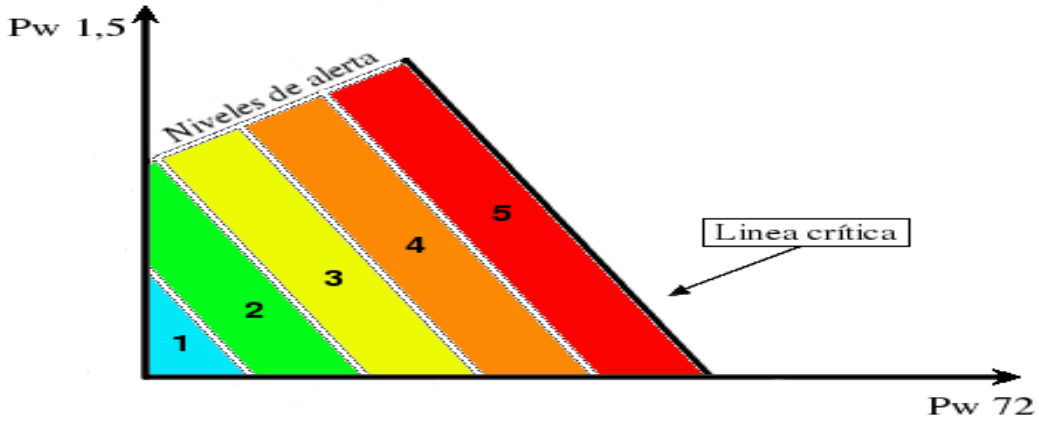


Figura 15: Niveles de alerta propuestos para el pronóstico de aludes torrenciales.

El nivel de alerta es obtenido al considerar la distancia del punto arrojado por *Pw1,5* y *Pw72* respecto a la línea de alerta superior, se considera el uso de 5 niveles, cada uno asociado a un número y un color:

- Nivel 1 (Turquesa): No existe riesgo, la precipitación acumulada es mínima.
- Nivel 2 (Verde): No existe riesgo, la precipitación acumulada es baja.
- Nivel 3 (amarillo): No existe riesgo para la población general, aunque sí para un sector concreto.
- Nivel 4 (naranja): Existe un riesgo importante de ocurrencia de un alud.
- Nivel 5 (rojo): El riesgo de ocurrencia de un alud es extremo.

La alerta puede arrojar valores superiores a 5.9 lo que se interpreta como que ya se ha sobrepasado la línea crítica y un desastre está en proceso, valores de este tipo no tienen efecto sobre el nivel de alerta (se mantiene en Nivel 5), pero son necesarios para el entrenamiento del modelo ANFIS.

Similar al caso anterior, el pronóstico se logra al usar valores conocidos del nivel de alerta y agregando las precipitaciones acumuladas para corto y largo plazo *Pw72(t)* y *Pw1,5(t)* según el formato:

$$[Pw72(t), Pw1,5(t), Alerta(t-(D-1)\Delta), \ldots, Alerta(t), Alerta(t+P)], \quad (16)$$

También es posible evaluar las precipitaciones *Pw72* y *Pw1,5* basándose en las predicciones del primer caso, la alerta arrojada sería el pronóstico del modelo y no sería necesario entrenar con la expresión (16).

2.3.3 Herramienta Web

Consiste de una página web que permitirá visualizar el nivel de alerta presente conjuntamente con el pronóstico del modelo, la página mostrará la evolución de la alerta mediante el caso 2 de la herramienta de pronóstico, adicionalmente la página permitirá acceso a las series históricas usadas tanto en el entrenamiento como en el pronóstico.

La herramienta web no realiza ningún procesamiento, depende enteramente de los datos almacenada en la base de datos, lo que quiere decir que las muestras de entrenamiento, y los resultados del pronóstico deben ser previamente calculados y almacenados.

2.4. Propuesta para la Automatización de las Estaciones Remotas

Considerando que el proyecto PROCEDA ya cuenta con las herramientas necesarias para la adquisición y transmisión de datos, se propone el uso de un administrador regular de procesos que permita la ejecución programada de dichas herramientas. En sistemas Unix esto puede ser logrado con el demonio Cron, en los sistemas Windows se puede lograr los mismos resultados con la función de Tareas Programadas. La ejecución mediante archivos de procesamiento por lotes (tipo Batch) también se considera como una opción viable.

Con la implementación de esta propuesta la función del operador queda limitada al entrenamiento y configuración de las herramientas de pronostico y web, tareas que serán poco exigentes con una base de datos operativa.

CAPÍTULO III

3. SOFTWARE DESARROLLADO

El presente capítulo expone los algoritmos usados en el desarrollo de las herramientas de pronóstico y entorno web, se debe destacar que la mayor parte de las rutinas de lenguaje C corresponden a adaptaciones del código ANFIS de dominio público desarrollados por Roger Jang [20], las rutinas de las funciones de pertenencia (fzfir.sci, bell_1.sci, bell_2.sci y trapeze.sci de los anexos B.1.11, B.1.12, B.1.13 y B.1.14 respectivamente) fueron obtenidas del toolbox FISLAB (Fuzzy Inference Systems toolbox for SCILAB) de G. Ortega [21], que a su vez las adaptó del código de MATLAB (c) creado por Z. Lotfi, el resto de las rutinas aquí mencionadas fueron desarrolladas para satisfacer los objetivos del presente trabajo de grado.

3.1. Herramientas de Entrenamiento y Pronóstico (ANFIS)

Modularmente hablando el modelo ANFIS tiene una estructura jerárquica como la mostrada en la Figura 16, donde un programa puede a su vez llamar a uno o más sub-programas, y devolver los resultados al programa principal.

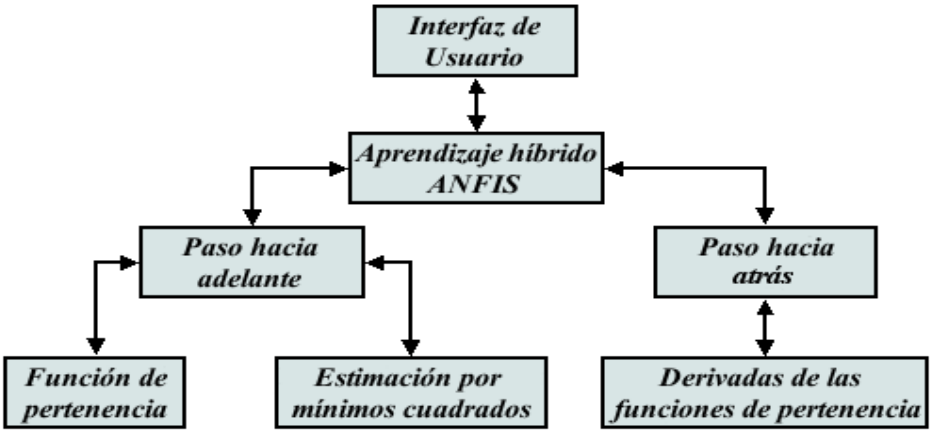


Figura 16: Diagrama de bloques jerárquico del modelo ANFIS

3.1.1 Interfaz de Usuario

En general este bloque consta de dos funciones, la primera es permitir al usuario suministrar las entradas del modelo y la segunda, mostrar los resultados de éste, se han creado dos programas que satisfacen estas condiciones.

3.1.1.1 Interfaz Gráfica

La Figura 17 muestra el entorno gráfico desarrollado en Scilab, el entorno es cargado al ejecutar la rutina “anfislab.sce” (Ver Anexo B.1.1), dicha rutina carga en secuencia: los macros, las librerías dinámicas y los componentes de la interfaz gráfica; la cual fue diseñada para permitir una configuración intuitiva de las características del modelo ANFIS de acuerdo a lo establecido en la sección 2.3.1

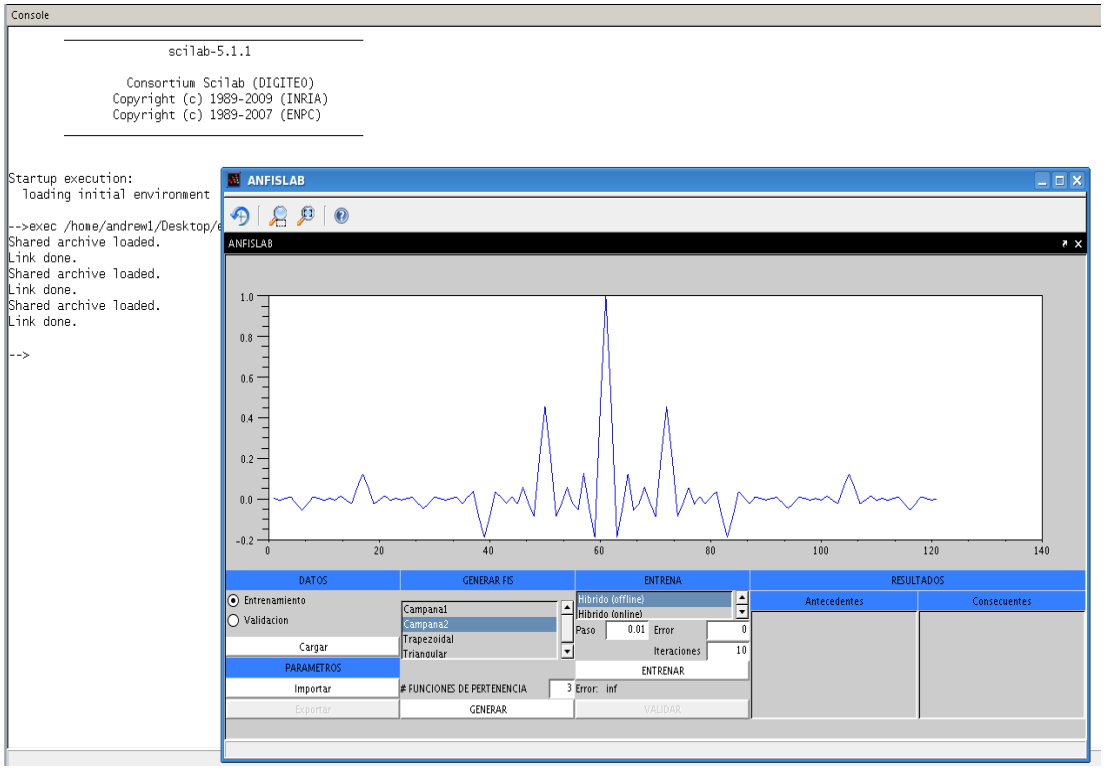


Figura 17: Entorno gráfico desarrollado en Scilab

Este programa suministra las entradas y muestra los resultados del bloque de aprendizaje híbrido (librería dinámica hecha en Lenguaje C) y el macro de validación (valida.sci, ver anexo B.1.8).

3.1.1.1.1 Uso de la Interfaz Gráfica

Considerando la Figura 18, la pestaña “Datos” contiene lo necesario para la carga y selección de datos, en ella pueden observarse dos selectores que nos permiten definir si los datos serán identificados como datos de entrenamiento o datos de validación (sección marcada en rojo), una vez definidos los datos que se quieren cargar se pulsa el botón de “Cargar” (marcada en verde), el cual ejecutará la rutina “cargasel.sci” (anexo B.1.3) abriendo la ventana “uigetfile”, donde se ubicará y seleccionará los datos deseados.

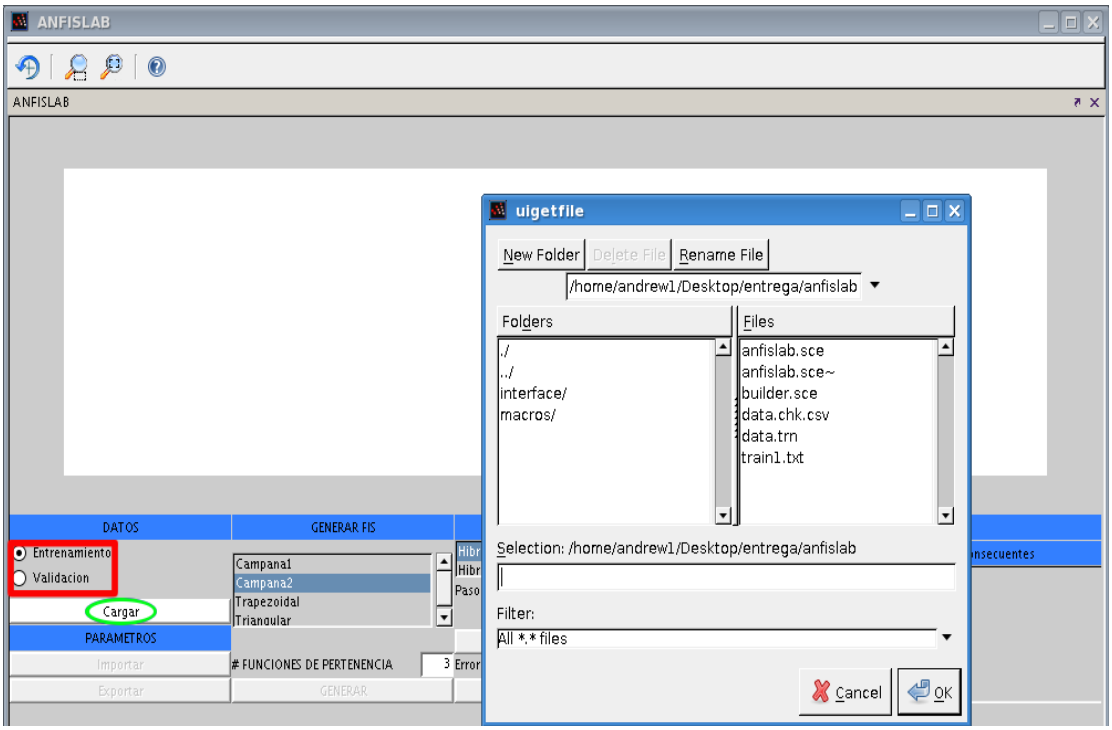


Figura 18: Carga y Selección de datos

En la Figura 19, la pestaña “Generar FIS” permite seleccionar el número y tipo de funciones de pertenencia (marcado en rojo), después de cargar los datos de entrenamiento se activa el botón “Generar” (marcado en verde), al pulsar dicho botón se ejecuta la rutina “genera.sci” (anexo B.1.4), encargada de determinar los parámetros antecedentes iniciales conforme lo establecido en la sección 2.3.1.2, mientras los parámetros consecuentes son puestos a cero (como puede observarse en la pestaña de “Resultados”), para dar una idea de la configuración usada también se grafica la distribución de las funciones en el universo de discurso de la primera entrada.

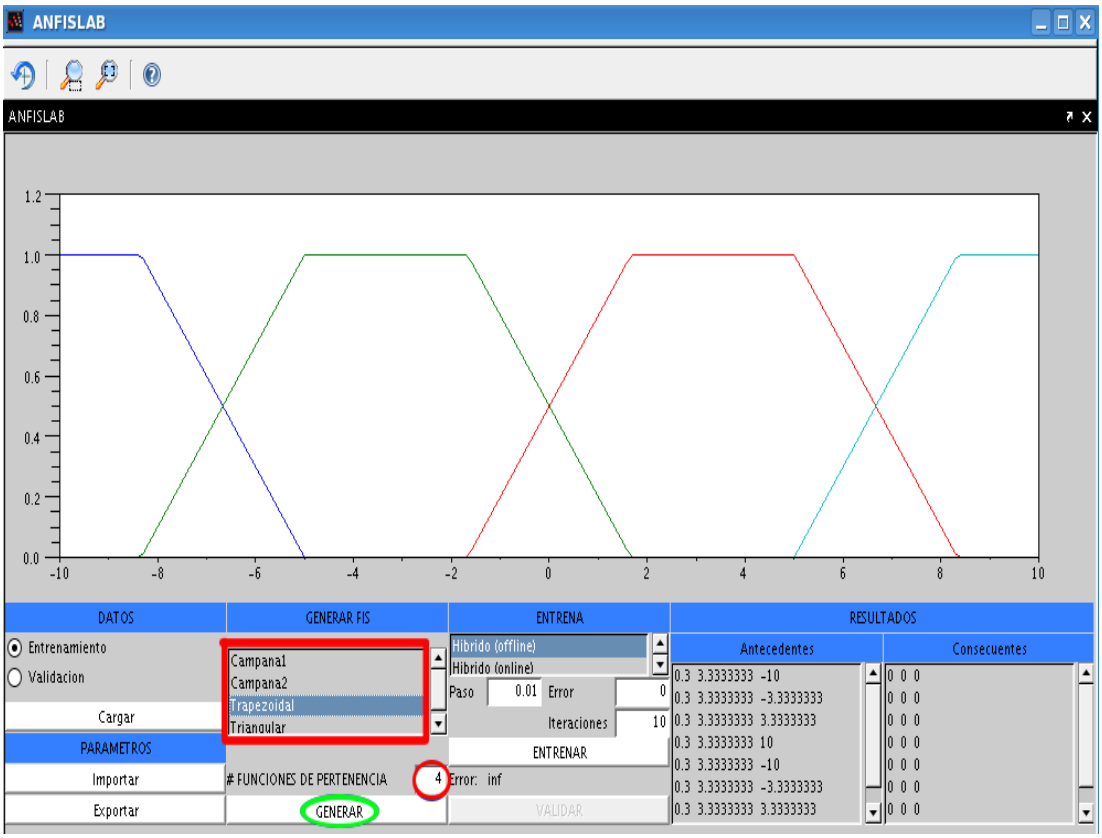


Figura 19: Configuración del modelo y generación de parámetros iniciales

Para la Figura 20, en la pestaña “Entrena” se debe seleccionar el tipo de entrenamiento, pudiendo ser entrenamiento híbrido fuera de línea (offline) ó el

entrenamiento en línea (online) (marcado en rojo) , para ambos tipos debe indicarse un paso inicial, sin embargo los criterios de parada por “Error” e “Iteraciones” (marcados en azul) solo son necesarios si se usa aprendizaje fuera de línea, después de pulsar el botón “Entrenar” (marcado en verde) el modelo ejecutara la subrutina “modela.sci” (anexo B.1.6) la cual llamara una de las librerías dinámicas hechas en C, en base al tipo de entrenamiento seleccionado actualizando los parámetros antecedentes y consecuentes de la pestaña de “Resultados” y mostrando la evolución del error en la ventana gráfica.

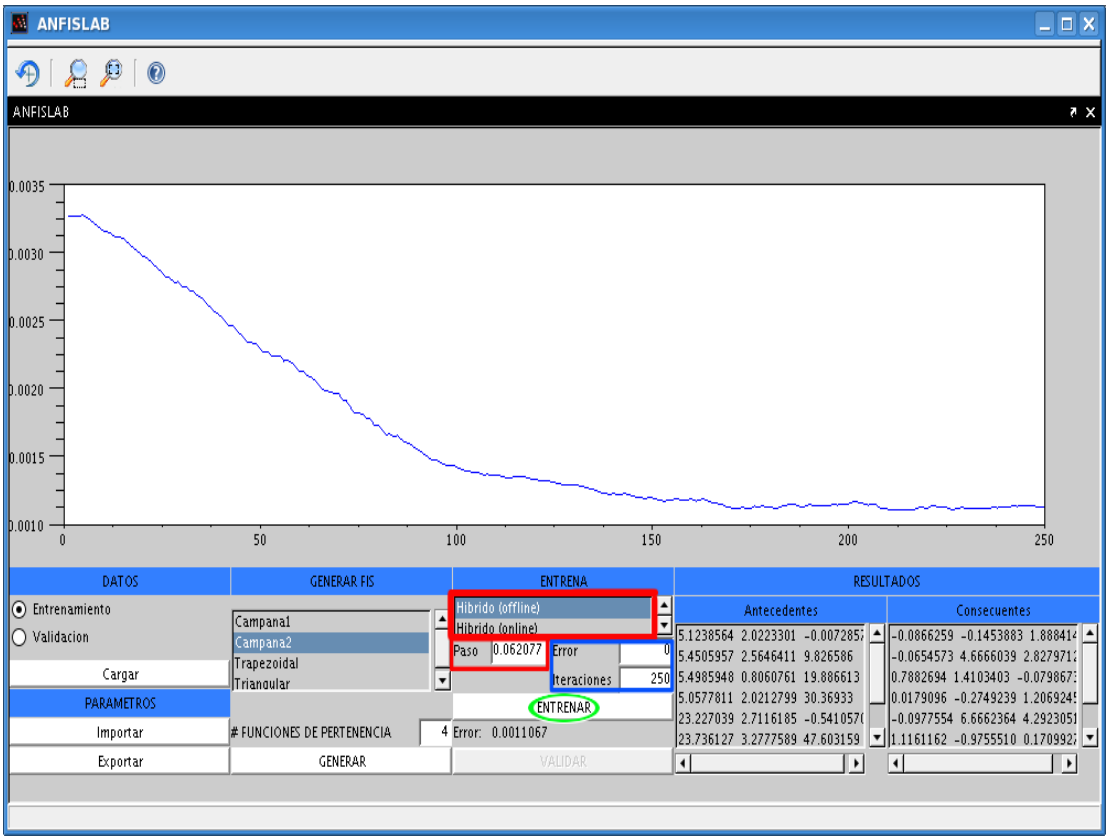


Figura 20: Entrenamiento del modelo ANFIS

Como se indicó en la sección 2.3.1.4, el paso (marcado en rojo) es la única variable aparte de los parámetros que se actualiza al culminar el entrenamiento.

En la pestaña “Parámetros” de la Figura 21, se pueden observar los botones de importación y exportación (marcados en verde), ellos nos permiten adquirir y guardar los parámetros del sistema en archivos de texto, al importar los parámetros se llama la rutina “import.sci” (anexo B.1.9) la cual abre la ventana “uigetfile” donde debe seleccionarse un archivo de texto de la misma manera que se cargaron los datos de entrenamiento y/o validación; para el caso de la exportación, deben haberse obtenido previamente los parámetros antecedentes y consecuentes (bien sea por entrenamiento o generación), ejecutando así la rutina “export.sci” (Anexo B.1.10) que igualmente abre la ventana Uigetfile donde se puede escoger entre sobre-escribir un archivo o guardarlo con un nombre nuevo, para la Figura 21 se exportan los parámetros del modelo llamando al archivo de texto “para.ini” (marcado en rojo).

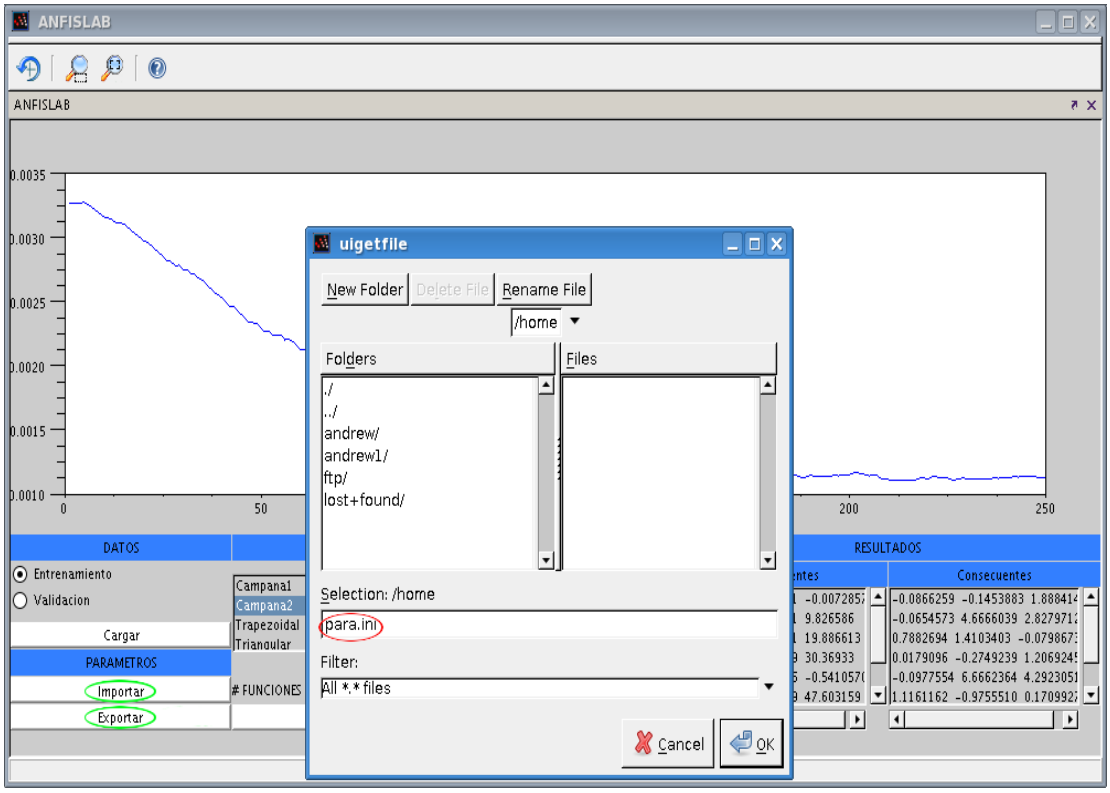


Figura 21: Importación y exportación de parámetros

Al pulsar el botón “Validar”, mostrado en la Figura 22 se ejecuta la rutina “valida.sci” (anexo B.1.8) la cual llama a la librería dinámica correspondiente a la validación, mostrando la estimación del modelo en la ventana gráfica cumpliendo con lo establecido en la sección 2.3.1.6.

Si los datos de validación no contienen la salida real del sistema entonces la ventana gráfica solo mostrara la respuesta del modelo en azul.

Si por otra parte los datos de validación contienen la salida del sistema entonces la ventana gráfica mostrara la salida real en azul y las respuesta del modelo en verde, esto acorde con la configuración por defecto de la función plot() en Scilab.

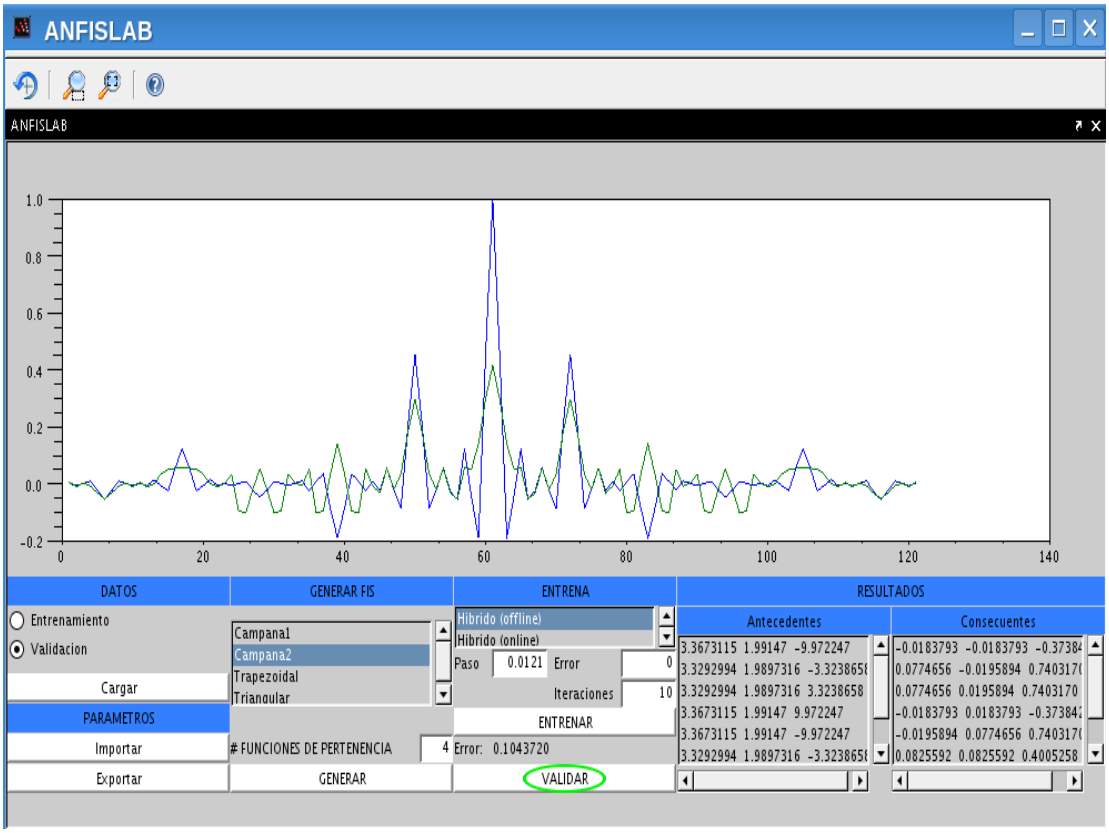


Figura 22: Validación del modelo

Las variables generadas por la interfaz gráfica pueden ser vistas y/o modificadas por la consola de Scilab, en la tabla 4 se resumen los nombres, tipos y descripciones de las variables generadas.

Tabla 4: Variables generadas por la interfaz gráfica

Nombre	Tipo	Descripción
abc	Matriz (*k*L*,3)	Parámetros Antecedentes
evol	Matriz (*pin*,3) o Matriz (*itr*,3)	Evolución del error y el paso a lo largo del entrenamiento, varia según el tipo de entrenamiento
Echk	Matriz (*pchk,k+1*) o Matriz (*pchk,k*)	Muestras de validación, varia si los datos de validación contienen la salida del sistema.
Ein	Matriz (*pin,k+1*)	Muestras de entrenamiento
err	Real positivo	Error cuadrático medio mínimo obtenido
FP	Entero con valores entre 1 y 4	Tipo de función de pertenencia
k	Entero	Entradas de entrenamiento
ka	Real positivo	Paso usado por el método de propagación hacia atrás
L	Entero	Número de funciones de pertenencia por entrada.
O	Vector (*pchk*)	Estimación del modelo
pchk	Entero	Número de muestras de validación
pin	Entero	Número de muestras de entrenamiento
pqr	Matriz (*L^k*,k+1*)	Parámetros consecuentes
Tpo	Entero con valores entre 1 y 2	Tipo de entrenamiento a usar: 1 offline, 2 online
T	Lógico con valores entre 0 y 1	Permite identificar si los datos son de entrenamiento o de validación
Yp	Vector (*pchk*)	Salida real del sistema, es nula si no se incluyo la salida en Echk.

Las rutinas “showres.sci”, “abcfixed.sci”, “grafica.sci”, “bell_1,sci”,

“bell_2.sci”, “trapeze.sci” y fzfir.sci (anexos B.1.7, B.1.2, B.1.11, B.1.12, B.1.13 y B.1.14) son subrutinas auxiliares usadas por las demás rutinas para satisfacer funciones especificas, una explicación general de su funcionamiento puede encontrarse en los comentarios de cada código.

3.1.1.2 Ejecutable de Procesamiento

Es un programa ejecutable desarrollado en ANSI C para satisfacer las consideraciones de la sección 2.3.1.7, su estructura y funcionamiento, están basados en el código original de Roger Jang [20], modificado para generar parte de su configuración de forma automática y ser compatible con los formatos de importación y exportación del entorno gráfico.

Previo a la ejecución del programa es necesario colocar 2 archivos de texto en la misma dirección del ejecutable, el primero deberá nombrarse “data.trn” y contiene la matriz de muestras de entrenamiento del modelo a inferir cumpliendo con lo establecido en la sección 2.2.2, el segundo archivo se debe nombrar “para.ini” y contiene las características del modelo junto con sus parámetros iníciales, este archivo puede generarse mediante el botón de exportación de la interfaz gráfica como se indica en la sección 2.3.1.5, opcionalmente se puede colocar un archivo con las muestras de validación “data.chk”.

Para la ejecución del programa se debe llamar al ejecutable seguido del número de muestras contenidas en los archivos “data” y los criterios de parada, como se indica a continuación:

$ ./anfis <número de muestras de entrenamiento> <número de muestras de validación> <número de iteraciones> <error mínimo requerido>

Solo es obligatorio agregar el número de muestras de entrenamiento ya que los demás valores poseen configuración por defecto (número de muestras de validación = 0, número de iteraciones = 500, error mínimo requerido = 0).

La rutina “main.c” (anexo B.2.24) obtiene las entradas de los archivos mencionados y llama al bloque de aprendizaje híbrido, para finalmente guardar los parámetros actualizados en el archivo “para.fin” y la evolución del entrenamiento en el archivo “evol.txt” (los nombres de estos archivos son definidos en la cabecera “misc_def.h” del anexo B.2.1).

En la Figura 23 se muestra un ejemplo de la forma de llamar la rutina para la ejecución de 10 iteraciones.

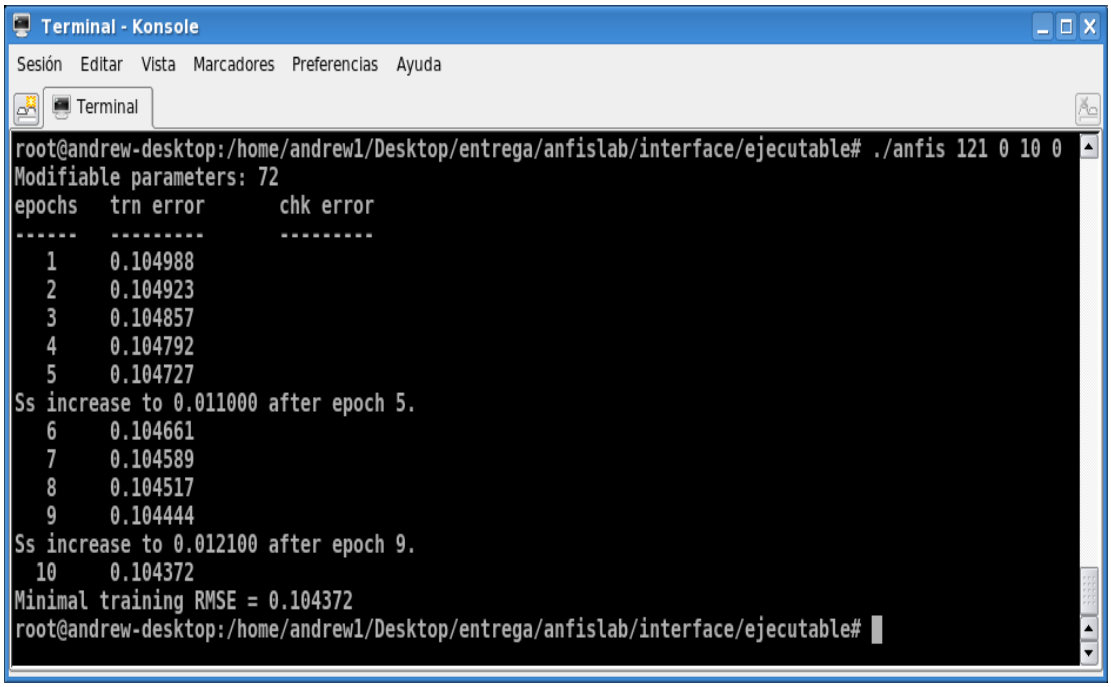


Figura 23: Entrenamiento del modelo ANFIS mediante el ejecutable de procesamiento

donde “121” corresponde al número de muestras contenidas en el archivo “data.trn”, “0” al número de muestras contenidas en el archivo de validación (que en este ejemplo no se usa), “10” corresponde al número de iteraciones y el ultimo “0” indica el error mínimo requerido.

A diferencia de la interfaz gráfica, el ejecutable solo permite el entrenamiento híbrido fuera de línea.

3.1.2 Aprendizaje Híbrido ANFIS

Este bloque engloba las rutinas de aprendizaje en línea y fuera de línea (hyonline.c y hybrid.c del anexo B.2.16 y B.2.17 respectivamente) cuyo diagrama de flujo se muestra en la Figura 24.

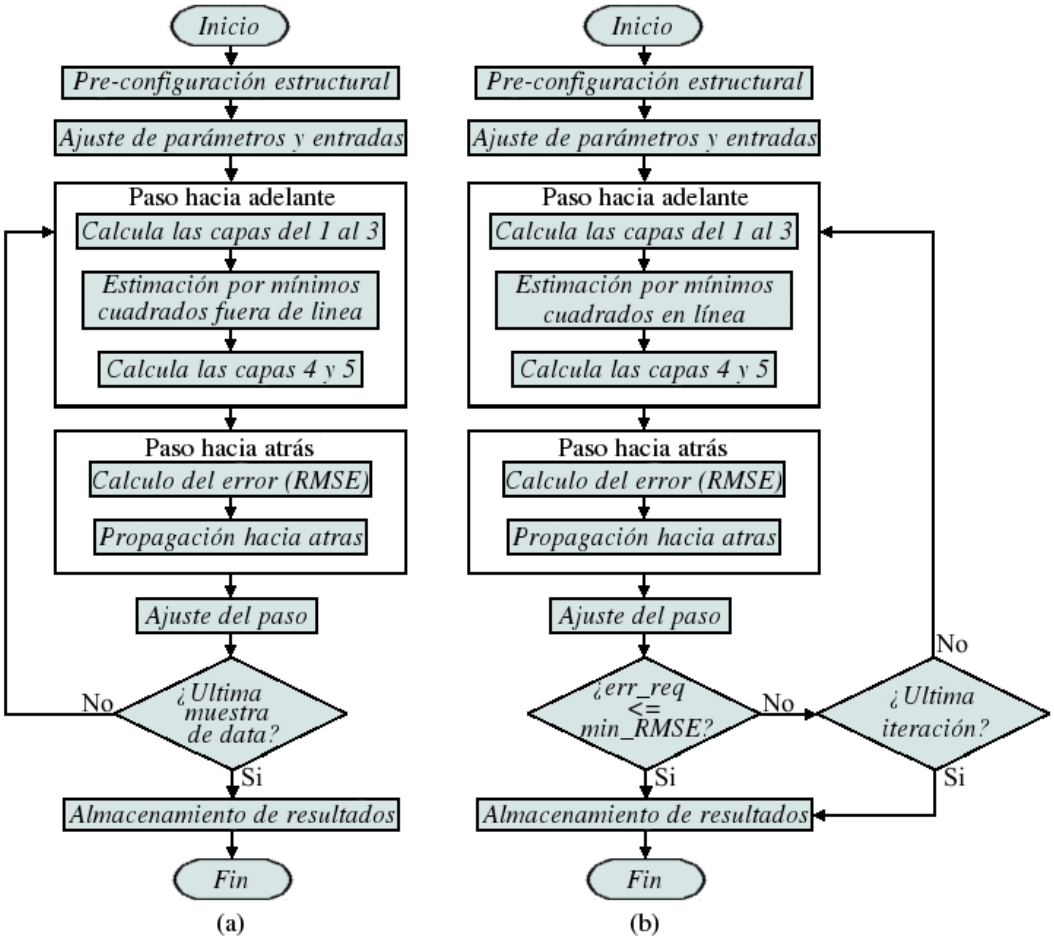


Figura 24: Diagrama de flujo para el modelo de aprendizaje híbrido; (a) en línea, (b) fuera de línea.

El bloque de “pre-configuración estructural” define la configuración de cada nodo en cada capa (rutina datastru.c, anexo B.2.6), el bloque de “ajuste de parámetros y entradas” acondiciona las entradas, salida, parámetros antecedentes (*abc*) y consecuentes (*pqr*) en arreglos y los asocia con en los nodos correspondientes (rutina input.c, anexo B.2.4), el “paso hacia adelante” determina los valores de cada capa con respecto a lo establecido en la sección 1.3.1 (rutina forward.c del anexo B.2.3), como se menciona en el anexo A.2, se usa la ecuación A.2.4 para la “estimación por mínimos cuadrados” fuera de línea y A.2.5. para el caso en línea (rutinas LSE_kalman.c y kalman.c de los anexos B.2.9 y B.2.8 respectivamente), el “paso hacia atrás” (rutina backward.c, anexo B.2.3) calcula el error cuadrático medio (*RMSE* por sus siglas en ingles) y propaga el error de acuerdo a lo establecido en el anexo A.1, la “actualización del paso” registra las variables de error modificando el paso según las reglas heurísticas definidas en 2.3.4. (rutina trn_err.c, anexo B.2.13), finalmente el bloque de “almacenamiento de resultados” crea un arreglo matricial con el error y el paso asociado a cada iteración.

Retomando la Figura 24, podemos observar que las iteraciones del programa son diferentes según el tipo de entrenamiento usado, el caso fuera de línea posee dos criterios de parada, el primero un error requerido (*err_req*) y el segundo un número máximo de iteraciones, ambos suministrados por el usuario según lo establecido en la sección 2.3.1.3, por otro lado, el criterio de parada del modelo en línea solo depende del número de muestras de entrenamiento suministradas.

3.1.3 Paso Hacia Adelante

Constituido por la rutina “forward.c” (anexo B.2.3), con algunas llamadas a las rutinas “kalman.c” o “LSE_kalman.c” (anexos B.2.9 y B.2.8), fue modificado del código ANFIS de dominio público [20] para soportar las funciones de pertenencia discutidas en la sección 2.3.1.2, cumpliendo con lo descrito en la Figura 25.

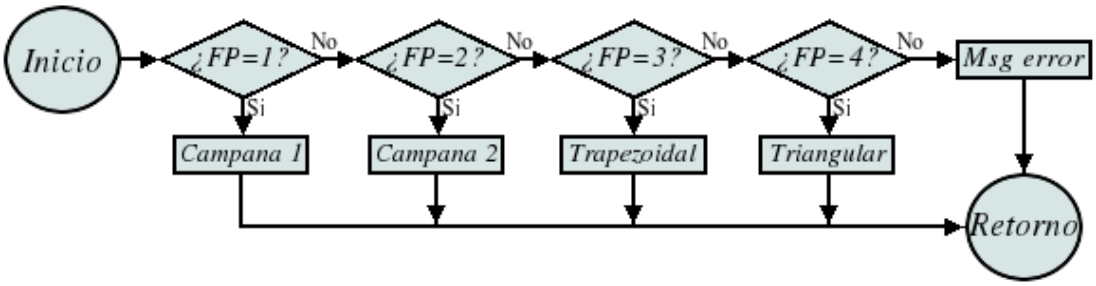


Figura 25: Diagrama de flujo de la función funsel() de la rutina forward.c (Anexo B.2.3).

La función funsel() de la rutina "forward.c", selecciona la función de pertenencia a usar en la capa 1, las demás capas son calculadas de acuerdo con lo descrito en la sección 1.3.1.

Previo a la "estimación por mínimos cuadrados", se necesita construir la matriz "*A*", que servirá como entrada para el sistema de ecuaciones $AX = B$ (ecuación A2.3 del Anexo A.2), dicha matriz es obtenida al guardar los valores de la capa 3 asociados a cada muestra.

Ya obtenida "*A*", los parámetros consecuentes son calculados mediante la estimación por mínimos cuadrados fuera de línea o en línea, expresiones A.2.4 y A.2.5 según sea el caso.

3.1.4 Paso Hacia Atrás

Compuesto por las rutinas "backward.c" y "de_dp.c" (anexos B.2.2 y B.2.7), calcula la propagación del error cuadrático medio desde la capa 5 hasta la capa 1 mediante el método del gradiente descendente descrito en el anexo A.1, al igual que en el caso anterior, estas rutinas fueron modificadas del código ANFIS de dominio público [20] para calcular las derivadas parciales de la primera capa en base a su función de pertenencia.

La función dmf_fp() de la rutina “de_dp.c” posee un diagrama de flujo idéntico al mostrado en la Figura 25, con la única diferencia de que en lugar de evaluar las funciones de pertenencia, evalúa las derivadas parciales de éstas. Con el fin de reducir los tiempos de ejecución, el paso hacia atrás utiliza la alternativa al cálculo de las derivadas parciales descrita en el Anexo A.3

3.1.5 Evaluación del Modelo

La rutina “checka.c” (anexo B.2.15) realiza llamados a los bloques de “pre-configuración estructural”**,** “ajuste de parámetros” y “entradas” para finalmente ejecutar un “paso hacia adelante” y obtener la salida estimada por el modelo.

A diferencia del entrenamiento, la evaluación no requiere la ejecución de un paso hacia atrás, ni precisa el cálculo los parámetros antecedentes, cumpliendo así con lo establecido en 2.3.1.6 la salida en las muestras de validación es opcional.

3.2. Interfaz Scilab/C, C/Scilab

Cuando Scilab llama a una librería externa, manda y recibe las variables en el formato usado por FORTRAN (por punteros), para que este formato sea correctamente interpretado por un programa en lenguaje C, es necesario crear un programa (rutina) que haga de intérprete, bajo esta consideración Scilab dispone de una herramienta llamada INTERSCI [24].

Las rutinas “check.c”, “hyon.c” y “hysci.c” (anexos B.2.21, B.2.22 y B.2.23) fueron generadas por la herramienta INTERSCI, mientras las rutinas “ccheck.c”, “chyontrain.c” y “chytrain.c” (anexos B.2.18, B.2.19 y B.2.20) cambian los formatos para adaptarse a los requerimientos de las otras rutinas.

3.3. Compilando Rutinas de C

Previendo futuras modificaciones al programa se crearon rutinas para la construcción del ejecutable de procesamiento y las librerías dinámicas, el ejecutable puede ser nuevamente compilado al llamar la rutina “Makefile” del anexo B.2.2.5, por otro lado Scilab y GCC tienen una excelente interacción de modo que se pueden compilar librerías en C directamente de la consola de Scilab, para ello se ejecutan en su consola (exec <archivo.sce>) las rutinas “hysci_builder.sce”, “hyon_builder.sce” y “checka_builder.sce” de los anexos B.1.16, B.1.17 y B.1.18.

3.4. Herramienta Web

La página principal mostrada en la Figura 26 muestra las alertas (presente y pronosticada), conjuntamente con las gráficas, y los enlaces de descarga e información, cumpliendo con lo establecido en la sección 2.3.3.

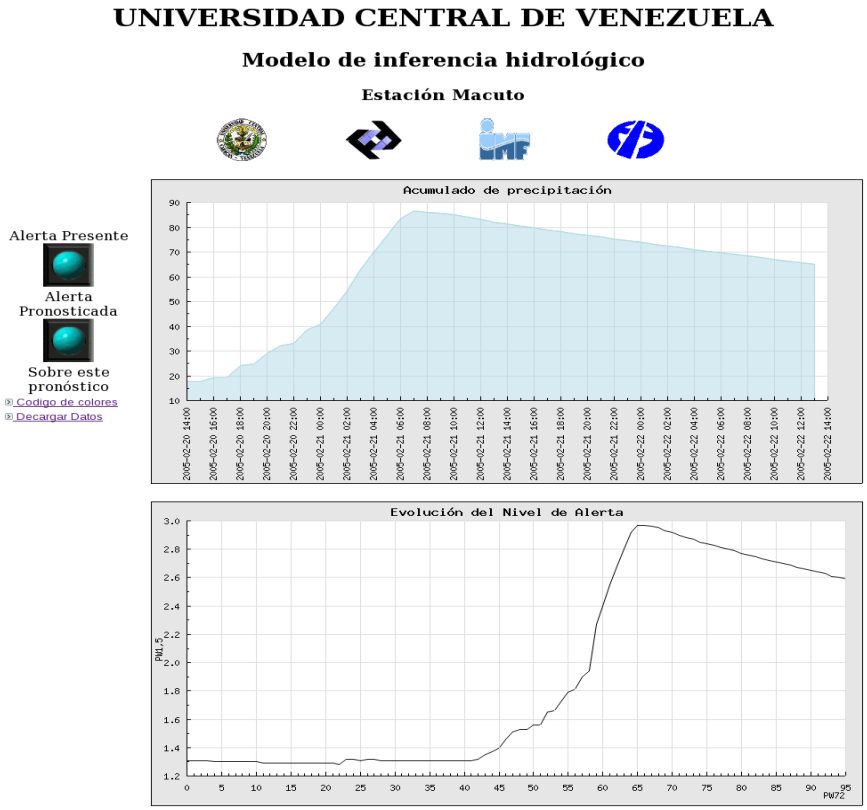


Figura 26: Entorno web

Desarrollada en PHP y HTML, está principalmente compuesta por la rutina “pa.php” (anexo B.3.1).

Tanto las gráficas como las alertas son dinámicas, ya que son generadas con base a los datos de entrenamiento y los resultados de la herramienta de pronóstico.

Para la obtención de los gráficos se uso la librería jpgraph [25], la primera gráfica es generada por la rutina “acumulado.php” (anexo B.3.2) y muestra la precipitación acumulada para un tiempo de vida medio de 72 horas (*Pw72*), la segunda gráfica es generada por la rutina “grafico_linea.php” (anexo B.3.3) y muestra la evolución de la alerta con su respectivo pronóstico, por defecto la gráfica 1 (gráfica superior de la Figura 26) considera las ultimas 48 muestras, mientras la gráfica 2 considera las ultimas 96, el número de datos considerado por cada gráfica puede ser cambiado al modificar la variable $lineas de cada rutina.

Al pulsar el enlace “Descarga de datos” se llama a la rutina “descarga.php” (anexo B.3.4) la cual fuerza la descarga de los datos usados para la generación de las gráficas, esta rutina puede ser modificada para permitir la descarga de otros datos de interés.

Los niveles de alerta y las gráficas dependen de la información contenida en la base de datos, por lo que la información debe ser actualizada regularmente para convertirse en un medio de divulgación efectivo, por defecto le herramienta web es actualizada cada 500 segundos, este tiempo debe adaptare al tiempo de actualización de la base se datos.

Las rutinas mostradas en el anexo B.3 están referidas a un archivo de texto,

por lo que deben referirse a la base de datos cuando esta se encuentre operativa.

Finalmente el enlace “Código de colores” muestra la página de información “colores.html” (anexo B.3.5), la cual resume los colores asociados a los niveles de alerta.

La Figura 27 muestra la pagina de información.

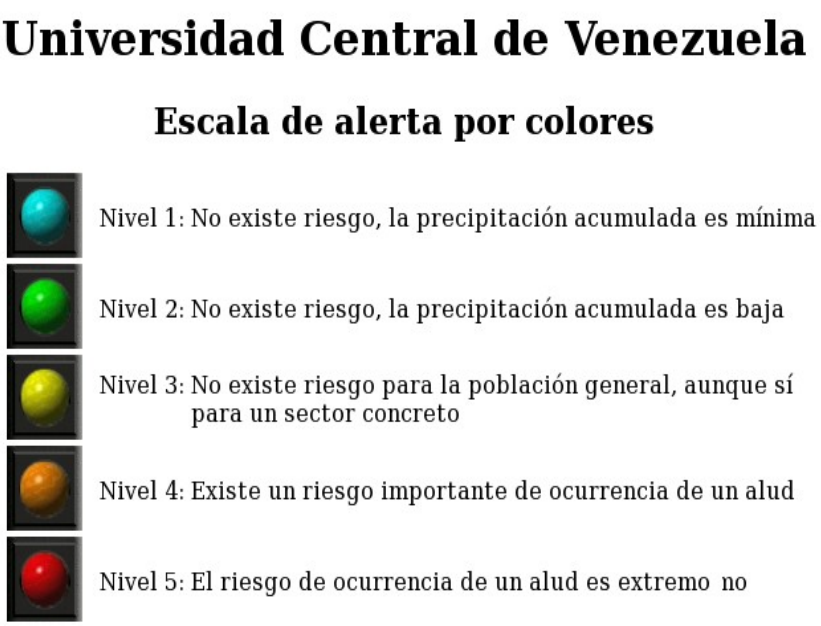


Figura 27: Pagina de información

CAPÍTULO IV

4. RESULTADOS Y ANÁLISIS

Este capítulo presenta los resultados y análisis de las simulaciones usadas para validar la efectividad del modelo ANFIS, se realizaron 4 simulaciones, la primera, tomada de la literatura, estima la salida de una función altamente no lineal, en la segunda usamos ANFIS para modelar el comportamiento de la serie de precipitación de una estación pluviométrica, en la tercera utilizamos ANFIS como una herramienta predictiva asociada a la precipitación de una estación meteorológica, finalmente la cuarta simulación utiliza el modelo ANFIS como herramienta de pronóstico para estimar el nivel de alerta asociado a la ocurrencia de un alud torrencial.

4.1. Consideraciones Prácticas

En las simulaciones presentadas el número de funciones de pertenencia usado es escogido de forma empírica, bien sea examinando la relación entradas-salida de las muestras de datos o por ensayo y error.

Por limitaciones computacionales la estructura de los datos (número de muestras, entradas, etc.) y el número de funciones de pertenencia fueron escogidos de modo que la suma de los parámetros antecedentes y consecuentes no superara los 600 parámetros modificables.

Tras el ajuste inicial de las funciones de pertenencia, los parámetros antecedentes iniciales son configurados de modo que las funciones estén igualmente

espaciadas en el universo de discurso, cumpliendo con los criterios de generación de la sección 2.3.2.

A menos que se indique lo contrario, el paso inicial *ka* será de 0.01 actualizándose acorde con las reglas heurísticas establecidas en la sección 2.3.4.

En todas las simulaciones la salida real del sistema se mostrara en azul mientras la respuestas del modelo se mostrara en rojo.

4.2. Resultados

4.2.1 Simulación 1- Modelo de una Función No Lineal

Se pretende estimar la salida "z" de una función seno cardinal no lineal

$$z = sinc(x, y) = \frac{sen(x)}{x} * \frac{sen(y)}{y} \tag{17}$$

Esta función es la misma usada como ejemplo por Roger Jang [1], se tomó como referencia para asegurar que las modificaciones hechas al código de dominio público [20] no afectaron su funcionamiento.

Se obtuvieron 121 muestras correspondientes a la evaluación de la ecuación "17" asumiendo las entradas "*x*" y "*y*" en un rango entre [-10,10], se utilizó un modelo ANFIS de 16 reglas con 4 funciones de pertenencia por entrada, para cada una de las funciones de pertenencia disponibles (campana 1, campana 2, trapezoidal y triangular).

La Figura 28 muestra las curvas de la evolución del error cuadrático medio (RMSE) asociadas a las 4 funciones de pertenencia, cada curva es el promedio de 10

corridas donde cada corrida inicia con un paso diferente (0.01, 0.02, ... 0.10).

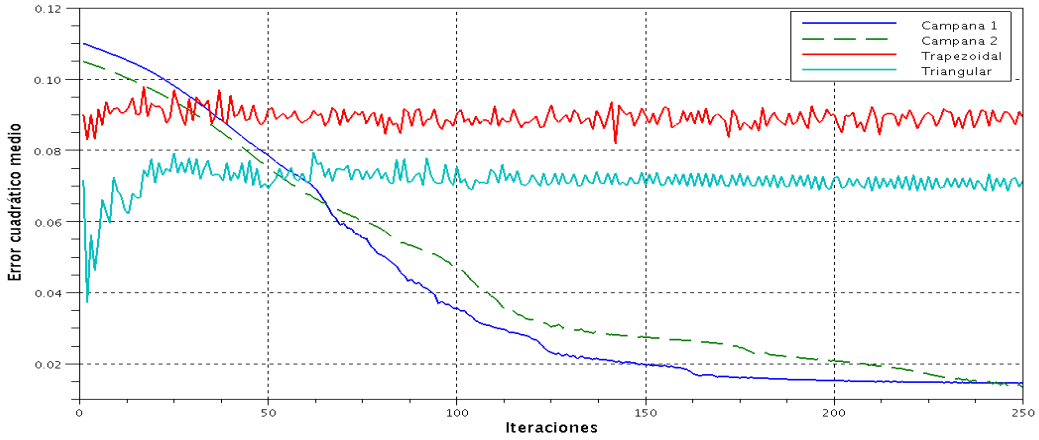


Figura 28: Curvas del error cuadrático medio asociadas al modelo ANFIS para los distintos tipos de funciones de pertenencia.

La Figura 29 muestra la evaluación del modelo para las distintas funciones de pertenencia y la tabla 5 resume el error cuadrático medio alcanzado por cada función.

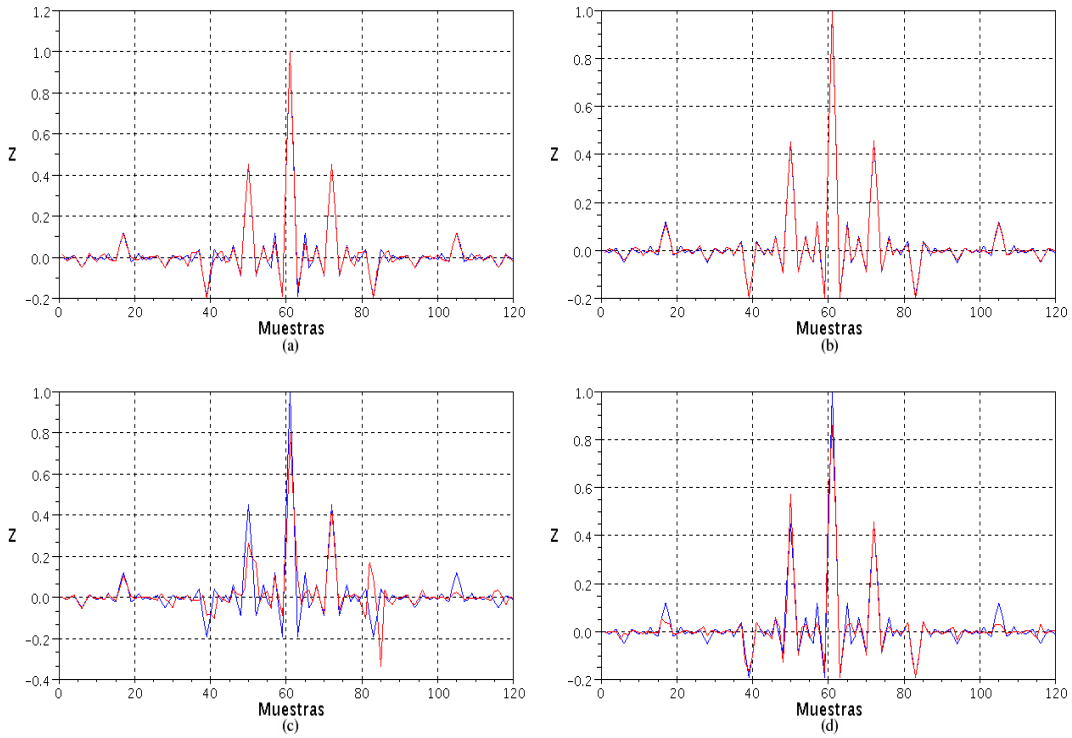


Figura 29: Respuesta del modelo (a) Campana 1; (b) Campana 2; (c) Trapezoidal; (d) Triangular.

Tabla 5: Errores mínimos asociados al entrenamiento del modelo

	Error cuadrático medio
Campana 1	0,197363
Campana 2	0,192636
Trapezoidal	0,847263
Triangular	0,398262

Lo expuesto en las figuras 28 y 29 reafirma la efectividad del modelo de inferencia ANFIS para funciones no lineales, las funciones en campana mostraron ser las más efectivas para esta simulación ya que el error disminuyó a lo largo de todo el entrenamiento, por otro lado las funciones trapezoidal y triangular presentaron oscilaciones indicando la presencia de mínimos locales, a pesar de ello los errores fueron lo bastantes bajos como para que las figuras 29 (c) y (d) mostraran gran similitud con la salida real.

4.2.2 Simulación 2- Modelando la Serie de Precipitación de una Estación Pluviométrica.

Aquí se usa el modelo ANFIS para evaluar la respuesta del modelo ante una serie de precipitación, la serie en cuestión corresponde a la estación pluviométrica de Macuto para los días del 5 al 10 de febrero de 2005, esta estación posee los registros más largos del proyecto PROCEDA, pero solo guarda registros de precipitación, la serie fue previamente ajustada al formato de la expresión (15) (sección 2.3.2), para $P=\Delta=1$ y $D=1$, como se muestra a continuación:

$$[\,y(t-1),\;\; y(t)\,], \tag{18}$$

Se obtuvieron 251 muestras, con las cuales se entrenó el modelo ANFIS para funciones en campana 2 con variaciones del número de funciones de pertenencia por entrada "*L*".

La Figura 30 muestra la evolución del error para 500 iteraciones considerando L=2, 3, 5, 7, 15 y 21, en la la tabla 6 se resumen el menor error de cada caso.

Tabla 6: Error cuadrático medio en base al número de funciones de pertenencia por entrada

Funciones de pertenencia por entrada "L"	Error cuadrático medio
L=2	2,968359
L=3	2,799773
L=5	2,456980
L=7	2,279582
L=15	1,695443
L=21	1,518849

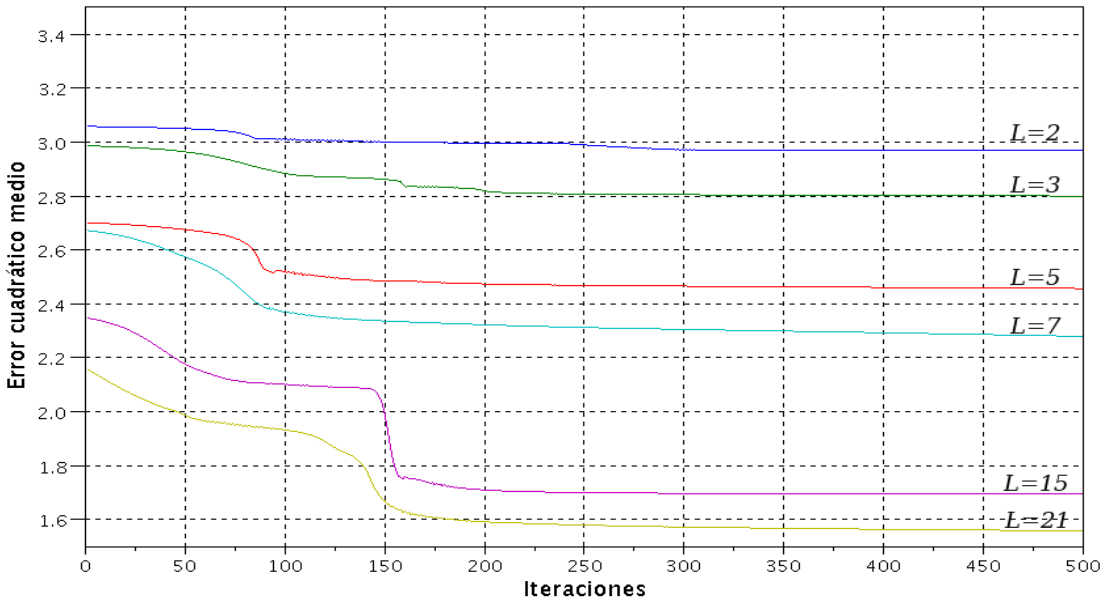


Figura 30: Curvas del error cuadrático medio Simulación 2

A fin de comparar los resultados también se ajustó mediante mínimos cuadrados un modelo autorregresivo lineal de orden 1 (AR1), este modelo es

comúnmente usado en hidrología. Una descripción general de él puede encontrarse en el anexo A.5.1.

El modelo AR1 obtenido a partir de los pares de muestra es:

$$\hat{y}(t) = 0{,}501 + 0{,}7015\ y(t-1) \tag{19}$$

Al evaluar el modelo AR1 de la ecuación (19) con los datos de entrenamiento se obtuvo un error cuadrático medio de 3.283736, error bastante mayor a los obtenidos por el modelo ANFIS en la Figura 30 y tabla 6.

En la Figura 31 podemos apreciar la salida real "*y(t)*" en 31(a) y en 31(b)-31(d) las respuesta del modelo AR1 y del modelo ANFIS para 2 y 21 funciones de pertenencia por entrada.

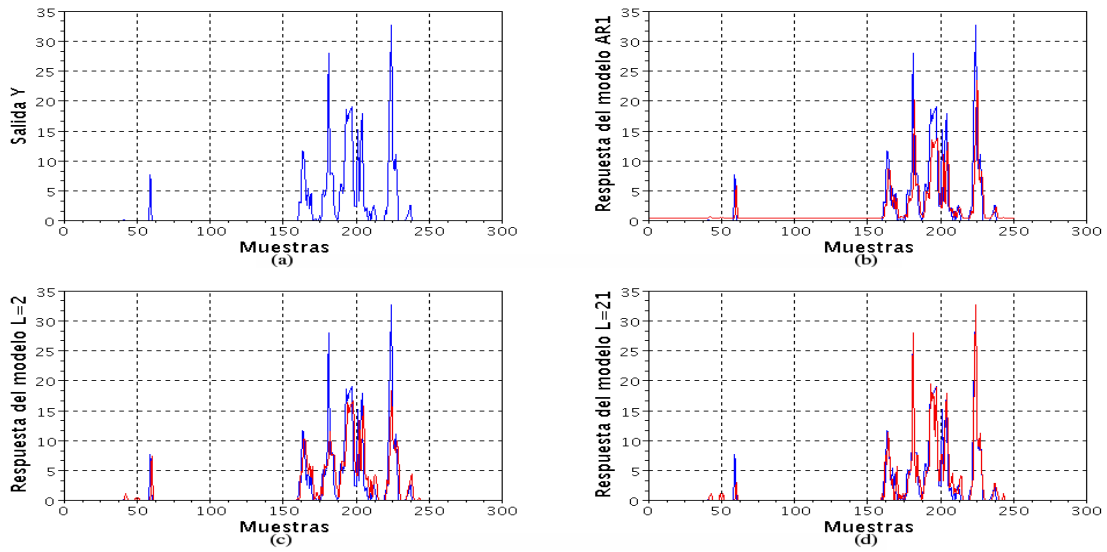


Figura 31: Respuestas de la simulación 2: (a) Salida Y, (b) Modelo AR1, (c) Modelo ANFIS para L=2, (d) Modelo ANFIS para L=21

El modelo AR1 presenta valores constantes en los periodos de inactividad, es decir cuando la salida real es cero, y aunque siguió satisfactoriamente a la salida real, se pudo observar que falla en alcanzar las magnitudes de ésta, por su parte el modelo ANFIS arroja una mejor respuesta a las magnitudes, con errores mucho menores durante los periodos de inactividad.

En la Figura 32 se aprecia el error cuadrático de cada modelo para las muestras de validación, aunque los errores del modelo ANFIS son claramente apreciables podemos ver que se encuentran a una escala mucho menor a los erroes obtenidos por el modelo AR1.

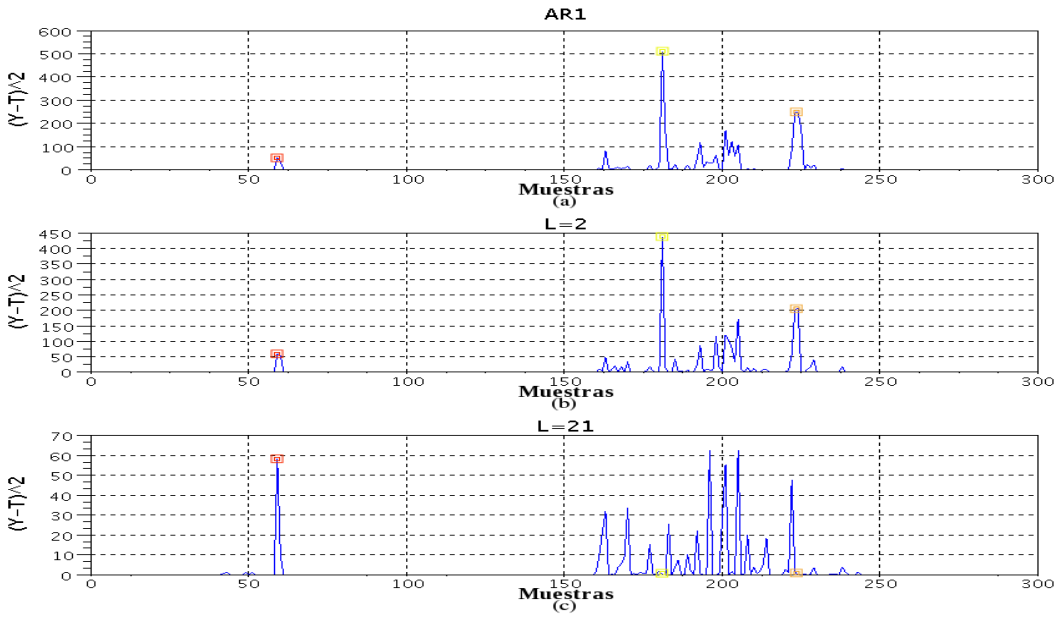


Figura 32: Error cuadrático simulación 2: (a) Modelo AR1, (b) Modelo ANFIS para L=2, (c) Modelo ANFIS para L=21

4.2.3 Simulación 3- Pronóstico de la Precipitación Asociada a una Estación Meteorológica

Aquí usamos el modelo ANFIS para predecir el comportamiento de la precipitación, se tomaron las muestras horarias, comprendidas entre las 4 pm del 06-

01-2005 y las 2 pm del 16-02-2005, del registro histórico de la cuenca de San José de Galipán, la cual es una estación meteorológica que además de precipitación contiene registros de humedad, temperatura, presión atmosférica y radiación solar, por limitaciones de cómputo solo se consideraron 6 entradas y 2 funciones de pertenencia por entrada, se ajustaron las muestras a la expresión (15) (sección 2.3.2), para $P=\Delta=1$ y $D=4$, con humedad (*Hu*) y presión (*Ps)* en tiempo presente, de la forma:

$$[Hu(t), Ps(t), Prec.(t-3), Prec.(t-2), Prec.(t-1), Prec.(t), Prec.(t+1)], \quad (20)$$

De las 978 muestras obtenidas al evaluar las variables en la expresión (20), las muestras pares fueron asignadas como datos de entrenamiento mientras las impares como datos de validación, se usaron 2 funciones de pertenencia por entrada y 64 reglas, la Figura 33 muestra la evolución del error para 500 iteraciones con los datos de entrenamiento.

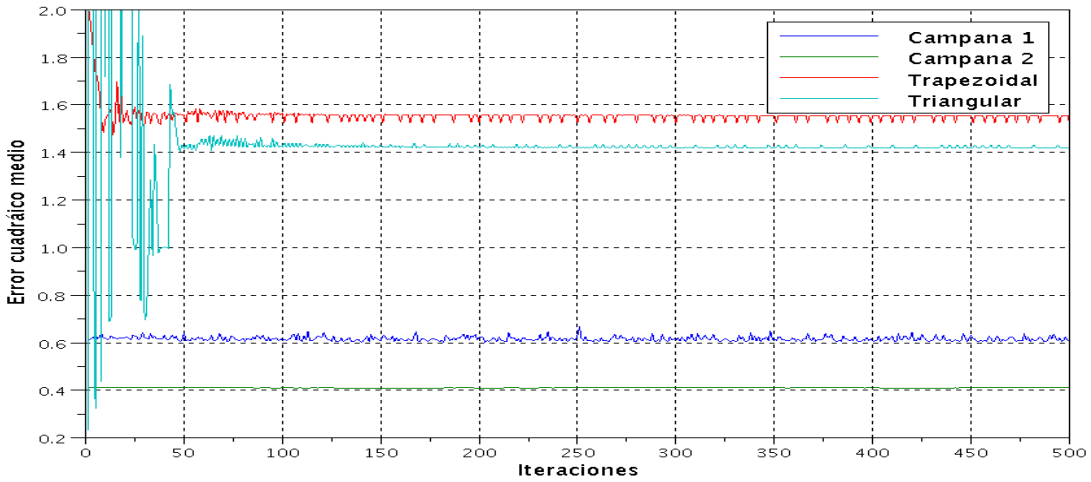


Figura 33: Curvas del error cuadrático medio Simulación 3

El error de la función triangular osciló entre 0,233138 y 8703,020 para sus primeras iteraciones, por tal motivo su escala fue reducida para facilitar su comparación con las demás funciones de pertenencia. En la tabla 7 se resume la evolución del error observada en la Figura 33.

De la Figura 33 también puede observarse como cada función mostró una tendencia a quedar atrapada en un mínimo local, sin embargo la función triangular presento en su primera iteración un error muy bajo que no fue superado a lo largo del entrenamiento y por ese motivo se muestra constante en la tabla 7.

Tabla 7: Menor error de entrenamiento según el número de iteraciones

	1 iteración	100 iteraciones	500 iteraciones
Campana 1	0,617833	0,605312	0,602281
Campana 2	0,411792	0,410171	0,409706
Trapezoidal	2,027753	1,515592	1,473849
Triangular	0,233138	0,233138	0,233138

En la Figura 34 se muestra la respuesta del modelo a los datos de entrenamiento.

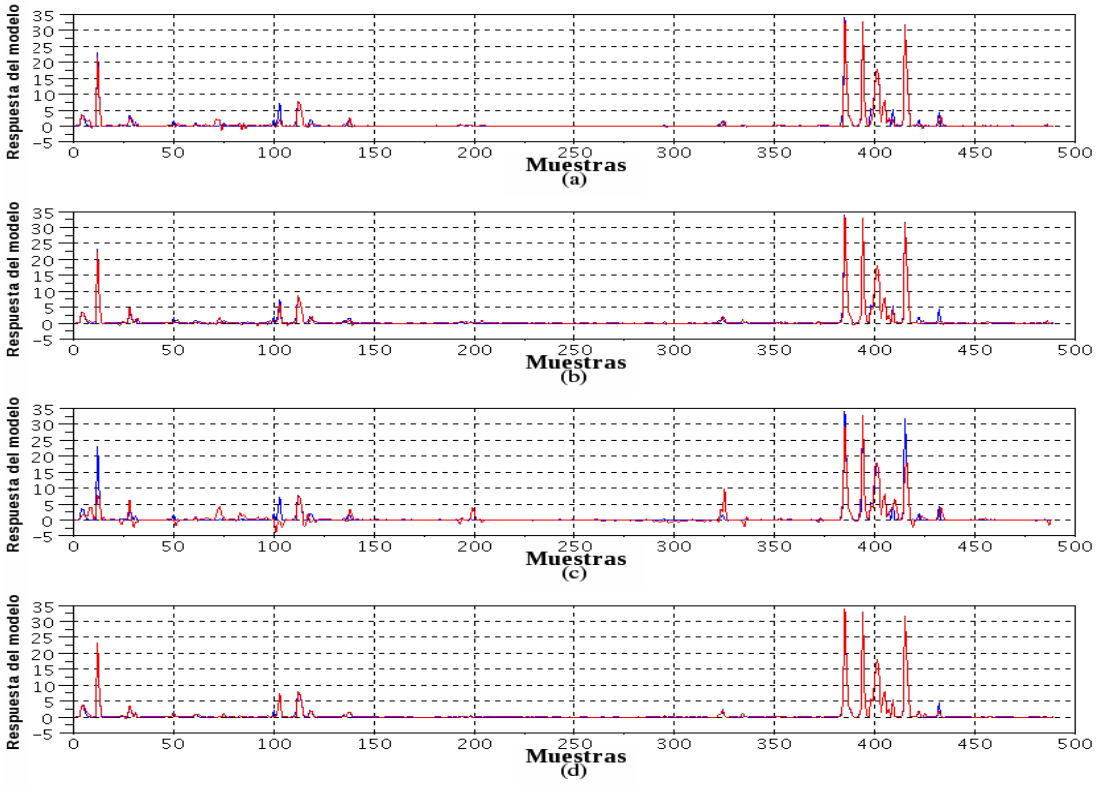


Figura 34: Respuesta del modelo para los datos de entrenamiento :(a) Campana 1; (b) Campana 2 (c) Trapezoidal; (d) Triangular

Se puede apreciar una gran semejanza con la salida real del sistema indiferentemente de la función de pertenencia usada.

La Figura 35 muestra la respuesta a los datos de validación, si bien el sistema sigue el comportamiento de la salida real, podemos apreciar que falla en alcanzar las magnitudes de ésta, y por tanto arroja errores mucho mayores respecto a los datos de entrenamiento.

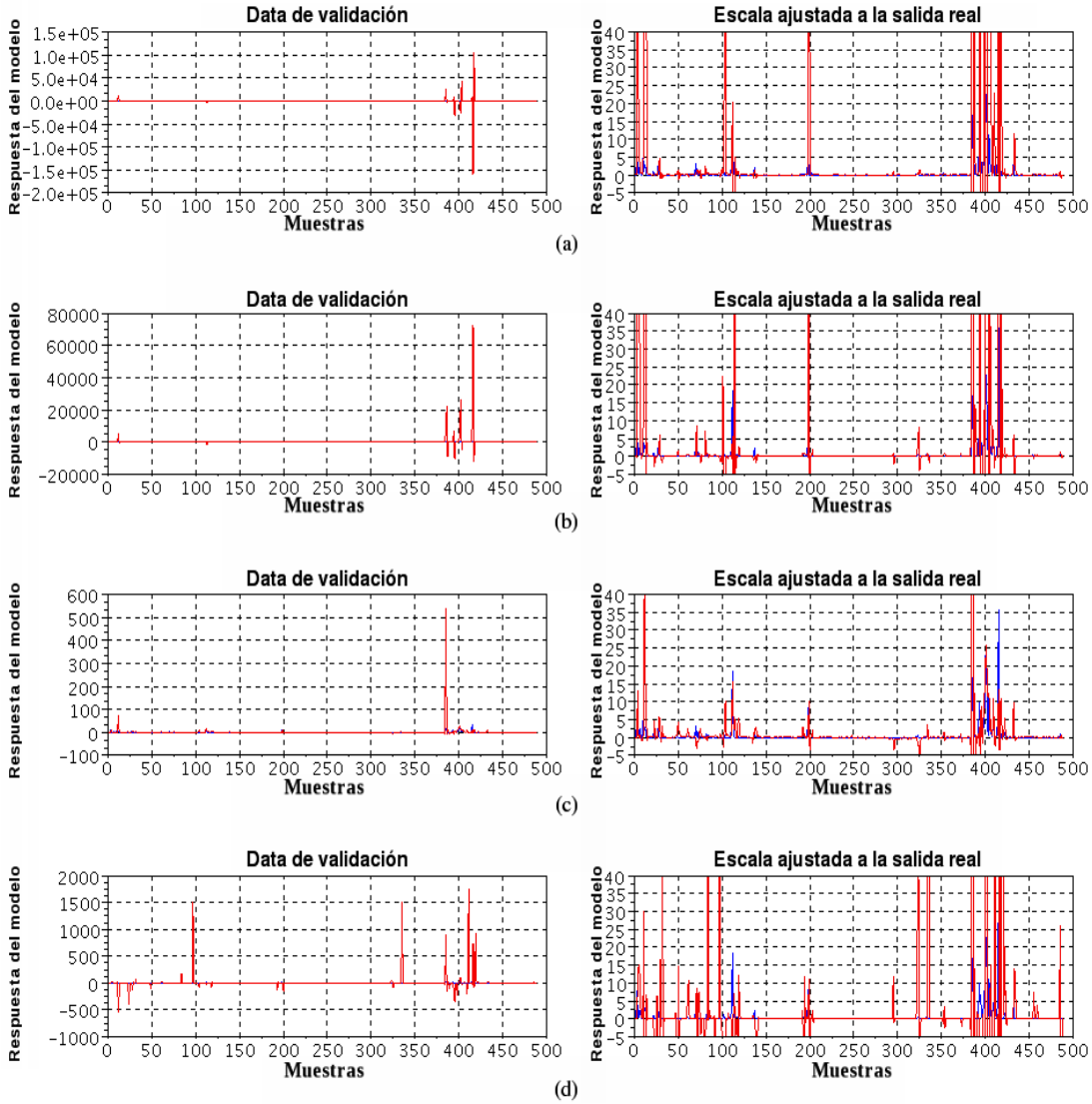


Figura 35: Respuesta del modelo para los datos de validación :(a) Campana 1; (b) Campana 2 (c) Trapezoidal; (d) Triangular

Los errores obtenidos durante la validación se resumen en la tabla 8, conforme el entrenamiento avanza, el error de validación mostró ser demasiado elevado en las funciones en campana y aunque estas mostraron mejoría conforme aumentaban las iteraciones, su error siguió siendo elevado en comparación con las funciones trapezoidal y triangular.

Tabla 8: Menor error de validación según el número de iteraciones

	1 iteración	100 iteraciones	500 iteraciones
Campana 1	10234,897764	9789,306679	9426,809967
Campana 2	3798,657754	3856,812898	3841,358121
Trapezoidal	4,126457	12,198181	23,898581
Triangular	147,953692	147,953692	147,953692

Las respuestas del modelo a los datos de validación arrojaron errores significativamente mayores a los obtenidos en el entrenamiento, también fue notable como la función trapezoidal que tenía el mayor error durante el entrenamiento arrojo los mejores resultados en los datos de validación, se realizaron otras simulaciones para variaciones de la expresión (20), con $D=2$, $D=3$, considerando valores pasados y presentes de otras variables, todas con resultados similares, por esta razón se concluye que la información contenida en los datos de entrenamiento no es suficiente para reproducir las características caóticas de la serie de precipitación usada.

Mediante una observación cuidadosa se pudo determinar, que la mejor respuesta del modelo a los datos de validación, era obtenida en la tercera iteración del entrenamiento con la función trapezoidal, en la Figura 36 se puede apreciar una comparación de la respuesta del modelo con la salida real.

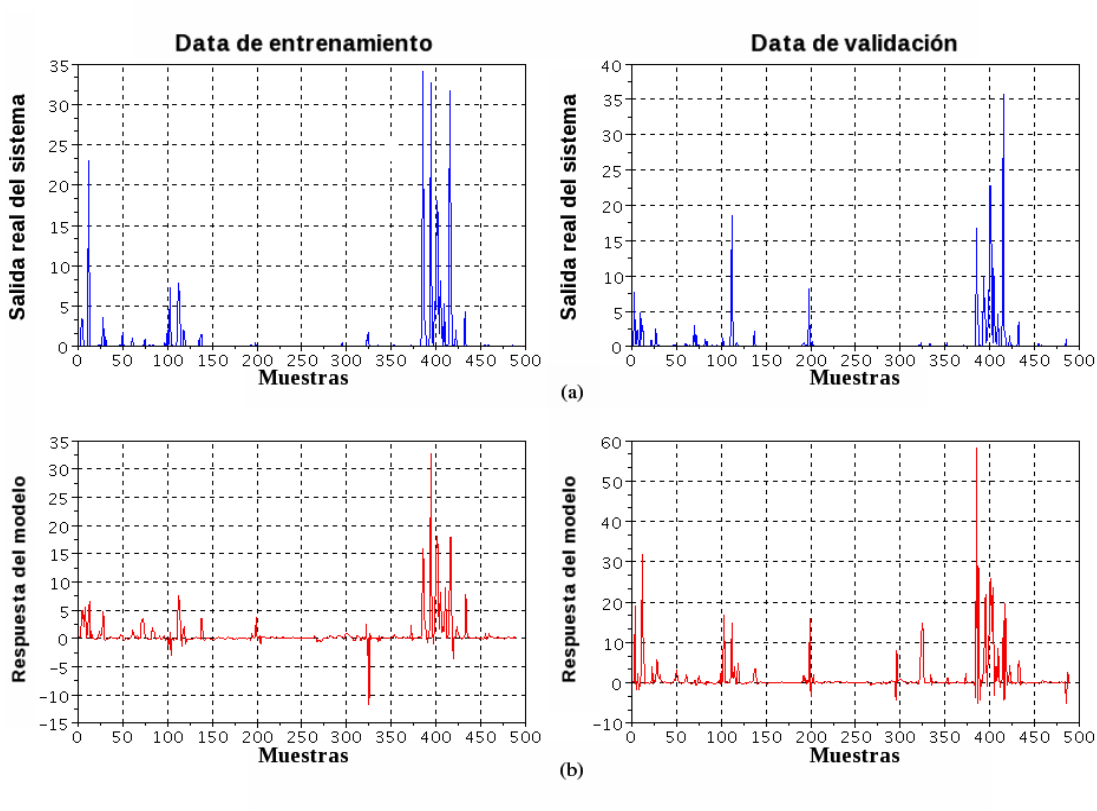


Figura 36: Menor error observado para los datos de validación :(a) Salida real del sistema; (b) Respuesta del modelo

Las respuestas de la Figura 36 arrojan errores de 1,904105 para el entrenamiento y 3,915608 para la validación, aun así el modelo ANFIS tiene dificultades para alcanzar todas las magnitudes de la salida real.

4.2.4 Simulación 4- ANFIS como Sistema de Alerta Temprana

Está experiencia usa el modelo ANFIS como herramienta de pronóstico para la alerta temprana de aludes torrenciales, de acuerdo con lo establecido en la sección 2.3.2, se usaron series horarias correspondientes a la estación “Macuto” desde las 12 am del 01-02-2005 hasta las 11 pm del 22-02-2005, la serie de niveles de alerta fue usada tanto para entrenamiento como para validación asignando las muestras impares al entrenamiento y las muestras pares a la validación, esto motivado a la ausencia de series con eventos similares en los registros históricos.

La Figura 37 muestra la curva de serpiente de la estación Macuto [23] para la fecha indicada, la línea roja indica la línea crítica y las demás líneas indican los diferentes niveles de alerta asignados.

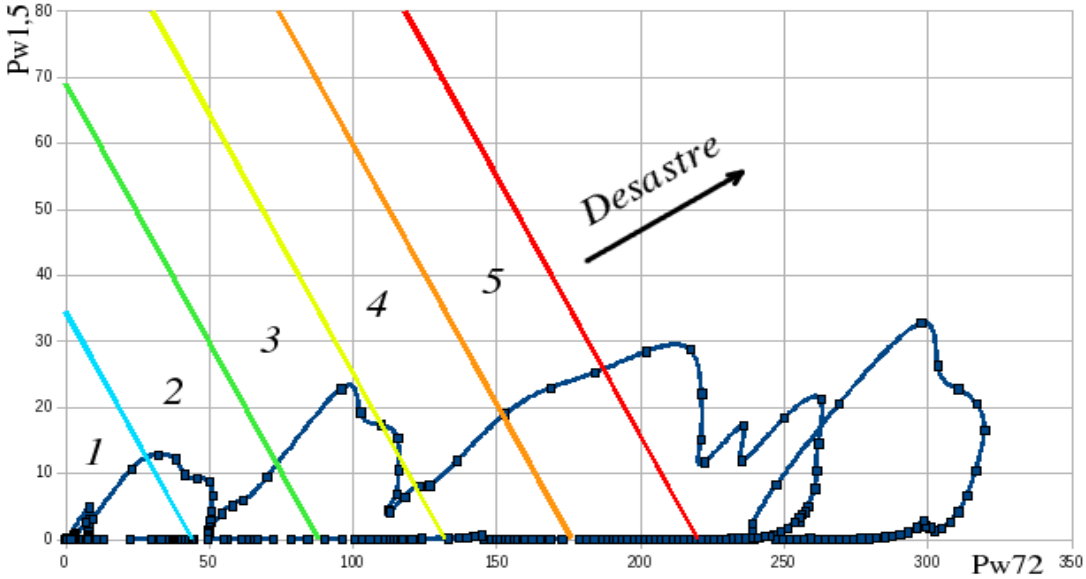


Figura 37: Curva de serpiente Macuto Febrero 2005 con respectivos niveles de alerta.

Los niveles de alerta fueron establecidos a intervalos igualmente espaciados de la denominada área segura, los eventos por encima de la línea crítica, se consideran eventos causales (se da por sentado que un alud torrencial ya ocurrió o está ocurriendo).

Previamente calculadas las precipitaciones ponderadas y los niveles de alerta se procedió a ordenar las muestras conforme a la expresión (16) de la sección 2.3.2, para *D=2 y P=Δ=1, 3, 6 y 12*, según:

$$[Pw72(t), Pw1,5(t), Alerta(t-\Delta), Alerta(t), Alerta(t+P)], \qquad (21)$$

Para cada caso se usaron 2 funciones de pertenencia por entrada con 16 reglas y 1000 iteraciones de entrenamiento, las figuras 38 y 39 muestran la respuesta del modelo para los datos de entrenamiento, donde el eje de las ordenadas indica el nivel de alerta pronosticados.

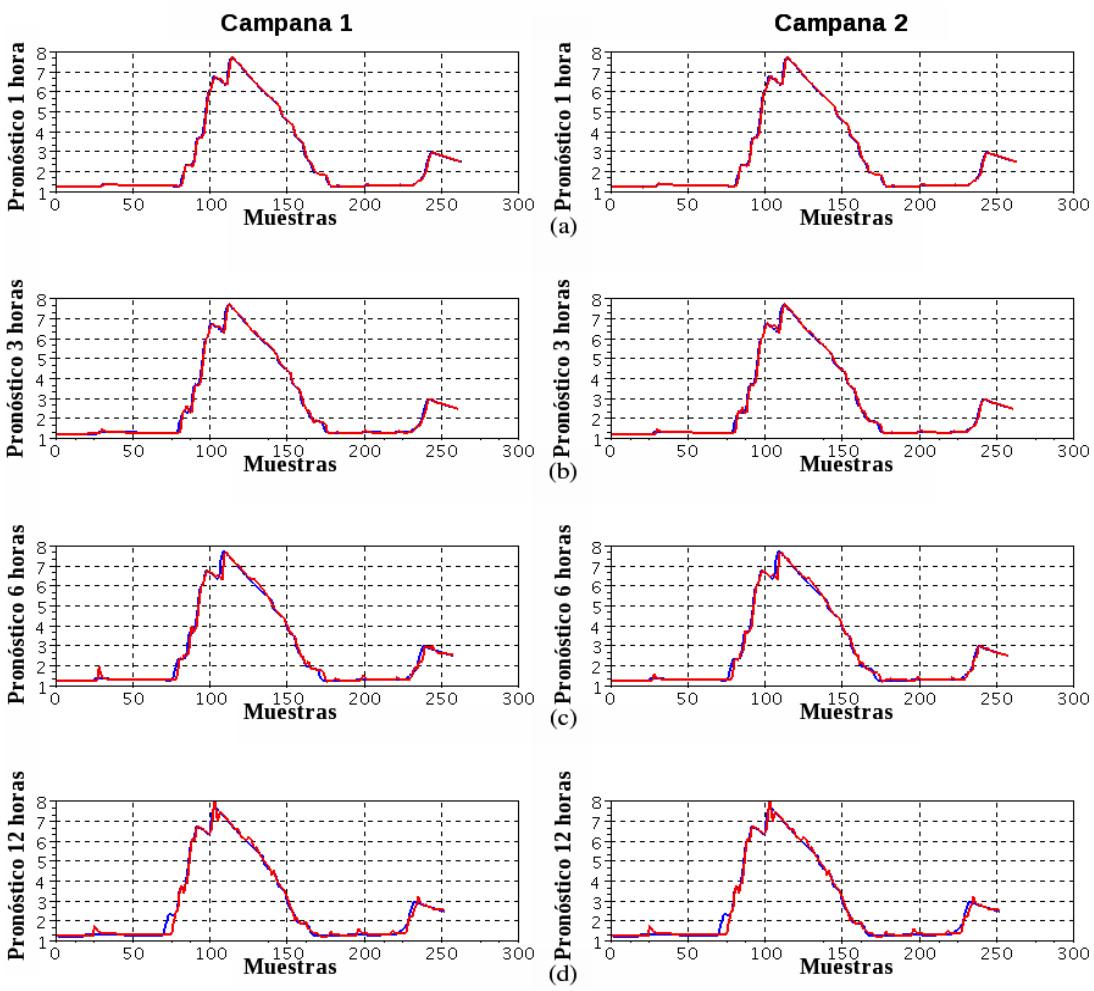


Figura 38: Respuesta del modelo para el entrenamiento con funciones en campana: (a) Pronóstico de 1 hora; (b) Pronóstico de 3 horas; (c) Pronóstico de 6 horas; (d) Pronóstico de 12 horas;

La tabla 9 resume los errores obtenidos según el número de horas pronosticadas y el tipo de función usado.

Tabla 9: Errores asociados a los datos de entrenamiento

	P=Δ=1	*P=Δ=3*	*P=Δ=6*	*P=Δ=12*
Campana 1	0,025176	0,066977	0,149623	0,167406
Campana 2	0,024776	0,062753	0,130909	0,178642
Trapezoidal	0,026013	0,092928	0,150030	0,235677
Triangular	0,003962	0,095774	0,164076	0,219561

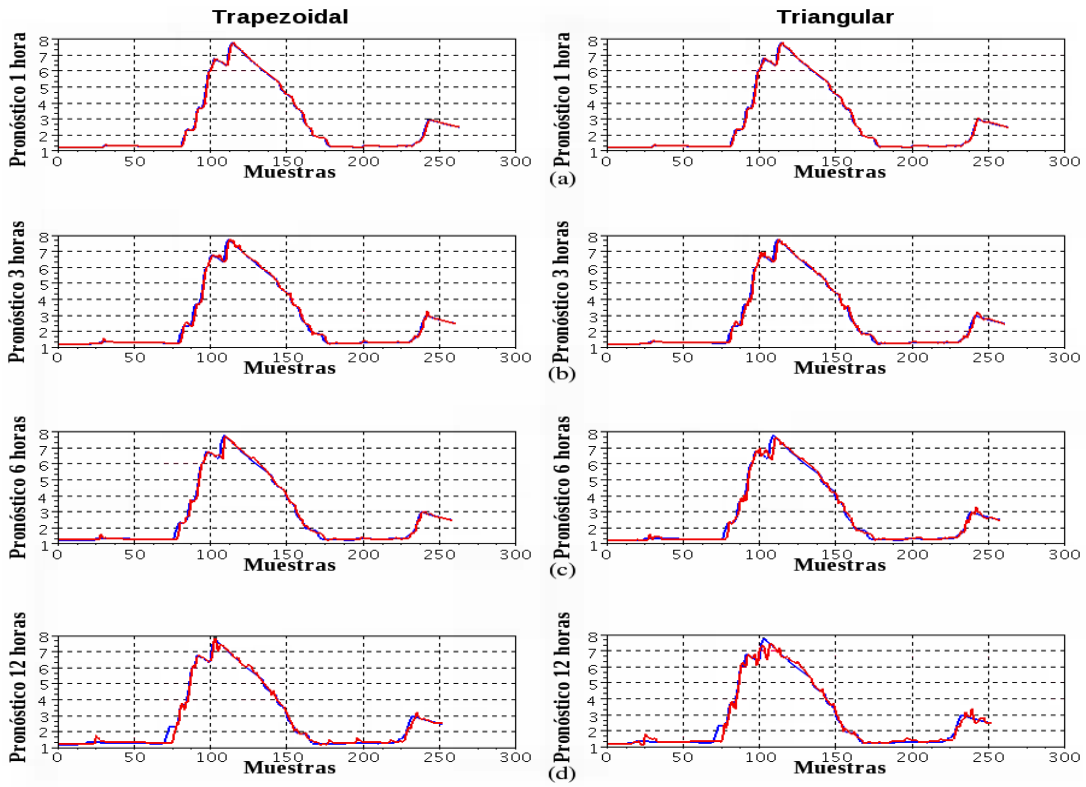


Figura 39: Respuesta del modelo para el entrenamiento con funciones trapezoidales y triangulares: (a) Pronóstico de 1 hora; (b) Pronóstico de 3 horas; (c) Pronóstico de 6 horas; (d) Pronóstico de 12 horas;

Finalmente en las figuras 40, 41, 42 y 43 se pueden apreciar las respuestas de cada modelo acorde a los datos de validación, nuevamente el eje de las ordenadas indica el nivel de alerta pronosticado.

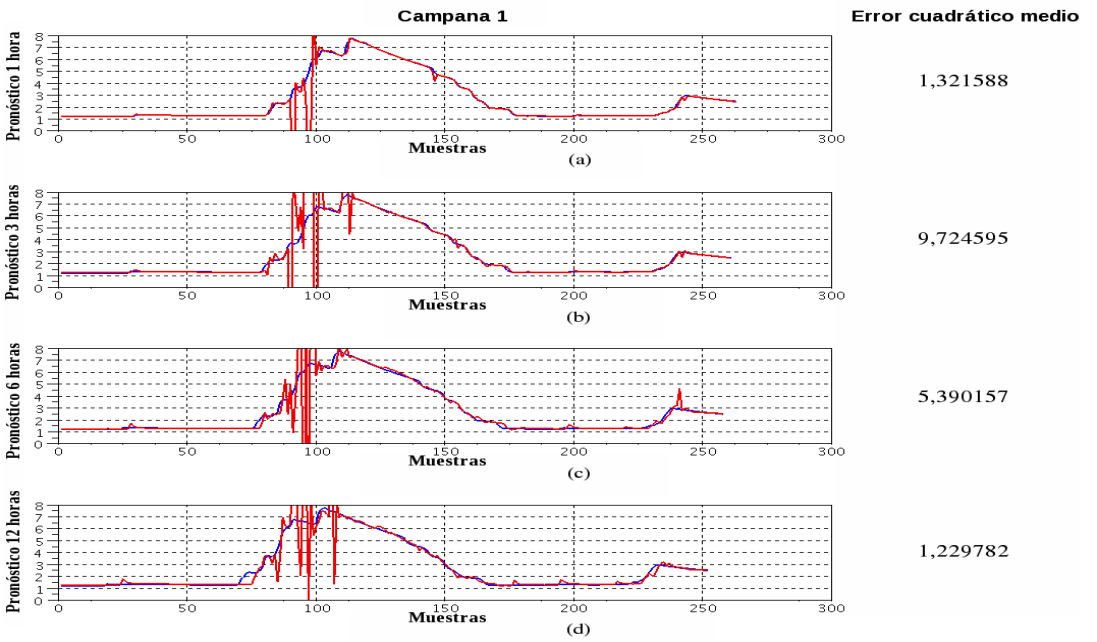


Figura 40: Respuesta del modelo para la función campana 1: (a) Pronóstico de 1 hora; (b) Pronóstico de 3 horas; (c) Pronóstico de 6 horas; (d) Pronóstico de 12 horas

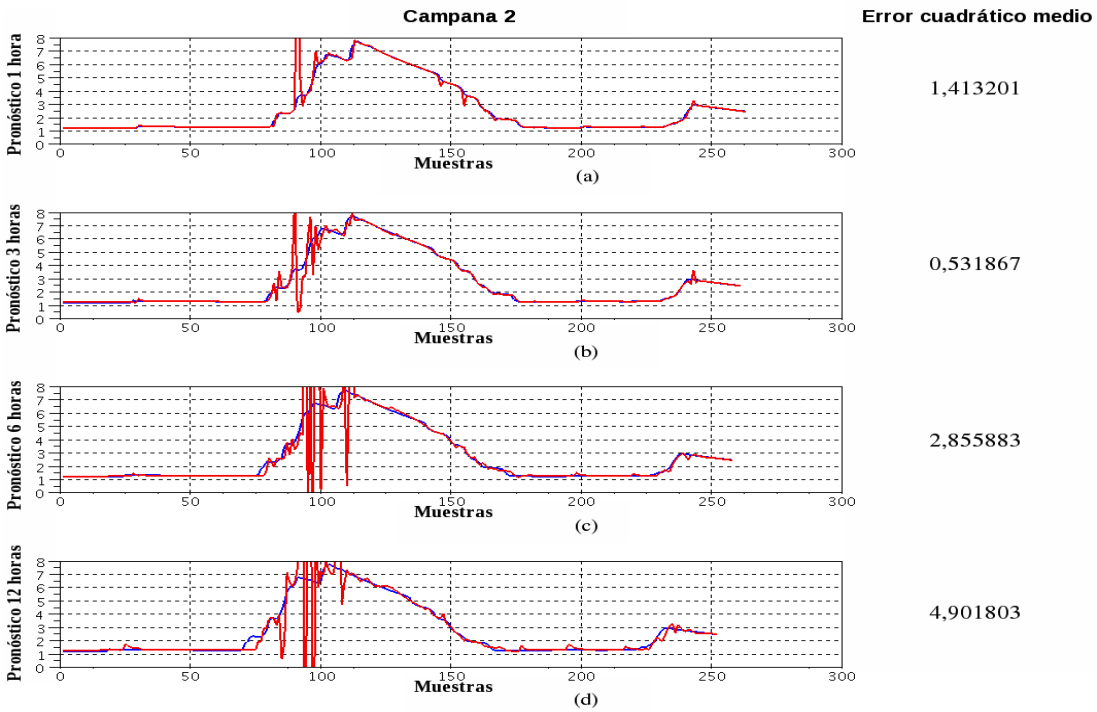


Figura 41: Respuesta del modelo para la función campana 2: (a) Pronóstico de 1 hora; (b) Pronóstico de 3 horas; (c) Pronóstico de 6 horas; (d) Pronóstico de 12 horas

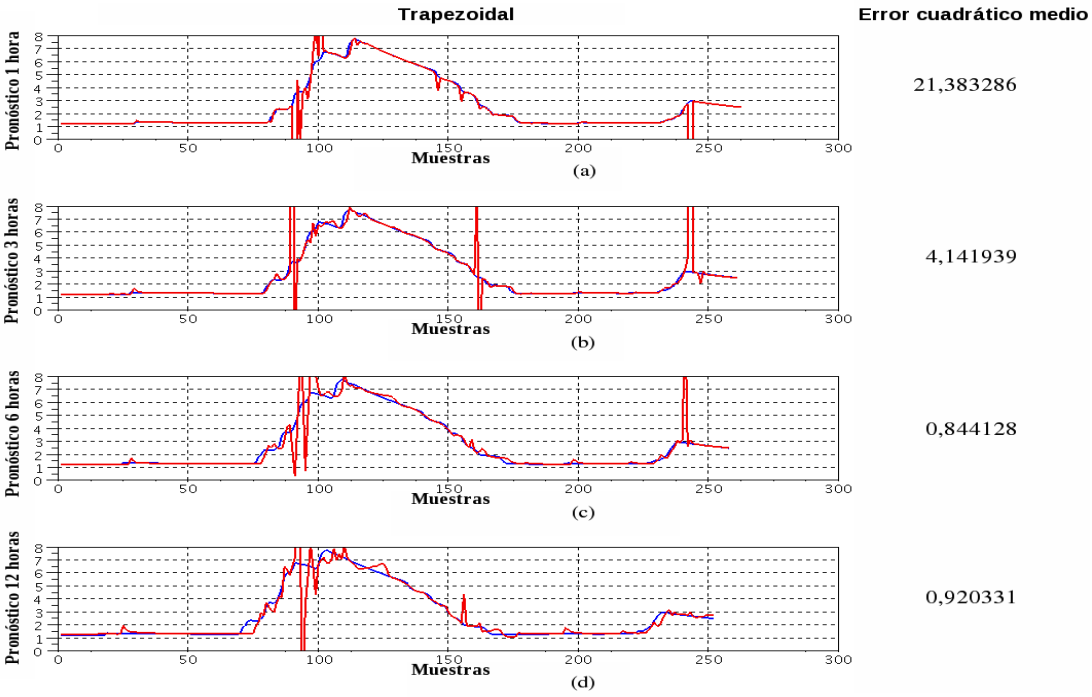


Figura 42: Respuesta del modelo para la función Trapezoidal: (a) Pronóstico de 1 hora; (b) Pronóstico de 3 horas; (c) Pronóstico de 6 horas; (d) Pronóstico de 12 horas

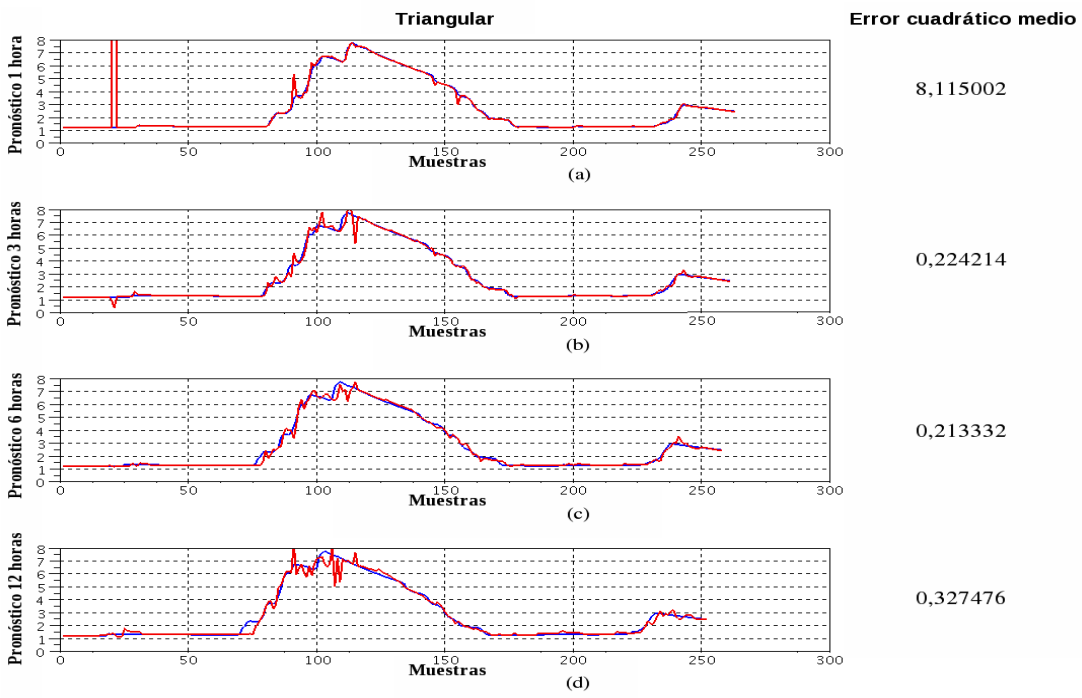


Figura 43: Respuesta del modelo para la función Triangular: (a) Pronóstico de 1 hora; (b) Pronóstico de 3 horas; (c) Pronóstico de 6 horas; (d) Pronóstico de 12 horas

Si bien los errores de validación del modelo son altos en comparación con los errores de entrenamiento, se pudo apreciar que estos errores son en su mayoría producto de picos y oscilaciones de pendiente pronunciada, en general, la respuesta del modelo mostró una tendencia a seguir la salida real, de modo que si limitamos el eje de las ordenadas (Pronóstico del modelo) a los niveles de alerta previamente alcanzados en el entrenamiento, resulta fácil filtrar estas oscilaciones y en consecuencia tener un pronóstico más acertado.

Conforme se aumentó el número de horas a pronosticar, los errores de entrenamiento de la tabla 9 mostraron una tendencia a ser cada vez más grandes, por otro lado los errores de validación no mostraron una tendencia clara, sin embargo al comparar los pronósticos de 1 hora de las figuras 40, 41, 42 y 43 con sus respectivos pronósticos para 3, 6 y 12 horas, se observa un decremento en la capacidad de respuesta ante cambios bruscos, sin embargo esto solo ocurre durante el primer nivel de alerta mejorando rápidamente después de alcanzar el segundo nivel.

Los resultados de este modelo, deben ser importados a la base de datos para que el servidor web, tenga acceso a ellos cumpliendo con lo establecido en la sección 3.4.

4.3. Errores Operativos del Software

- La interfaz gráfica no auto ajusta las escalas de la ventana de gráficos.

  Al ejecutar Anfislab por primera vez, el programa siempre falla en autoajustar las escalas de la ventana gráfica, esto parece ser un error interno de Scilab, fue reportado al equipo de Scilab y se espera que este error se corrija para la versión 5.2.

- La pestaña de “Resultados” no muestra todos los parámetros Antecedentes/Consecuentes.

  En la versión 4.x de Scilab era posible usar el estilo de texto del uicontrol para mostrar múltiples líneas del tipo String, pero las versiones 5.x no permiten esta operación, de modo que para mostrar los resultados se usó el estilo lista que esta limitado a 16 líneas, esta limitación también fue reportada al equipo de Scilab y se espera su corrección en las versiones siguientes, de momento la única forma de ver los parámetros en su totalidad es llamando a su variable correspondiente mediante la consola de Scilab, y por los archivos generados mediante la opción de exportación.

- Scilab se cierra al terminar las iteraciones de entrenamiento.

  Las librerías dinámicas desarrolladas poseen asignación dinámica de memoria, si los datos usados en el entrenamiento son muy grandes, poseen muchas entradas o se utilizan demasiadas funciones de pertenencia por entrada, entonces es posible que el programa requiera más memoria de la asignada por Scilab produciendo un fallo de segmentación que cerrará la consola, una solución consiste en aumentar el Stack de memoria usado por Scilab mediante la función “stacksize”, si aún así se presenta el mismo error entonces deberá usarse el ejecutable de procesamiento descrito en la sección 3.1.1.2.

CONCLUSIONES

Las simulaciones han demostrado las capacidades adaptativas del modelo de inferencia ANFIS, para refinar las reglas de razonamiento Si-Entonces en base a los mecanismos de aprendizaje híbridos y así ajustar su respuesta a series de tiempo caóticas y funciones altamente no lineales.

En los pronósticos realizados se obtuvieron respuestas favorables a las salidas reales de los datos de validación, pero en muchos casos las predicciones tuvieron dificultades en alcanzar las magnitudes de la salida de real, esto se debió a que la información contenida en la serie de entrenamiento no fue suficiente para reproducir sus características estocásticas, de modo que sería necesario el uso de series de entrenamiento más extensas, un mayor número de entradas y/o un mayor número de funciones de pertenencia por entrada, dado las necesidades de cómputo que implicaría un entrenamiento de ese tipo, dichas simulaciones no fueron presentadas en el trabajo.

Por otra parte, los pronósticos de los niveles de alerta mostraron un retraso en las respuestas del modelo conforme se aumentaban el número de horas a predecir, sin embargo, este retraso es rápidamente compensado y ya para el segundo nivel se tienen resultados satisfactorios, por lo que podemos concluir que las respuestas del modelo ante las verdaderas situaciones de peligro son muy acertadas.

La interfaz gráfica desarrollada en Scilab mostró ser una herramienta intuitiva y eficaz para la configuración del modelo de inferencia ANFIS tanto en el

entrenamiento como en la validación, los errores observados en dicha interfaz se debieron a cambios realizados en la transición de Scilab 4.x a 5.x sin embargo al ser un proyecto de código abierto en pleno desarrollo, estos errores pudieran ser depurados rápidamente. Adicionalmente Scilab contiene paquetes de paralelismo que pueden ser explotados en futuras mejoras del software y puede ejecutarse en modo embebido lo que es altamente deseable en un sistema de alerta temprana.

Considerando que cada cuenca tiene características propias las lineas criticas y los niveles de alerta no pueden tomarse de manera universal de modo que tanto el método del comité como la herramienta web se encuentran limitados a aquellas cuencas en donde el proyecto PROCEDA posee registros con suficiente información sobre eventos causales y no causales.

Los resultados obtenidos conjuntamente con las herramientas desarrolladas satisfacen los objetivos planteados, y con una base de datos operativa podrán ser fácilmente integrados, para la ejecución y puesta en marcha de un sistema automatizado de alerta temprana, que además podrá ser mejorado y adaptado a otras aplicaciones sin problemas de licencia, gracias a su total desarrollo con tecnologías de código abierto.

RECOMENDACIONES

El montaje y puesta a punto de la base de datos es indispensable para el correcto funcionamiento del sistema desarrollado, por lo que debe implementarse a la brevedad posible.

El sistema desarrollado mostró ser demasiado rígido con el uso de las funciones de pertenencia, de modo que podría modificarse para permitir usar distintas funciones en un mismo modelo y diferente número de funciones de pertenencia por entrada, adicionalmente podrían implementarse otras funciones a trozos, que permitan a las funciones de pertenencia trapezoidal y triangular tener pendientes asimétricas.

Las series de entrenamiento deben contener la mayor cantidad de información posible, se sugiere guardar registro de distintas configuraciones del modelo ANFIS y seleccionar, siempre que sea posible, la configuración más adecuada según las necesidades del momento.

Se recomienda un estudio más detallado del efecto de las variables meteorológicas, su comportamiento estacional y la posible aplicación de transformaciones en las series para minimizar los tiempos de entrenamiento.

Se propone el estudio e implementación de otros métodos de aprendizaje como, gradiente conjugado, algoritmos genéticos, Gauss-Newton ó el método de Levenberg-Marquardt, adicionalmente se recomienda probar con métodos que favorezcan la computación en paralelo como por ejemplo, el algoritmo de estimación

de Widrow-Hogg [1].

No se realizaron optimizaciones del software ni de los métodos usados en los pronósticos por lo que se sugiere una revisión por parte de un equipo multidisciplinario de expertos.

REFERENCIAS BIBLIOGRÁFICAS

[1] J.-S. Roger Jang, “ANFIS: Adaptative-Network-Based Fuzzy Inference System”, IEEE Transactions on systems, man, and cybernetics, VOL. 23, NO. 3, Mayo/junio 1993.

[2] L. A. Zadeh, “Outline of a New Approach to the Analysis of Complex Systems and Decision Processes”, IEEE Trans. SMC, SMC-3 (1), Enero, 1973, pp. 28-44

[3] L. A. Zadeh, Fuzzy Sets. Information and Control. 1965, New York Academic Press; 8: 338–353.

[4] T. Pérez Araque, “Sistemas Neurodifusos y Geneticodifusos Aplicados en Sistemas de Control”, Trabajo de grado para optar al título de Magister Scientiarum en Ingeniería Eléctrica. UCV. Caracas, 2005.

[5] T. Takagi, M. Sugeno, “Derivation of Fuzzy Control Rules From Human Operator's Control Action”, en Proc. IFAC Symp. Fuzzy Inform., Knowledge Representation and Decision Analisys, Julio 1983, pp. 55-60.

[6] M. Sugeno and G.T. Kang, “Structure Identification of Fuzzy Model”, Fuzzy Set and System, Vol. 28, pp. 15-33, 1988.

[7] J.-S. Roger Jang, C.-T. Sun, Eiji Mizutani “Neuro-Fuzzy and Soft Computing a Computational Approach to Learning and Machine Intelligence”, Prentice Hall, Upper Saddler River NJ, 1997

[8] http://www.redes-neuronales.netfirms.com/tutorial-redes-neuronales/tutorial-redes.htm Febrero de 2009

[9] http://www.ibiblio.org/pub/linux/docs/LuCaS/Presentaciones/200304curso-glisa/redes_neuronales/curso-glisa-redes_neuronales-html/x38.html Febrero de 2009

[10] P. Werbos, “Beyond regression: New tool for prediction and analysis in the Behavioral Sciences”, Ph.D. dissertation, Harvard Univ., Cambridge MA, 1974.

[11] D. E. Rumelhart, G. E. Hinton, y R.J. Williams. “Learning Internal Representations by Error Propagation”. In D. E. Rumelhart y James L. McClelland, Editors, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volume 1, chapter 8, pages 318-363. MIT Press, Cambridge, MA., 1986.

[12] J.-S. Roger Jang, “Fuzzy Modeling Using Generalized Neural Networks and Kalman Filter Algorithm,” en Proc. Ninth Nat. Conf. Artificial Intell. AAAI-91), July 1991, pp. 762-767.

[13] D. Loucks, J. R. Stedinger, D. A. Haith, “Water Resource Systems Planning and Analysis”, Prentice Hall Englewood. Cliffs NJ, 1981.

[14] http://www.scilab.org/legal/index_legal.php?page=license Marzo de 2009

[15] http://www.scilab.org/ Marzo de 2009

[16] http://www.fsf.org/ Marzo de 2009

[17] http://www.opensource.org/ Marzo de 2009

[18] http://gcc.gnu.org/ Marzo de 2009

[19] http://www.php.net/ Marzo de 2009

[20] http://www.cs.nthu.edu.tw/~jang/publication.htm Marzo de 2009

[21] http://www.scilab.org/contrib/index_contrib.php?page=download Marzo de 2009

[22] Tremante P., “Introducción al Control Difuso”, Trabajo presentado a la ilustre Universidad Central de Venezuela para optar a la categoría de Profesor Agregado en el escalafón universitario. UCV. Caracas, Mayo de 2006

[23] Gascon T., “Umbrales de Lluvia para Aludes Torrenciales en el Área Oeste del Estado Vargas”, Trabajo de grado para optar al título de Ingeniero Hidrometeorologista. UCV. Caracas, 2008

[24] http://www.scilab.org/doc/intersci.pdf Marzo de 2009

[25] http://www.aditus.nu/jpgraph/ Junio de 2009.

ANEXOS

ANEXO A

Definiciones Generales

A.1 Método de Propagación Hacia Atrás (Backpropagation)

Este método tiene dos variantes, una para el caso de redes con conexión hacia adelante y otra para el caso con conexión realimentada, dado el alcance de este trabajo, solo se definirá el método de propagación hacia atrás para las redes adaptativas con conexión hacia adelante.

Sea una red adaptativa de L capas y la k-ésima capa tiene k nodos, denotaremos al nodo en la i-ésima posición de la capa k por los índices (k,i) y su función de activación por O_i^k, esta depende de las señales entrantes y su conjunto de parámetros viene dada por la siguiente ecuación:

$$O_i^k = f\left(O_{1,}^{k-1} O_{2,}^{k-1} O_{\#(k-1)}^{k-1}, a, b, c, \ldots\right) \quad , \tag{A1.1}$$

donde a, b, c, etc. Son parámetros pertenecientes a este nodo. Suponiendo que los datos de entrenamiento tienen P muestras, entonces podemos definir el error de medición para la p-ésima muestra $(1 \le p \le P)$ como la sumatoria de los errores cuadráticos.

$$Ep = \sum_{m=1}^{\#L} \left(T_{(m,p)} - O_{(m,p)}^{L}\right)^2 \quad , \tag{A1.2}$$

donde $T_{m,p}$ es el m-ésimo componente del p-ésimo vector de salida deseado, y $O^L{}_{m,p}$ es el m-ésimo componente del vector de salida obtenido por aplicación del p-ésimo vector de entrada. Entonces, el error medido resultante es

$$E = \sum_{p=1}^{P} Ep \quad , \tag{A1.3}$$

En orden de desarrollar un procedimiento de aprendizaje que implemente el método del gradiente descendente en el error medido *E* sobre el espacio de los parámetros, primero debe calcularse la tasa de error para la *p*-ésima muestra y por cada nodo de salida *O*, es decir, derivar la ecuación con respecto al nodo de salida *i* de la capa *L*.

$$\frac{\partial Ep}{\partial O^{L}_{(i,p)}} = -2\left(T_{(i,p)} - O^{L}_{(i,p)}\right) \quad , \tag{A1.4}$$

Para los nodos *(k,i)* internos, el error puede ser propagado mediante la regla de la cadena de Werbos [10].

$$\frac{\partial Ep}{\partial O^{k}_{(i,p)}} = \sum_{m=1}^{\rightarrow k+1} \frac{\partial Ep}{\partial O^{k+1}_{(m,p)}} \frac{\partial O^{k+1}_{(m,p)}}{\partial O^{k}_{(i,p)}} \quad , \tag{A1.5}$$

donde $1 \leq k \leq L\text{-}1$ y puede verse que la tasa de error de un nodo interno puede representarse por la combinación lineal de las variaciones de error en las capas siguientes.

Ahora si α es un parámetro conocido se tiene.

$$\frac{\partial Ep}{\partial \alpha} = \sum_{O^{*} \in S} \frac{\partial Ep}{\partial O^{*}} \frac{\partial O^{*}}{\partial \alpha} \quad , \tag{A1.6}$$

Siendo *S* el conjunto de nodos cuyas salidas depende de α Entonces la

derivada de la medición total del error *E* con respecto a α es

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial Ep}{\partial \alpha} \quad , \tag{A1.7}$$

De acuerdo a la teoría asociada al gradiente descendente sin minimización de línea, la fórmula de actualización para un parámetro α es

$$\Delta \alpha = -\eta \frac{\partial Ep}{\partial \alpha} \quad , \tag{A1.8}$$

donde η es la taza de aprendizaje que puede ser expresada como

$$\eta = \frac{ka}{\sqrt{\sum_{\alpha} \left(\frac{\partial E}{\partial \alpha}\right)^2}} \quad , \tag{A1.9}$$

Donde ka es el tamaño del paso, es decir la longitud de cada transición del gradiente en el espacio del parámetro. Generalmente se suele cambiar el valor de ka para variar la velocidad de convergencia.

Existen dos paradigmas en las redes adaptativas, el aprendizaje fuera de línea donde la actualización del parámetro α esta basado en la ecuación (A1.7) y ocurre cuando han sido presentadas todas las muestras de entrenamiento, y el aprendizaje en línea donde los parámetros son actualizados inmediatamente después que se presenta cada muestra de entrenamiento basándose en la ecuación (A1.6).

A.2 Método de Aprendizaje Híbrido

El método de propagación hacia atrás tiende a ser lento y puede quedar atrapado en mínimos locales, Jyh-Shing Roger Jang [12] propone una regla de aprendizaje híbrida que combina los métodos del gradiente descendente con la

estimación por mínimos cuadrados (Least Square Estimate) para la identificación de parámetros. [1].

Por simplicidad, se asumirá que la red adaptativa bajo consideración tiene solo una salida.

$$Salida = F(\vec{I}, S) \quad , \tag{A2.1}$$

donde $\vec{I}$ es el conjunto de variables de entrada y S es el conjunto de parámetros. Si existe una función H tal que la composición de $H \circ F$ es lineal en algunos elementos de S, entonces esos elementos pueden se*r* identificados mediante el método de mínimos cuadrados.

Más formalmente, si el conjunto de parámetros puede descomponerse en dos conjuntos:

$$S = S_1 \oplus S_2 \quad , \tag{A2.2}$$

donde $\oplus$ representa suma directa tal que $H \circ F$ es lineal en los elementos de S_2. Ahora dados elementos de S_1 se puede inyectar P muestras de entrenamiento en la función resultante $H \circ F$ y formar una ecuación matricial de la forma:

$$AX = B \quad , \tag{A2.3}$$

donde X es un vector desconocido cuyos elementos son parámetros de S_2. Si $|S_2|=m$, entonces las dimensiones de A,X y B son pxm, mx1 y px1, respectivamente. Como p (número de muestras de entrenamiento) es usualmente mayor que m (número de

parámetros lineales) entonces se tiene un problema de sobredeterminación que generalmente “no tiene una única solución”, por tal motivo se usará la estimación por mínimos cuadrados (L.S.E.) de X,X^*, con las fórmulas secuenciales.

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1} a_{i+1} (B_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} &= S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}, i = 0,1,\ldots,P-1 \end{aligned} \quad , \qquad \text{(A2.4)}$$

donde el vector i-ésimo de la matriz A en la eq. (A2.3) corresponde a a_i^T y el i-ésimo elemento del vector B corresponde a b_i^T, entonces X puede ser calculado iterativamente. Las condiciones iniciales de la ecuación (A2.4) son Xo=0 y So=γI donde γ debe ser un número grande positivo (en nuestro caso γ=1000000) e I corresponde a la matriz identidad de dimensiones mxm.

La ecuación (A2.4) puede adaptarse para su aprendizaje en línea aplicando el factor de olvido λ.

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1} a_{i+1} (B_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} &= \frac{1}{\lambda}\left[S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{\lambda + a_{i+1}^T S_i a_{i+1}} \right] \end{aligned} \quad , \qquad \text{(A2.5)}$$

donde λ debe tener valores entre 0 y 1. Entre más pequeño sea λ, más rápido decaerán los efectos de los datos antiguos. Pero un λ muy pequeño causara inestabilidad, la cual debe evitarse.

Una descripción más detallada de este método pueden encontrarse en el texto original de ANFIS de J.-S. Roger Jang [1].

A.3 Alternativa al Cálculo de las Derivadas Parciales

El cálculo de una derivada por métodos numéricos es un proceso iterativo que puede resultar muy costoso computacionalmente hablando, al observar las ecuaciones que rigen el comportamiento de las capas del modelo ANFIS (sección 1.3.1) podemos observar que aunque sus parámetros cambien la función en sí, no es variable durante la ejecución del modelo, y en tal sentido las ecuaciones de sus derivadas parciales tampoco han de variar, en consecuencia es posible definir a priori las derivadas parciales de cada capa con solo conocer las características del modelo (*FP, L, k, etc.*) y con ello reducir los tiempos de ejecución y procesamiento.

En base a lo dicho anteriormente se obtiene la tabla 9

Tabla 10: Derivadas parciales de las capas 1, 2 y 3

$O_5 = \sum_i O_{4i}$	$\frac{\partial O_5}{\partial O_{4i}} = 1$
$O_{4i} = O_{3i}(p_i x + q_i y + r_i)$	$\frac{\partial O_{4i}}{\partial O_{3i}} = (p_i x + q_i y + r_i)$
$O_{3i} = \frac{O_{2j}}{\sum_m O_{2m}}$	$\frac{\partial O_{3i}}{\partial O_{2j}} = \frac{\sum_m O_{2m} - O_{2j}}{\sum_m O_{2m}^2}$ *si* $i = j$ *o* $\frac{\partial O_{3i}}{\partial O_{2j}} = \frac{O_{2i}}{\sum_m O_{2m}^2}$ *si* $i \neq j$

Como se dijo en la sección 1.3.1, la expresión de O_2 se obtiene de multiplicar las combinaciones lineales de los paramentos de O_1 que pertenezcan a distintas entradas, de modo que la derivada parcial de O_2 con respecto a un parámetro de O_1 resultará en el producto de sus otros componentes, por ejemplo la tabla 10 considera un componente de O_2 para un sistema de 3 entradas:

Tabla 11: Derivada parcial de un componente cualquiera de la capa 2 considerando un k=3

$O_2 = O_{1,1} \cdot O_{1,2} \cdot O_{1,3}$	$\frac{\partial O_2}{\partial O_{1,1}} = O_{1,2} \cdot O_{1,3}$ $\frac{\partial O_2}{\partial O_{1,2}} = O_{1,1} \cdot O_{1,3}$ $\frac{\partial O_2}{\partial O_{1,3}} = O_{1,1} \cdot O_{1,2}$

El número de componentes de O_2 se rige por la expresión L^k de modo que el número de derivadas parciales posibles para O_2 será del orden de $k \cdot L^k$.

Finalmente para la capa O_1, se tendrán 4 posibles resultados en base a la función de pertenencia, la tabla 4 resume las derivadas parciales de las funciones de pertenencia definidas en la tabla 1 sección 1.1.1.1.

Tabla 12: Derivadas parciales con respecto a los parámetros antecedentes

	(FP=1) Campana 1:	(FP=2) Campana 2:	(FP=3) Trapezoidal:	(FP=4) Triangular:
$\frac{\partial O_1}{\partial a}$	$\frac{2 \cdot b \cdot \left(\frac{x-c}{a}\right)^{2 \cdot (b-1)} \cdot (x-c)}{e^{\left(\frac{x-c}{a}\right)^{2b}} \cdot a^2}$	$\frac{2 \cdot b \cdot \left(\frac{x-c}{a}\right)^{2 \cdot (b-1)} \cdot (x-c)}{a^2 \left(\left(\frac{x-c}{a}\right)^{2 \cdot b} + 1\right)^2}$	$\begin{cases} -x+c+\frac{b}{2} & si \quad x > c+\frac{b}{2} \\ x-c+\frac{b}{2} & si \quad x < c-\frac{b}{2} \\ 0 & E.O.C. \end{cases}$	$\begin{cases} -x+c & si \quad x > c \\ x-c & si \quad x < c \\ 0 & E.O.C. \end{cases}$
$\frac{\partial O_1}{\partial b}$	$\frac{2 \cdot \left(\frac{x-c}{a}\right)^{2 \cdot b} \cdot ln\left(\frac{x-c}{a}\right)}{e^{\left(\frac{x-c}{a}\right)^{2b}}}$	$-\frac{2 \cdot \left(\frac{x-c}{a}\right)^{2 \cdot b} \cdot ln\left(\frac{x-c}{a}\right)}{\left(\left(\frac{x-c}{a}\right)^{2 \cdot b} + 1\right)^2}$	$\begin{cases} \frac{a}{2} & si \quad x < c-\frac{b}{2} \quad o \quad x > c+\frac{b}{2} \\ 0 & E.O.C. \end{cases}$	
$\frac{\partial O_1}{\partial c}$	$\frac{2 \cdot b \cdot \left(\frac{x-c}{a}\right)^{2 \cdot (b-1)}}{e^{\left(\frac{x-c}{a}\right)^{2b}} \cdot a}$	$\frac{2 \cdot b \cdot \left(\frac{x-c}{a}\right)^{2 \cdot (b-1)}}{a \left(\left(\frac{x-c}{a}\right)^{2 \cdot b} + 1\right)^2}$	$\begin{cases} a & si \quad x < c-\frac{b}{2} \quad o \quad x > c+\frac{b}{2} \\ 0 & E.O.C. \end{cases}$	$\begin{cases} a & si \quad x < c \quad o \quad x > c \\ 0 & E.O.C. \end{cases}$

A.4 Método del Comité [23]

Este método aparece en el IDI-Infrastructure Development Institute (2004). Ministry of Land, Infraestructure and Transport (2004) “Development of Warning

and Evacuation System Against Sediment Disasters in Developing Countries".Japón.

Considera todas las precipitaciones ponderadamente, incluyendo la reflejada en la abscisa y ordenada. Tanto para el indicador de largo plazo que refleja la precipitación acumulada como para el de corto plazo, que expresa la intensidad de la lluvia se usa la "precipitación de trabajo".

La precipitación de trabajo P_w es la suma de las precipitaciones de la serie de lluvias, afectadas por un coeficiente de reducción que refleja su antigüedad en la serie.

$$P_w = \alpha_1 P_1 + \alpha_2 P_2 + \alpha_3 P_3 = \sum \alpha_t P_t \qquad \text{(A4.1)}$$

Donde: P_t precipitación t horas antes del momento del cálculo

α_t coeficiente de reducción correspondiente a P_t

$$\alpha_t = 0{,}5^{\frac{t}{T}} \qquad \text{(A4.2)}$$

T "tiempo de media vida", es decir tiempo en que el coeficiente de reducción se hace igual a 0,5.

Si por ejemplo T=1 hora

$$P_w = 0{,}5 P_1 + 0{,}25 P_2 + 0{,}125 P_3 + ...$$

Figura 44: Formato del gráfico de evaluación del método del comité [23]

En el gráfico de evaluación (Figura 44) los dos indicadores son:

- en abscisa, la precipitación de trabajo calculada con tiempo de media vida (TMV) de 72 horas, que es usada como indicador de largo plazo.
- en ordenada, la precipitación de trabajo calculada con tiempo de media vida de 1,5 hora, expresando el indicador de corto plazo.

A.5 Otros Modelos Usados en la Hidrología

El estudio de los parámetros climáticos se constituye por muchas variables aleatorias cuyos valores cambian con el tiempo, los registros históricos de esas variables son conocidas como series temporales y las variables aleatorias pertenecientes a dichas series se conocen como procesos estocásticos. [13]

Existen varios modelos usados para la predicción de parámetros hidrológicos la preferencia de uno con respecto a otro depende de las características de la serie, en el presente trabajo solo se dará una descripción uno de ellos.

A.5.1 Autorregresivos (AR)

Son los modelos estocásticos dinámicos más simples desde el punto de vista computacional. La construcción de este modelo consiste en involucrar dependencia de parámetros anteriores con respecto al valor actual. Se tiene un modelo autorregresivo de primer orden (AR1) cuando se satisface la ecuación.

$$X_t = aX_{t-1} + \varepsilon_t, \qquad \text{(A5.1.1)}$$

donde a es una constante y $\{\varepsilon_t\}$ es un proceso aleatorio estacionario (generalmente asociado al error). Para el caso de un modelo autorregresivo de segundo orden (AR2) [13].

$$X_t = a_1 X_{t-1} + a_2 X_{t-2} + \varepsilon_t, \qquad \text{(A5.1.2)}$$

Análogamente se puede extender el caso anterior para un modelo autoregresivo de orden n (ARn).

$$X_t = \sum_{i=1}^{n} a_i X_{t-i} + \varepsilon_t, \qquad \text{(A5.1.3)}$$

Despejando $\{\varepsilon_t\}$ se tiene:

$$X_t + a_1 X_{t-1} + \ldots + a_k X_{t-k} = \varepsilon_t, \qquad \text{(A5.1.4)}$$

que resulta equivalente a decir

$$\alpha(B)X_t=\varepsilon_t, donde, \quad \alpha(z)=1+a_1z+...+a_lz^k \quad , \qquad \text{(A5.1.5)}$$

A.5.2 Promedio Móvil (MA)

Un modelo de promedio móvil de orden l (MAl) se expresa de la forma:

$$X_t=B_0\varepsilon_t+b_1\varepsilon_{t-1}+b_l\varepsilon_{t-l} \quad , \qquad \text{(A5.2.1)}$$

Donde b_0, b_1,...b_l, son constantes y $\{\varepsilon_t\}$ es un proceso aleatorio puro (generalmente asociado al error), entonces la expresión anterior es equivalente a

$$X_t=\beta(B)\varepsilon_t, donde, \quad \beta(z)=b_0+b_1z+...+b_lz^l \quad , \qquad \text{(A5.2.2)}$$

Sin perder generalidad podemos asumir b_0=1 ya que si no lo fuera se podría definir un nuevo proceso

$$\varepsilon_t^*=b_0\varepsilon_t(para\ todo\ t) \quad , \qquad \text{(A5.2.3)}$$

Si $\{\varepsilon_t\}$ es un proceso aleatorio pero entonces también lo es $\{\varepsilon^*_t\}$ y X_t puede expresarse en términos de $\{\varepsilon^*_t\}$ exactamente igual que en (A.5.2.1).

A.5.3 Mezcla de autorregresivo y promedio móvil (ARMA)

Es una combinación de los dos modelos anteriores, un modelo autorregresivo/promedio móvil de orden (k,l) (ARMA(k,l)) se denota como:

$$X_t + a_1 X_{t-1} + \ldots + a_k X_{t-k} = b_0 \varepsilon_t + b_1 \varepsilon_{t-1} + \ldots + b_l \varepsilon_{t-l}, \qquad \text{(A5.3.1)}$$

Donde nuevamente $\{\varepsilon_t\}$ es un proceso aleatorio puro (generalmente asociado al error) y $a_1,\ldots a_k, b_0,\ldots b_l$ son constantes. Al igual que antes podemos asumir $b_0=1$

Usando la misma notación de las dos secciones previas tenemos:

$$\alpha(B) X_t = \beta(B) \varepsilon_t, \qquad \text{(A5.3.2)}$$

Puede trabajarse esta ecuación para obtener una expresión X_t en términos de $\{\varepsilon_s; st\}$, si se asegura que los coeficientes $(a_1,\ldots a_k, b_0,\ldots b_l)$ son tales que X_t es estacionario, en cuyo caso se pueden ignorar las funciones complementarias (que decaen a cero) de modo que la ecuación anterior quedaría

$$X_t = \alpha^{-1}(B) \beta(B) \varepsilon_t, \qquad \text{(A5.3.3)}$$

Si se expande α-1(B) como una serie de transformadas Z entonces se puede expresar Xt como una combinación lineal de los valores presentes y pasados {εt}.

A.5.4 Autorregresivo Integrado y Promedio Móvil (ARIMA)

Box y Jenkins (1970) discutieron un tipo particular de proceso no estacionario en donde X_t no era estacionario pero su *d*-ésima derivada $\Delta^d X_t$ seguía el comportamiento de un modelo ARMA estacionario. De modo que usando la expresión $Yt=\Delta^d X_t$ (donde $\Delta=(1-B)$ denota el operador diferencial) se tiene la ecuación

$$\alpha(B) Y_t = \beta(B) \varepsilon_t,$$

$$\textit{donde} \quad \alpha(z) = \sum_{u=0}^{k} a_k z^k, \quad \beta(z) = \sum_{u=0}^{l} b_l z^l \quad (\alpha_0 = \beta_0 = 1) \qquad , \qquad \text{(A5.4.1)}$$

Salvo el uso de la conversión usada (Y_t) resulta idéntica (A.5.3.2).

ANEXO B

B.1. **Códigos Scilab**

Las rutinas mostradas en esta sección son las usadas por la interfaz gráfica desarrollada en Scilab, todas fueron creadas para satisfacer las necesidades de esta tesis, con la excepción de las rutinas fzfir.sci, bell_1.sci, bell_2.sci y trapeze.sci que fueron obtenidas del toolbox FISLAB (Fuzzy Inference Systems toolbox for SCILAB) de G. Ortega, (Email: gortega@estec.esa.nl), que a su vez los adapto del código de MATLAB (c) creado por Z. Lotfi, (Email: lotfia@s1.elec.uq.oz.au).

B.1.1. Rutina para cargar la interfaz gráfica (anfislab.sce)

```
//Hace que los radio botones de entrenamiento y validación sean mutuamente excluyentes
function [T]=exclu(Ti,op1,op2)
            if Ti==0 then
                  set(op2,'value','0');set(op1,'value','1');
            elseif Ti==1 then
                  set(op1,'value','0');set(op2,'value','1');
            end
            T=Ti;
endfunction

//Carga los componentes de la interfaz gráfica
function [Lst1,Lst2,Lst3,Lst4,op1,op2,E1,E2,E3,E4,E5,E6,ET1,ET2,ET3,ET4,T7]=toolbox()
f2=figure("figure_name","ANFISLAB","Position",[200,0,1000,600],...
                        "auto_resize","off","Units", "pixels");
                        xsetech([-0.1,0,1.2,0.7]);
                        da=gda();
                        da.auto_clear = 'on';

  // Remover los menús de Scilab
  delmenu(f2.figure_id, gettext("&File"));
  delmenu(f2.figure_id, gettext("&Tools"));
  delmenu(f2.figure_id, gettext("&Edit"));
  delmenu(f2.figure_id, gettext("&?"));
//Etiquetas Ti#
Ti1=uicontrol(f2,"style","text","string","DATOS","position",[0,130,200,20],...
          "horizontalalignment","center","background",[0.2,0.5,1]);
Ti2=uicontrol(f2,"style","text","string","PARAMETROS","position",[0,42,200,20],...
          "horizontalalignment","center","background",[0.2,0.5,1]);
Ti3=uicontrol(f2,"style","text","string","GENERAR FIS","position",[201,130,200,20],...
          "horizontalalignment","center","background",[0.2,0.5,1]);
Ti4=uicontrol(f2,"style","text","string","ENTRENA","position",[402,130,200,20],...
          "horizontalalignment","center","background",[0.2,0.5,1]);
```

```
Ti5=uicontrol(f2,"style","text","string","RESULTADOS","position",[603,130,385,20],...
        "horizontalalignment","center","background",[0.2,0.5,1]);
Ti6=uicontrol(f2,"style","text","string","Antecedentes","position",[603,109,192,20],...
        "horizontalalignment","center","background",[0.2,0.5,1]);
Ti7=uicontrol(f2,"style","text","string","Consecuentes","position",[796,109,192,20],...
        "horizontalalignment","center","background",[0.2,0.5,1]);
//texto T#
T3=uicontrol(f2,"style","text","string","# FUNCIONES DE PERTENENCIA",...
        "position",[201,21,150,20],"horizontalalignment","left");
T4=uicontrol(f2,"style","text","string","Error","position",[492,80,50,20],...
        "horizontalalignment","left");
T5=uicontrol(f2,"style","text","string","Iteraciones","position",[492,59,60,20],...
        "horizontalalignment","left");
T6=uicontrol(f2,"style","text","string","Error:","position",[402,21,35,20],...
        "horizontalalignment","left");
T7=uicontrol(f2,"style","text","string","inf","position",[437,21,70,20],...
        "horizontalalignment","left");
T8=uicontrol(f2,"style","text","string","Paso","position",[402,80,55,20],...
        "horizontalalignment","left");
//Listas Lst#
Lst1=uicontrol(f2,'style','listbox',"position",[201,60,200,60],...
        "horizontalalignment","left");
        set(Lst1, 'string', "Campana1|Campana2|Trapezoidal|Triangular");
        set(Lst1, 'value', 2);
Lst2=uicontrol(f2,'style','listbox',"position",[402,100,200,30],"horizontalalignment","left");
        set(Lst2, 'string', "Hibrido (offline)|Hibrido (online)");
        set(Lst2, 'value', 1);
Lst3=uicontrol(f2,"style","listbox","string",","position",[603,0,192,110],...
        "horizontalalignment","center");
Lst4=uicontrol(f2,"style","listbox","string",","position",[796,0,192,110],...
        "horizontalalignment","center");
//radio botones op#
op1=uicontrol(f2,"style","radiobutton","string","Entrenamiento",...
    "Min",0,"Max",1,"value",1,"Position",[0,110,200,20],"horizontalalignment","left",...
    "callback","[T]=exclu(0,op1,op2)");
op2=uicontrol(f2,"style","radiobutton","string","Validacion",...
    "Min",0,"Max",1,"value",0,"Position",[0,90,200,20],"horizontalalignment","left",...
    "enable","on","callback","[T]=exclu(1,op1,op2)");
//botones E#
E1=uicontrol(f2,"style","pushbutton","string","Cargar","position",[0,63,200,20],...
        "horizontalalignment","center", "background",[1,1,1],...
        "callback","[Ein,Echk]=cargasel(T,Echk,E2,E4,E5,E6,direccion)");
E2=uicontrol(f2,"style","pushbutton","string","Importar","position",[0,21,200,20],...
        "horizontalalignment","center", "background",[1,1,1],...
        "enable","off","callback","[k,L,FP,ka,abc,pqr]=import(Ein,Echk,ET1,ET4,Lst1,Lst3,Lst4,T7,
E5,E6)");
E3=uicontrol(f2,"style","pushbutton","string","Exportar","position",[0,0,200,20],...
        "horizontalalignment","center", "background",[1,1,1],...
        "enable","off","callback","export(Lst1,ET1,ET4,abc,pqr)");
E4=uicontrol(f2,"style","pushbutton","string","GENERAR","position",[201,0,200,20],...
        "horizontalalignment","center", "background",[1,1,1],...
        "enable","off","callback","[abc,pqr,FP,k,L]=genera(Ein,E1,E2,E3,E4,E5,ET1,T7,Lst1,Lst3,Ls
t4)");
```

```
E5=uicontrol(f2,"style","pushbutton","string","ENTRENAR","position",[402,39,200,20],...
             "horizontalalignment","center",      "background",[1,1,1],...
             "enable","off","callback","[abc,pqr,evol]=modela(Ein,Echk,L,FP,abc,pqr,Lst2,Lst3,Lst4,T7,E
T2,ET3,ET4,E4,E5,E6)");
E6=uicontrol(f2,"style","pushbutton","string","VALIDAR","position",[402,0,200,20],...
             "horizontalalignment","center",      "background",[1,1,1],...
             "enable","off","callback","[O,Yp]=valida(Echk,k,L,FP,abc,pqr,Lst3,Lst4,T7)");
//editbox ET#
ET1=uicontrol(f2,"style","edit","string","3","position",[371,21,30,20],...
             "horizontalalignment","right");
ET2=uicontrol(f2,"style","edit","string","0","position",[552,80,50,20],...
             "horizontalalignment","right");
ET3=uicontrol(f2,"style","edit","string","10","position",[552,59,50,20],...
             "horizontalalignment","right");
ET4=uicontrol(f2,"style","edit","string","0.01","position",[436,80,50,20],...
             "horizontalalignment","right");
endfunction
//Cargar librerías dinámicas, o Objetos compartidos.
direccion = get_absolute_file_path("anfislab.sce");
exec(direccion+'/'+'builder.sce');//construir macros del toolbox
exec(direccion+'/'+'interface/loader4.sce');//librería para aprendizaje online
exec(direccion+'/'+'interface/loader3.sce');//librería para validación del modelo.
exec(direccion+'/'+'interface/loader2.sce');//librería para aprendizaje offline
T=0;Echk=[];Ein=[];
[Lst1,Lst2,Lst3,Lst4,op1,op2,E1,E2,E3,E4,E5,E6,ET1,ET2,ET3,ET4,T7]=toolbox();
```

B.1.2. Rutina para generar los parámetros antecedentes iniciales (abcfixed.sci)

```
function abc=abcfixed(E,L,FP);

// abcfixed: Calcula los parámetros iniciales del modelo ANFIS
[p,ko]=size(E);
k=ko-1;

//parámetros antecedentes
if FP==1 then
  for h=1:k
    if L==1 then
      a(1,h)=max(E(:,h))*9999;
      c(1,h)=max(E(:,h));
    else
      pso(1,h)=(max(E(:,h))-min(E(:,h)))/(2*(L-1));
      c(:,h)=[min(E(:,h)):2*pso(1,h):max(E(:,h))]';
      a(1,h)=pso(1,h)*(1+0.0959573 );
    end;b=2;
  end
elseif FP==2 then
  for h=1:k
    if L==1 then
      a(1,h)=max(E(:,h))*9999;
      c(1,h)=max(E(:,h));
    else
```

```
      a(1,h)=(max(E(:,h))-min(E(:,h)))/(2*(L-1));
      c(:,h)=[min(E(:,h)):2*a(1,h):max(E(:,h))]';
    end;b=2;
  end
elseif FP==3 then
  for h=1:k
    if L==1 then
      b(1,h)=max(E(:,h))*9999;
      c(1,h)=max(E(:,h));
      a(1,h)=0;
    else
      pso(1,h)=(max(E(:,h))-min(E(:,h)))/(L-1);
      c(:,h)=[min(E(:,h)):pso(1,h):max(E(:,h))]';
      b(:,h)=pso(1,h)/2;
      a(:,h)=1/(pso(1,h)-b(:,h));
    end
  end
elseif FP==4 then
  for h=1:k
    if L==1 then
      b(1,h)=max(E(:,h))*9999;
      c(1,h)=max(E(:,h));
      a(1,h)=0;
    else
      pso(1,h)=(max(E(:,h))-min(E(:,h)))/(L-1);
      c(:,h)=[min(E(:,h)):pso(1,h):max(E(:,h))]';
      b(:,h)=0;
      a(:,h)=1/(pso(1,h));
    end
  end
end

//ordena matriz inicial abc
n=0;s=0;
  for t=1:k
    for i=s+1:1:L*t
      n=n+1;
      if n==L+1 then n=1;end
      abc(i,1)=a(1,t);
      if FP==1|FP==2 then
        abc(i,2)=b;
      elseif FP==3
            abc(i,2)=b(1,t);
      elseif FP==4
        abc(i,2)=0;
      end
      abc(i,3)=c(n,t);
      s=i;
    end
  end
endfunction
```

B.1.3. Rutina para cargar los datos (cargasel.sci)

```
function [Ein,Echk]=cargasel(T,Echk,E2,E4,E5,E6,direccion)

// Cargasel: Interfaz para la carga de datos de entrenamiento o validación
direccion=uigetfile(["*.*"],direccion);
E= fscanfMat(direccion);
[p,ka] = size(E)
  if T==0 then
     Ein=E;
     set(E2,"enable","on");
     set(E4,"enable","on");
     plot([1:p],Ein(:,ka));
  elseif T==1 then
     Echk=E;
     set(E2,"enable","on");
       if Ein~=[] then
         set(E6,"enable","on");
         plot([1:p],Echk(:,ka));
       end
  end
endfunction
```

B.1.4. Rutina para generar los parámetros iniciales (genera.sci)

```
function [abc,pqr,FP,k,L]=genera(Ein,E1,E2,E3,E4,E5,ET1,T7,Lst1,Lst3,Lst4)

// Genera: Carga la configuración en memoria genera los parámetros iniciales y los muestra.
set(E1,"enable","off");
set(E2,"enable","off");
set(E3,"enable","off");
set(E4,"enable","off");
set(E5,"enable","off");
[p,ko]=size(Ein);
k=ko-1;
FP=get(Lst1, "value");
Le=get(ET1, "string");
  if isnum(Le) then
    L=evstr(Le)
    if L~=int(L) then
      set(E1,"enable","on");
      set(E2,"enable","on");
      set(E3,"enable","on");
      set(E4,"enable","on");
      set(E5,"enable","on");
      error('El parámetro # funciones de pertenencia debe ser un número entero!')
    end
  else
    set(E1,"enable","on");
    set(E2,"enable","on");
    set(E3,"enable","on");
    set(E4,"enable","on");
    set(E5,"enable","on");
```

```
    error('El parámetro # funciones de pertenencia debe ser un número entero!')
  end
abc=abcfixed(Ein,L,FP)
grafica(Ein,k,abc,FP,L);
pqr=zeros(L^k,(k+1));
err='inf';
showres(abc,pqr,err,L,k,Lst3,Lst4,T7);
set(E1,"enable","on");
set(E2,"enable","on");
set(E3,"enable","on");
set(E4,"enable","on");
set(E5,"enable","on");
endfunction
```

B.1.5. Rutina para mostrar las funciones de pertenencia (grafica.sci)

```
function grafica(Ein,k,abc,FP,L)

// grafica: Dibuja la distribución de las funciones de pertenencia en el universo de discurso de la
primera entrada.
abcm=matrix(abc',3*k*L,1);
U(1,:)=[abc(1,3):(abc(L,3)-abc(1,3))/200:abc(L,3)];
m=0;
  for i=1:3:3*L;
    m=m+1;
    Ap(m,:) = fzfir(FP,U(1,:), abcm(i), abcm(i+1),abcm(i+2));
  end
delete()
plot(U,Ap');
endfunction
```

B.1.6. Rutina de llamar las librerías dinámicas del modelo ANFIS (modela.sci)

```
function [abc,pqr,evol]=modela(Ein,Echk,L,FP,abc,pqr,Lst2,Lst3,Lst4,T7,ET2,ET3,ET4,E4,E5,E6)

// modela: Pasa los parámetros desde la GUI a las librerías dinámicas y muestra los resultados.
winId=progressionbar('Espere');
set(E4,"enable","off");
set(E5,"enable","off");
set(E6,"enable","off");
[pin,ko]=size(Ein);
  if Echk==[] then
    pchk=0;Echk=zeros(1,ko)
  else
    [pchk,ko2]=size(Echk);
    if ko2 ~= ko then
      Echk=zeros(1,ko);pchk=0;
    end
  end
k=ko-1;
erre=get(ET2, "string");
  if isnum(erre) then
```

```
    err=evstr(erre)
    else winclose(winId); error('El parámetro error debe ser un número real!')
  end
itre=get(ET3, "string");
  if isnum(itre) then
    itr=evstr(itre)
    if itr~=int(itr) then
      winclose(winId);
      error('El parámetro iteraciones debe ser un número entero!')
    end
  else winclose(winId); error('El parámetro iteraciones debe ser un número entero!')
  end
kae=get(ET4, "string");
  if isnum(kae) then
    ka=evstr(kae)
  else winclose(winId); error('El parámetro paso debe ser un número real!')
  end
tpo=get(Lst2, "value");
  if tpo==1 then
    [abc,pqr,evola]=hytrain(k,L,pin,pchk,FP,itr,err,ka,abc,pqr,Ein,Echk);
    for i=1:itr
      if evola(i,2) == 0 then
        itr=i-1;break;
      else evol(i,:)=evola(i,:);
      end
    end
  elseif tpo==2 then
    [abc,pqr,evol]=hyontrain(k,L,pin,FP,ka,abc,pqr,Ein);
  end
delete()
plot([1:max(evol(:,1))],evol(1:max(evol(:,1)),2));
strka=string(evol(max(evol(:,1)),3));
set(ET4,'string',strka);
showres(abc,pqr,min(evol(:,2)),L,k,Lst3,Lst4,T7)
set(E4,"enable","on");
set(E5,"enable","on");
if Echk~=[] then
  set(E6,"enable","on");
end
winclose(winId);
endfunction
```

B.1.7. Rutina para mostrar los resultados en la interfaz gráfica (showres.sci)

```
function showres(abc,pqr,err,L,k,Lst3,Lst4,T7)

// Showres: Muestra los resultados en la GUI.
strpqr=string(pqr);
strgabc=string(abc);
for i=1:L*k
  show(i)=strcat(strgabc(i,:),'  ');
  show2(i)=strcat(strpqr(i,:),'  ');
```

```
end
set(Lst3,'string',strcat(show,'|'));
set(Lst4,'string',strcat(show2,'|'));
ero=string(err)
set(T7,'string',ero);
endfunction
```

B.1.8. Rutina de validación (valida.sci)

```
// Valida: Gráfica la predicción del sistema con respecto a la salida real de los datos de validación.
[pchk,ka]=size(Echk);
if ka==k then
  Echk=[Echk zeros(pchk,1)]
  O5=check(k,L,pchk,FP,abc,pqr,Echk);
  O=O5'
  if Ein~=[] then
    delete()
  end
  plot([1:pchk],[O]);
  Yp=[0];
elseif ka==k+1 then
  Ep=Echk(:,1:ka-1);
  Yp=Echk(:,ka);
  k=ka-1;
  O5=check(k,L,pchk,FP,abc,pqr,Echk);
  O=O5'
  if Ein~=[] then
    delete()
  end
  plot([1:pchk],[Yp O]);
  err=(Yp-O)^2;
  errchk=string(sqrt(sum(err)/pchk));
  set(T7,'string',errchk);
end
endfunction
```

B.1.9. Rutina de importación (import.sci)

```
function [k,L,FP,ka,abc,pqr]=import(Ein,Echk,ET1,ET4,Lst1,Lst3,Lst4,T7,E5,E6)

// Import: Importa los parámetros del sistema desde un archivo de texto.
direccion=uigetfile(["*.*"],direccion);
M = fscanfMat(direccion);
[a,b]=size(M)
k=M(1,1);
L=M(2,1);
FP=M(3,1);
ka=M(4,1);
abct=M(5: (4+3*k*L) ,1);
pqrt=M( (5+3*k*L):a,1);
[b,xd]=size(pqrt);
```

```
xc=b/L^k;
pqr=matrix(pqrt,L^k,xc);
abc=matrix(abct,k*L,3);
err='inf';
set(Lst1,"value",FP);
set(ET1,"string",string(L));
set(ET4,"string",string(ka));
if Ein~=[] then
  grafica(Ein,k,abc,FP,L);
  set(E5,"enable","on");
end
if Echk~=[] then
  grafica(Echk,k,abc,FP,L);
  set(E6,"enable","on");
end
pqr=matrix(pqr,(L^k),(k+1))
showres(abc,pqr,err,L,k,Lst3,Lst4,T7)
endfunction
```

B.1.10. Rutina de exportación (export.sci)

```
function export(Lst1,ET1,ET4,abc,pqr)

// Export: Exporta los parámetros del sistema a un archivo de texto.
FP=get(Lst1, "value");
Le=get(ET1, "string");
kae=get(ET4, "string");
L=evstr(Le);
ka=evstr(kae);
[b,xc]=size(pqr);
k=xc-1;
pqrt=matrix(pqr,L^k*xc,1);
abct=matrix(abc,k*L*3,1);
exporta=[k;L;FP;ka;abct;pqrt];
direccion=uigetfile(["*.*"],direccion);
fprintfMat(direccion,exporta)
endfunction
```

B.1.11. Función de pertenencia campana 1 (bell_1.sci)

```
function y = bell_1(x, a, b, c)

// BELL_1: Generalized bell-shaped membership function with  three
//       parameters a, b, c. (Defaults are 1, 1, and 0 respectively).
//
//                     Y=BELL_1(X, a, b, c)
//
//       Returns a matrix y with the same size as X; each element of
//       Y is a grade of membership.Y=exp(-((((X -c)/a)^2)^b) );
//
//                     See also BELL_2, SIGMOID, TRAPEZE and MF_PANEL.
```

```
// FISLAB: Fuzzy Inference Systems toolbox for SCILAB
// Created for MATLAB by
//          (c) A. Lotfi, University of Queensland (Email: lotfia@s1.elec.uq.oz.au) 13-10-93
// Translated to SCILAB
//          by G. Ortega, European Space Agency (Email: gortega@estec.esa.nl) 25-07-96
// The program has been tested on SCILAB version 2.2, PC Pentium, runing Linux 1.3.30

[nargout,nargin]=argn(0);
if nargin < 4 then c=0; end,
if nargin < 3 then b=1; end,
if nargin < 2 then a=1; end,
[m,n] = size(x);
for i=1:m
  for j=1:n
    y(i,j)=exp(-((((x(i,j) -c)/a) ^2) ^b));
  end
end
endfunction
```

B.1.12. Función de pertenencia Campana 2 (bell_2.sci)

```
function y = bell_2(x, a, b, c)

// BELL_2: Generalized bell-shaped membership function with three
//       parameters a, b, c. (defaults are 1,1, and 0).
//
//                   Y = BELL_2(X, a, b, c)
//
//       Returns a matrix y with the same size as X; each element
//       of Y is a grade of membership. Y=1/(1+(((X - c)/a)^2)^b);
//
//                See also BELL_1, SIGMOID,TRAPEZE and MF_PANEL.

//      Copyright (c) 1993 Jyh-Shing Roger Jang, U. C. Berkeley
//      jang@eecs.berkeley.edu
//      (Originally Tested on Matlab version 4.0a, HP workstation)
//      Permission is granted to modify and re-distribute this code
//      in any manner as long as this notice is preserved.
//      All standard disclaimers apply.
//      6-28-93.

// FISLAB: Fuzzy Inference Systems toolbox for SCILAB
// Created for MATLAB by
//          (c) A. Lotfi, University of Queensland (Email: lotfia@s1.elec.uq.oz.au) 13-10-93
// Translated to SCILAB
//          by G. Ortega, European Space Agency (Email: gortega@estec.esa.nl) 25-07-96
// The program has been tested on SCILAB version 2.2, PC Pentium, runing Linux 1.3.30

[nargout,nargin]=argn(0);
if nargin < 4 then, c=0;end,
if nargin < 3 then, b=1;end,
if nargin < 2 then, a=1;end,
```

```
[m,n] = size(x);
for i=1:m
  for j=1:n
    f=(x(i,j) - c)                  //
    if f==0 then f=0.0001; end, //modificado
    y(i,j)=1/(1+((f/a)^2)^b);   //
  end
end
endfunction
```

B.1.13. Función de pertenencia trapezoidal y triangular (trapeze.sci)

```
function f=trapeze(x,s,t,c)
//TRAPEZE:  Membership function for matrix X will be drawn using a
//         trapezoidal shape.
//
//                        [Y]=TRAPEZE(X,a,b,c)
//
//         Where a, b and c are slope, flatness and center of trapezoidal.
//         Defaults value for c and b and a are 0, 0 and 1 respectively.
//
//                      See also BELL_1, BELL_2, SIGMOID, and MF_PANEL.


// FISLAB: Fuzzy Inference Systems toolbox for SCILAB
// Created for MATLAB by
//           (c) A. Lotfi, University of Queensland (Email: lotfia@s1.elec.uq.oz.au) 13-10-93
// Translated to SCILAB
//           by G. Ortega, European Space Agency (Email: gortega@estec.esa.nl) 25-10-96
// The program has been tested on SCILAB version 2.2, PC Pentium, runing Linux 1.3.30

[nargout,nargin]=argn(0);
if nargin < 4 then, c=0; end,
if nargin < 3 then, t=0; end,
if nargin < 2 then, s=1; end,
if s == 0 then, s=2.22e-16; end,    //taken from MATLAB eps function
z=c+t/2+1/s;
y=c-t/2-1/s;
[m,n]=size(x);
for i=1:m
  for j=1:n
    if x(i,j) >= z | x(i,j) <= y then
      f(i,j) = 0;
    elseif x(i,j) < c+t/2 & x(i,j) > c-t/2
      f(i,j) = 1;
    elseif x(i,j) > c
      f(i,j) = s*(z-x(i,j));
    else
      f(i,j) = s*(x(i,j)-y);
    end,
  end,  //for j
end  //for i
```

```
endfunction
```

B.1.14. Selección de la función de pertenencia (fzfir.sci)

```
function ap = fzfir(n,u,a,b,c)
//FZFIR  The operation fuzzification has the effect of  transforming
//       a classical set/crisp value Ao into a fuzzy set Ap. The fuzzifier
//       developed in this function can handle five different types of
//            membership function for crisp value Ao. The universe must be
//            expressed an a ascending vector U.
//
//           [Ap] = FZFIR(1,U,a,b,Ao)  .........  bell_1.m
//           [Ap] = FZFIR(2,U,a,b,Ao)  .........  bell_2.m
//           [Ap] = FZFIR(3,U,s,t,Ao)  .........  trapeze.m
//           [Ap] = FZFIR(4,U,a,0,Ao)  .........  trapeze.m
//           [Ap] = FZFIR(5,U,0,0,Ao)  .........  fuzzy singleton
//
//            The third and forth input variables are parameters of related
//            membership function.
//
//                        See also BELL_1, BELL_2, TRAPEZE, SIGMOID, and DEFZFIR.

// FISLAB: Fuzzy Inference Systems toolbox for SCILAB
// Created for MATLAB by
//           (c) A. Lotfi, University of Queensland (Email: lotfia@s1.elec.uq.oz.au) 13-10-93
// Translated to SCILAB
//           by G. Ortega, European Space Agency (Email: gortega@estec.esa.nl) 25-07-96
// The program has been tested on SCILAB version 2.2, PC Pentium, runing Linux 1.3.30

u=u(:)';

// Checking the inputs of function
if n == 1 then,
  ap=bell_1(u,a,b,c);
elseif n == 2,
  ap=bell_2(u,a,b,c);
elseif n == 3,
  ap=trapeze(u,a,b,c);
elseif n == 4        //
  ap=trapeze(u,a,b,c);//modificado
else                 //
  ap=zeros(size(u));
  ap(max(find(abs(u-c) == min(abs(u-c))))) = 1;
end
endfunction
```

B.1.15. Rutina para construcción de macros (builder.sce)

```
mode(-1);
// specific part
libname='anfislablib'
// generic part
```

```
// get the absolute path
[units,typs,nams]=file();
clear units typs
for k=size(nams,'*'):-1:1
  l=strindex(nams(k),'builder.sce');
  if l<>[] then
    DIR=part(nams(k),1:l($)-1);
    break
  end
end
if ~MSDOS then
  if part(DIR,1)<>'/' then DIR=getcwd()+'/'+DIR,end
  MACROS=DIR+'macros/'
else
  if part(DIR,2)<>':' then DIR=getcwd()+'\'+DIR,end
  MACROS=DIR+'macros\'
end
//compile sci files if necessary and build lib file
genlib(libname,MACROS)
clear fd err nams DIR libname MACROS genlib
```

B.1.16. Rutina para la construcción de la librería de aprendizaje fuera de línea (hysci_builder.sce)

```
// generated with intersci
ilib_name = 'libhysci'                  // interface library name
files=['hysci' 'chytrain' 'hybrid' 'input' 'forward' 'kalman' 'backward' 'stepsize' 'de_dp' 'trn_err' 'new_para'
'chk_err' 'datastru' 'debug' 'lib' 'output' ];
libs=[];
table =["hytrain","intshytrain"];
ilib_build(ilib_name,table,files,libs);
```

B.1.17. Rutina para la construcción de la librería aprendizaje en línea (hyon_builder.sce)

```
// generated with intersci
ilib_name = 'libhyon'                   // interface library name
files=['hyon' 'chyontrain' 'hyonline' 'input' 'forward' 'LSE_kalman' 'backward' 'stepsize' 'de_dp' 'trn_err'
'new_para' 'datastru' 'lib' 'output' ];
libs=[];
table =["hyontrain","intshyontrain"];
ilib_build(ilib_name,table,files,libs);
```

B.1.18. Rutina para la construcción de la librería de validación (check_builder.sce)

```
// generated with intersci
ilib_name = 'libchecka'                 // interface library name
files=['check' 'ccheck' 'checka' 'input' 'forward' 'datastru' 'debug' 'lib'];
libs=[];
```

```
table =["check","intscheck"];
ilib_build(ilib_name,table,files,libs);
```

B.1.19. Rutina para cargar la librería de aprendizaje en línea (loader2.sce)

```
// ------------------------------------------------------
// generated by builder.sce: Please do not edit this file
// ------------------------------------------------------

libhysci_path = get_file_path('loader2.sce');
list_functions = [            'hytrain';
];
addinter(libhysci_path+'/libhysci.so','libhysci',list_functions);
// remove temp. variables on stack
clear libhysci_path;
clear list_functions;
clear get_file_path;
// ------------------------------------------------------
```

B.1.20. Rutina para cargar la librería de aprendizaje fuera de línea (loader4.sce)

```
// ------------------------------------------------------
// generated by builder.sce: Please do not edit this file
// ------------------------------------------------------

libhyon_path = get_file_path('loader4.sce');
list_functions = [            'hyontrain';
];
addinter(libhyon_path+'/libhyon.so','libhyon',list_functions);
// remove temp. variables on stack
clear libhyon_path;
clear list_functions;
clear get_file_path;
// ------------------------------------------------------
```

B.1.21. Rutina para cargar la librería de validación (loader3.sce)

```
// ------------------------------------------------------
// generated by builder.sce: Please do not edit this file
// ------------------------------------------------------

libchecka_path = get_file_path('loader3.sce');
list_functions = [            'check';
];
addinter(libchecka_path+'/libchecka.so','libchecka',list_functions);
// remove temp. variables on stack
clear libchecka_path;
clear list_functions;
clear get_file_path;
// ------------------------------------------------------
```

B.2. **Códigos lenguaje C**

Las siguientes rutinas corresponden a una adaptación del código ANFIS de dominio público usado por J.-S. Roger Jang en su trabajo de 1993 [1], los códigos originales pueden ser obtenidos de la página de publicaciones del autor [20], por motivos de licencia todos los archivos tienen el siguiente aviso.

```
/*
  Copyright (c) 1991 Jyh-Shing Roger Jang, Dept of EECS, U.C. Berkeley
  BISC (Berkeley Initiative on Soft Computing) group
  jang@eecs.berkeley.edu
  Permission is granted to modify and re-distribute this code in any manner
  as long as this notice is preserved.  All standard disclaimers apply.
*/
```

B.2.1. Archivos de cabecera (anfis.h, misc_def.h y standard.h)

anfis.h

```
------------------------------
#include "standard.h"

/* data structure */

typedef struct node_s {
        int index;              /* global node index within network */
        int layer;              /* which layer */
        int local_index;   /* local node index within layer */
        int parameter_n;  /* parameter no. */
        double value;           /* node value */
        double tmp;             /* for holding temporatory result */
        double de_do;           /* derivative of E to O */
        int function_index;     /* node function index  */
        struct node_list_s *fan_in;/* list of fan_in nodes */
        struct node_list_s *fan_out;        /* list of fan_out nodes */
        struct parameter_list_s *parameter;/* list of parameters */
} NODE_T;

typedef struct integer_list_s {
        int index;
        struct integer_list_s *next;
} INTEGER_LIST_T;

typedef struct node_list_s {
        NODE_T *content;
        struct node_list_s *next;
} NODE_LIST_T;
```

```
typedef struct parameter_list_s {
        int fixed;
        double content;
        double de_dp;
        struct parameter_list_s *next;
} PARAMETER_LIST_T;

/* global variables */
extern In_n;  /* number of input variables */
extern Mf_n;   /* number of membership functions along each input */
extern Node_n; /* number of total nodes */
extern Rule_n;    /* number of nodes in the 4-th layer */
extern funper;   /*funcion de pertenencia*/
extern NODE_T **node_p;
```

misc_def.h

```
----------------------------
/* file definition */
/*#define TRAIN_ERR_FILE   "error.trn"
#define CHECK_ERR_FILE   "error.chk"*/
#define INIT_PARA_FILE   "para.ini"
#define FINA_PARA_FILE   "para.fin"
#define TRAIN_DATA_FILE  "data.trn"
#define CHECK_DATA_FILE  "data.chk"
/*#define STEP_SIZE_FILE   "stepsize"*/
#define ARCH_EVOLUCION   "evol.txt"

/* error type definition, see trn_err.c */
/* 0 --> RMSE (root mean square error)
   1 --> NDEI (non-dimensional error index)
   2 --> ARV  (average relative variance)
   3 --> APE  (average percentage error) */
#define ERROR_TYPE 0
```

standard.h

```
----------------------------
/* standard header files */
#include <stdio.h>
#include <strings.h>
#include <stdlib.h>
#include <math.h>

/* handy macros */
#define ABS(x)   ( x > 0 ? (x): (-(x)) )
#define MAX(x,y) ( x > y ? (x) : (y) )
#define MIN(x,y) ( x < y ? (x) : (y) )
#define SGN(x)   ( x > 0 ? (1) : (-1) )

/* library function declaration */
/* memory allocation */
char *create_array();
```

```
char **create_matrix();
char ***create_cubic();
void free_array();
void free_matrix();
void free_cubic();

/* print to stdio and write to files */
void print_array();
void print_matrix();
void write_array();
void write_matrix();

/* file open */
FILE *open_file1();
FILE *open_file2();

/* matrix functions */
void m_plus_m();
void m_minus_m();
void m_times_m();
void s_mult_m();
void s_times_m();
void m_transpose();
double m_norm();

/* least square estimate */
void initialize_kalman();
double LSE_kalman();

/* random number generator */
double random1();
double random2();

/* others */
void exit1();
char *basename();
```

B.2.2. Rutina de paso hacia adelante (forward.c)

```
#include "anfis.h"

/* calculate node outputs from node 'from' to node 'to' */
void
calculate_output(from, to)
int from, to;
{
        double input(), funsel(), multiply();
        double normalize(), consequent(), sumatoria();
        static double (*function[6])() = {input, funsel, multiply,
                        normalize, consequent, sumatoria};
        int i;
        int function_index;
        if ((from > to) || (from < In_n) || (to >= Node_n))
```

```
                exit1("Error in calculate_output!");
        for (i = from; i <= to; i++) {
                function_index= node_p[i]->function_index;
                node_p[i]->value = (*function[function_index])(i);
        }
}

double
input(node_index)
int node_index;
{
        printf("This shouldn't have been called!\n");
        return(0);
}

double
multiply(node_index)
int node_index;
{
        double product = 1.0;
        NODE_LIST_T *p;

        NODE_LIST_T *arg_p = node_p[node_index]->fan_in;
        PARAMETER_LIST_T *para_p = node_p[node_index]->parameter;
        if (para_p != NULL)
                exit1("Error in multiply!");
        for (p = arg_p; p != NULL; p = p->next)
                product *= p->content->value;
        return(product);
}

double
normalize(node_index)
int node_index;
{
        NODE_LIST_T *arg_p = node_p[node_index]->fan_in;
        PARAMETER_LIST_T *para_p = node_p[node_index]->parameter;
        int i;
        double denom = 0;
        NODE_LIST_T *p;

        if (para_p != NULL)
                exit1("Error in normalize!");

        for (p = arg_p; p != NULL; p = p->next)
                denom += p->content->value;

        p = arg_p;
        for (i = 0; i < node_p[node_index]->local_index; i++)
                p = p->next;

        if (denom == 0)
    denom=1;
```

```
/*                      exit1("Error in normalize!");*/
        return(p->content->value/denom);
}

double
consequent(node_index)
int node_index;
{
        NODE_LIST_T *arg_p = node_p[node_index]->fan_in;
        PARAMETER_LIST_T *para_p = node_p[node_index]->parameter;
        int i;
        double x, a, total = 0;
        for (i = 0; i < In_n + 1; i++) {
                x = arg_p->content->value;
                a = para_p->content;
                if (i == In_n)
                        break;
                total += x*a;
                arg_p = arg_p->next;
                para_p = para_p->next;
        }
        return(x*(total + a));
        /*
        double a, b, c, wn, x1, x2;
        x1 = arg_p->content->value;
        x2 = arg_p->next->content->value;
        wn = arg_p->next->next->content->value;
        a = para_p->content;
        b = para_p->next->content;
        c = para_p->next->next->content;
        return((a*x1 + b*x2 + c)*wn);
        */
}

double
sumatoria(node_index)
int node_index;
{
        NODE_LIST_T *arg_p = node_p[node_index]->fan_in;
        NODE_LIST_T *t;
        double total = 0;
        if (arg_p == NULL)
                exit1("Error! Given pointer is NIL!");

        for (t = arg_p; t != NULL; t = t->next)
                total += t->content->value;
        return(total);
}
/*modificaciones de Andrés Avendaño*/

double
funsel(node_index)
int node_index;
```

```
{
        double bell_1(),bell_2(),trapezoidal(),triangular();
        switch (funper) {
                case 1:
                        return(bell_1(node_index));
                case 2:
                        return(bell_2(node_index));
                case 3:
                        return(trapezoidal(node_index));
                case 4:
                        return(triangular(node_index));
                default:
                        exit1("Error en funper, debe asignar un entero entre 1 y 4!");
        }
}
/* membership function = exp(-(pow(pow((x - c)/a, 2),b))*/
double
bell_1(node_index)
int node_index;
{
        NODE_LIST_T *arg_p = node_p[node_index]->fan_in;
        PARAMETER_LIST_T *para_p = node_p[node_index]->parameter;
        double c, a, b, x;
        double tmp1, tmp2;

        x = arg_p->content->value;
        a = para_p->content;
        b = para_p->next->content;
        c = para_p->next->next->content;
        if (a == 0)
                exit1("Error in bell_1!");
        tmp1 = (x - c)/a;
        tmp2 = tmp1 == 0 ? 0 : pow(pow(tmp1, 2.0), b);
        return(1/(exp(tmp2)));
}
/* membership function = 1/(1+pow((x - c)/a, 2*b)) */
double
bell_2(node_index)
int node_index;
{
        NODE_LIST_T *arg_p = node_p[node_index]->fan_in;
        PARAMETER_LIST_T *para_p = node_p[node_index]->parameter;
        double c, a, b, x;
        double tmp1, tmp2;

        x = arg_p->content->value;
        a = para_p->content;
        b = para_p->next->content;
        c = para_p->next->next->content;
/*printf("a %10f.\t b %10fd.\t c %10f.\n", a, b, c);*/
        if (a == 0)
                exit1("Error in bell_2!");
        tmp1 = (x - c)/a;
```

```
        tmp2 = tmp1 == 0 ? 0 : pow(pow(tmp1, 2.0), b);
        return(1/(1+ tmp2));
}
/* membership function = trapezoidal */
double
trapezoidal(node_index)
int node_index;
{
        NODE_LIST_T *arg_p = node_p[node_index]->fan_in;
        PARAMETER_LIST_T *para_p = node_p[node_index]->parameter;
        double c, a, b, x;
        double tmp1, tmp2;

        x = arg_p->content->value;
        a = para_p->content;
        b = para_p->next->content;
        c = para_p->next->next->content;
        if (a == 0)
                exit1("Error en trapezoidal!");
        tmp1 = c+b/2+1/a;
        tmp2 = c-b/2-1/a;
    if ( (x >= tmp1) || (x <= tmp2) )
            return(0.0);
    else if ( (x < (c+b/2)) && (x > (c-b/2) ) )
        return(1);
    else if ( x >= (c + b/2)){
        return(a*(tmp1-x));
    }
    else if ( x <= (c - b/2)){
        return(a*(x-tmp1));
    }
    else
      exit1("Error en trapezoidal!");
}

double
triangular(node_index)
int node_index;
{
        NODE_LIST_T *arg_p = node_p[node_index]->fan_in;
        PARAMETER_LIST_T *para_p = node_p[node_index]->parameter;
        double c, a, x;
        double tmp1, tmp2;

        x = arg_p->content->value;
        a = para_p->content;
        c = para_p->next->next->content;
        if (a == 0)
                exit1("Error en triangular1!");
        tmp1 = c+1/a;
        tmp2 = c-1/a;
    if ( (x >= tmp1) || (x <= tmp2) )
            return(0.0);
```

```
    else if ( (x < (c) ) && (x > (c) ) )
        return(1);
    else if ( x >= (c)){
        return(a*(tmp1-x));
    }
    else if ( x <= (c)){
        return(a*(x-tmp1));
    }
    else
      exit1("Error en triangular2!");
}
```

B.2.3. Rutina para las entradas del sistema (input.c)

```
#include "anfis.h"

/* get parameters from abc & pqr */
void
get_parameter(abc,pqr)
double **abc, **pqr;
{
        int i, j, k,ka;
        int parameter_n;
        double tmp;
        PARAMETER_LIST_T *p;
j=0;ka=0;
        for (i = 0; i < Node_n; i++) {
                if (node_p[i]->parameter == NULL)
                        continue;
                parameter_n = node_p[i]->parameter_n;
                p = node_p[i]->parameter;

  if (j < In_n*Mf_n)
  {
                for (k = 0; k < 3; k++)
                        {
      tmp=abc[j][k];
                p->content = tmp;
                        p = p->next;
                        }
  j=j++;
  }
  else if (j >= In_n*Mf_n)
    {
                for (k = 0; k < (In_n+1); k++)
                        {
      tmp=pqr[j-In_n*Mf_n][k];
                p->content = tmp;
ka=ka++;
                        p = p->next;
                        }
  j=j++;
```

```
  }
        }
}
/* modificado*/
void
get_data(data_matrix, data_n, data_pointer)
int data_n;
double **data_matrix, **data_pointer;
{
        int i, j;
        double tmp;

        if (data_n == 0)
                return;
        for (i = 0; i < data_n; i++)
                for (j = 0; j < In_n + 1; j++)
                        {
        tmp = data_matrix[i][j];
                                data_pointer[i][j] = tmp;
            }
}

/* put input part of (j+1)-th data (training or checking) to input nodes. */
void
put_input_data(j, data_pointer)
int j;
double **data_pointer;
{
        int k;
        for (k = 0; k < In_n; k++)
    {
                node_p[k]->value = data_pointer[j][k];
    }
}
```

B.2.4. Rutina para las salidas del sistema (output.c)

```
#include "anfis.h"

/* write parameters to file 'file_name' */
void
write_parameter(abc,pqr)
double **abc, **pqr;
{
        int i, j, k, ka;
        double tmp;
        PARAMETER_LIST_T *p;
j=0;k=0;ka=0;
        for (i = 0; i < Node_n; i++) {
                if (node_p[i]->parameter == NULL)
                        continue;
                for (p = node_p[i]->parameter; p != NULL; p = p->next) {
```

```
        if (j < In_n*Mf_n)
          {
          tmp=p->content;
          abc[j][k]=tmp;
        k=k++;
          if (k>2){
          j=j++;
          k=0;
          }
        }
        else if (j >= In_n*Mf_n)
          {
          tmp=p->content;
          pqr[j-In_n*Mf_n][k]=tmp;
        k=k++;
          if (k>(In_n)){
          j=j++;
          k=0;
          }
        }
        }
            }

}

/* record the current parameters in an array */
void
record_parameter(parameter_array)
double *parameter_array;
{
            int i, j = 0;
            PARAMETER_LIST_T *p;

            for (i = 0; i < Node_n; i++) {
                        if (node_p[i]->parameter == NULL)
                                    continue;
                        for (p = node_p[i]->parameter; p != NULL; p = p->next)
                                    parameter_array[j++] = p->content;
            }
}

/* restore the parameter in parameter_array back to ANFIS */
void
restore_parameter(parameter_array)
double *parameter_array;
{
            int i, j = 0;
            PARAMETER_LIST_T *p;

            for (i = 0; i < Node_n; i++) {
                        if (node_p[i]->parameter == NULL)
                                    continue;
                        for (p = node_p[i]->parameter; p != NULL; p = p->next)
```

```
                        p->content = parameter_array[j++];
        }
}
```

B.2.5. Rutinas para la creación y manipulación de las capas y componentes nodales (datastru.c)

```
#include "anfis.h"
#include <stdio.h>
#include <math.h>

static int **config;

void
get_config()
{
        int i, j;
        int connected();
  int tmp;
config = (int **)create_matrix(Node_n, Node_n, sizeof(int));
        for (i = 0; i < Node_n; i++){
                for (j = 0; j < Node_n; j++){
      tmp = connected(i, j);
                        config[i][j] = tmp;
      }
        }
}

/* return 1 if there is a link from node i to node j */
int
connected(i, j)
int i, j;
{
        int which_layer();
        void other_rep();
        int layer1, layer2;
        int group;
        int position;
        static int *rep;
        static initialization;
        if (i >= j)
                return(0);
        layer1 = which_layer(i);
        layer2 = which_layer(j);
        if ((layer2 - layer1 != 1) && layer1 != 0)
                return(0);
        switch(layer1) {
                case 0:
                        /* connections from layer 0 to layer 1 */
                        if (between(In_n + i*Mf_n, j, In_n + (i+1)*Mf_n - 1))
                                return(1);
                        /* connections from layer 0 to layer 5 */
                        if (between(In_n + In_n*Mf_n + 2*Rule_n, j,
                                In_n + In_n*Mf_n + 3*Rule_n- 1))
```

```
                        return(1);
                    break;
                case 1:
                    if (initialization != 7151373){
                        rep = (int *)calloc(In_n, sizeof(int));
                        initialization = 7151371;
                    }
                    group = (i - In_n)/Mf_n;
                    position = (i - In_n) % Mf_n;
                    other_rep(rep, j - In_n - In_n*Mf_n);
                    if (rep[group] == position)
                        return(1);
                    break;
                case 2:
                    /* layer 2 and 3 are fully connected */
                    return(1);
                    break;
                case 3:
                    if (j - i == Rule_n)
                        return(1);
                    break;
                case 4:
                    /* layer 4 and 5 are fully connected */
                    return(1);
                    break;
                case 5:
                    return(0);
                    break;
                default:
                    printf("Error in layer!\n");
            }
            return(0);
}

/* other representation of j using mf_n as the base */
void
other_rep(rep, j)
int j;
int *rep;
{
            int i;
            for (i = 0; i < In_n; i++) {
                    rep[In_n - i - 1] = j % Mf_n;
                    /* j = (int)floor((double)(j/mf_n)); */
                    j = j/Mf_n;
            }
}

/* return the layer of given node index */
int
which_layer(index)
int index;
{
```

```
        int between();
        if (between(0, index, In_n - 1))
                return(0);
        if (between(In_n, index, In_n + In_n*Mf_n - 1))
                return(1);
        if (between(In_n + In_n*Mf_n, index, In_n + In_n*Mf_n + Rule_n- 1))
                return(2);
        if (between(In_n + In_n*Mf_n + Rule_n, index, In_n + In_n*Mf_n + 2*Rule_n- 1))
                return(3);
        if (between(In_n + In_n*Mf_n + 2*Rule_n, index, In_n + In_n*Mf_n + 3*Rule_n- 1))
                return(4);
        if (index = In_n + In_n*Mf_n + 3*Rule_n)
                return(5);
        printf("Error in which_layer!\n");
        exit(1);
}

/* return 1 if x is between l and u, that is, l <= x <= u */
int
between(l, x, u)
int l, x, u;
{
/*      printf("l%d \t x%d \t u%d \n",l,x,u);*/
        if ((l <= x) && (x <= u))
                return(1);
        else
                return(0);
}

void
build_anfis()
{
        void build_layer();
        NODE_LIST_T *build_node_list();
        int i;

        build_layer(0, In_n, 0, 0, 0);
        build_layer(1, In_n*Mf_n, In_n, 3, 1);
        build_layer(2, Rule_n, In_n+In_n*Mf_n, 0, 2);
        build_layer(3, Rule_n, In_n+In_n*Mf_n+Rule_n, 0, 3);
        build_layer(4, Rule_n, In_n+In_n*Mf_n+2*Rule_n, In_n + 1, 4);
        build_layer(5, 1, In_n+In_n*Mf_n+3*Rule_n, 0, 5);

        for (i = 0; i < Node_n; i++) {
                node_p[i]->fan_in  = build_node_list(0, i);
                node_p[i]->fan_out = build_node_list(1, i);
        }
}

/* Build a node list of layer-th layer, with n node,
   starting at index index, and each
   node has parameter_n parameters and node function
   function[function_index]. */
```

```
void
build_layer(layer, n, index, parameter_n, function_index)
int layer, n, index, parameter_n, function_index;
{
        PARAMETER_LIST_T *build_parameter_list();
        int i;
        NODE_T *q;

        if (n == 0)
                printf("Possible error in build_layer!\n");

        for (i = 0; i < n; i++) {
                q = (NODE_T *) malloc(sizeof (NODE_T));
                node_p[i + index] = q;
                q->index = i + index;
                q->layer = layer;
                q->local_index = i;
                q->parameter_n = parameter_n;
                q->function_index = function_index;
                q->parameter = build_parameter_list(parameter_n);
        }
}

/* Build a parameter list with length n. */
PARAMETER_LIST_T *
build_parameter_list(n)
int n;
{
        PARAMETER_LIST_T *head, *p, *q;
        int i;

        if (n < 0)
                exit1("Error in build_parameter_list()!");

        if (n == 0)
                return(NULL);

        head = (PARAMETER_LIST_T *) malloc(sizeof (PARAMETER_LIST_T));
        p = head;
        for (i = 1; i < n; i++){
                q = (PARAMETER_LIST_T *) malloc(sizeof (PARAMETER_LIST_T));
                p->next = q;
                p = q;
        }
        p->next = NULL;
        return(head);
}

/* type == 0 --> build node list along column n of matrix config.
   type == 1 --> build node list along row n of matrix config.     */
NODE_LIST_T *
build_node_list(type, n)
int type, n;
```

```
{
        NODE_LIST_T *p, *q, *dummy;
        int i;

        p = (NODE_LIST_T *) malloc(sizeof (NODE_LIST_T));
        dummy = p;
        dummy->next = NULL;
        if (type == 0)
                for (i = 0; i < Node_n; i++)
                        if (config[i][n]){
                                q = (NODE_LIST_T *) malloc(sizeof (NODE_LIST_T));
                                q->content = node_p[i];
                                p->next = q;
                                p = q;
                        }
        if (type == 1)
                for (i = 0; i < Node_n; i++)
                        if (config[n][i]){
                                q = (NODE_LIST_T *) malloc(sizeof (NODE_LIST_T));
                                q->content = node_p[i];
                                p->next = q;
                                p = q;
                        }
        p->next = NULL;
        q = dummy;
        dummy = dummy->next;
        free(q);
        return(dummy);
}

/* set parameter mode: fixed = 1 --> fixed parameter;
                       fixed = 0 --> modifiable parameter */
int
set_parameter_mode()
{
        int i, modifiable_p_count = 0;
        PARAMETER_LIST_T *p;

        for (i = 0; i < Node_n; i++) {
                if (node_p[i]->parameter == NULL)
                        continue;
                for (p = node_p[i]->parameter; p != NULL; p = p->next) {
                        p->fixed = 0;
                        modifiable_p_count++;
                }
        }
        printf("Modifiable parameters: %d\n", modifiable_p_count);
        return(modifiable_p_count);
}
```

B.2.6. Rutina para el cálculo de las derivadas parciales (de_dp.c)

```
#include "anfis.h"
```

```
/* calculate de/dp of each parameters */
void
update_de_dp()
{
        int i, j;
        PARAMETER_LIST_T *p;
        double do_dp;
        double derivative_o_p();

        for (i = 0; i < Node_n; i++) {
                if (node_p[i]->parameter == NULL)
                        continue;
                j = 0;
                for (p = node_p[i]->parameter; p != NULL; p = p->next) {
                        do_dp = derivative_o_p(i, j);
                        p->de_dp += (node_p[i]->de_do)*do_dp;
        if (i <= 10)
                        j++;
                }
        }
}

/* clear de/dp of each parameters */
void
clear_de_dp()
{
        int i;
        PARAMETER_LIST_T *p;

        for (i = 0; i < Node_n; i++) {
                if (node_p[i]->parameter == NULL)
                        continue;
                for (p = node_p[i]->parameter; p != NULL; p = p->next)
                        p->de_dp = 0;
        }
}

/* calculate the  derivative of node i w.r.t it's j-th parameter */
double
derivative_o_p(i, j)
int i, j;
{
        double dmf_fp();
        double dconsequent_dp();
        int layer = node_p[i]->layer;

        switch(layer) {
                case 1:
                        return(dmf_fp(i, j));
                case 4:
                        return(dconsequent_dp(i, j));
                default:
```

```
                        exit1("Error in derivative_o_p!");
        }
}
/*parámetros consecuentes*/
double
dconsequent_dp(i, j)
int i, j;
{
        NODE_LIST_T *arg_p = node_p[i]->fan_in ;
        PARAMETER_LIST_T *para_p = node_p[i]->parameter;
        NODE_LIST_T *a_p;
        int k;
        double wn, x;

        for (a_p = arg_p; a_p->next != NULL; a_p = a_p->next);
        wn = a_p->content->value;
        if (j == In_n)
                return(wn);
        for (a_p = arg_p, k = 0; k < j; a_p = a_p->next, k++);
        x = a_p->content->value;
        return(x*wn);
        /*
        double a, b, c, wn, x1, x2;
        x1 = arg_p->content->value;
        x2 = arg_p->next->content->value;
        wn = arg_p->next->next->content->value;
        a = para_p->content;
        b = para_p->next->content;
        c = para_p->next->next->content;
        switch(j) {
                case 0:
                        return(x1*wn);
                        break;
                case 1:
                        return(x2*wn);
                        break;
                case 2:
                        return(wn);
                        break;
                default:
                        exit1("Error in dconsequent_dp!");
        }
        */
}

/*Selección de función de pertenencia*/
double
dmf_fp(i, j)
int i, j;
{
        double dmf_dp1(),dmf_dp2(),dmf_dp3(),dmf_dp4();
        switch (funper) {
                case 1:
```

```
                return(dmf_dp1(i, j));
        case 2:
                return(dmf_dp2(i, j));
        case 3:
                return(dmf_dp3(i, j));
        case 4:
                return(dmf_dp4(i, j));
        default:
                exit1("Error en funper, debe asignar un entero entre 1 y 4!");
    }
}
/*derivadas parciales bell_1*/
double
dmf_dp1(i, j)
int i, j;
{
    NODE_LIST_T *arg_p = node_p[i]->fan_in ;
    PARAMETER_LIST_T *para_p = node_p[i]->parameter;
    double c, a, b, x;
    double tmp1, tmp2;
    double denom;
    x = arg_p->content->value;
    a = para_p->content;
    b = para_p->next->content;
    c = para_p->next->next->content;
/*printf("a %10f.\t b %10f.\t c %10f.\n", a, b, c);*/
    if (a == 0)
        exit1("Error in dmf_dp1!");
    tmp1 = (x - c)/a;
    tmp2 = tmp1 == 0 ? 0 : pow(pow(tmp1, 2.0), b);
    denom = exp(tmp2);
    switch(j) {
        case 0: /* partial mf to partial a */
            return(2*b*tmp2/(a*denom));
        case 1: /* partial mf to partial b */
            if (tmp1 == 0)
                return(0.0);
            else
                return(-log(tmp1*tmp1)*tmp2/denom);
        case 2: /* partial mf to partial c */
            if (x == c)
                return(0.0);
            else
                return(2*b*tmp2/((x - c)*(denom)));
        default:
            exit1("Error in dmf_dp1!");
    }
}
/*derivadas parciales bell_2*/
double
dmf_dp2(i, j)
int i, j;
{
```

```
	NODE_LIST_T *arg_p = node_p[i]->fan_in ;
	PARAMETER_LIST_T *para_p = node_p[i]->parameter;
	double c, a, b, x;
	double tmp1, tmp2;
	double denom;

	x = arg_p->content->value;
	a = para_p->content;
	b = para_p->next->content;
	c = para_p->next->next->content;
	if (a == 0)
		exit1("Error in dmf_dp2!");
	tmp1 = (x - c)/a;
	tmp2 = tmp1 == 0 ? 0 : pow(pow(tmp1, 2.0), (b));
	denom = (1 + tmp2)*(1 + tmp2);
	switch(j) {
		case 0: /* partial mf to partial a */
			return(2*b*tmp2/(a*denom));
		case 1: /* partial mf to partial b */
			if (tmp1 == 0)
				return(0.0);
			else
				return(-log(tmp1*tmp1)*tmp2/denom);
		case 2: /* partial mf to partial c */
			if (x == c)
				return(0.0);
			else
				return(2*b*tmp2/((x - c)*(denom)));
		default:
			exit1("Error in dmf_dp2!");
	}
}
/*derivadas parciales trapezoidal*/
double
dmf_dp3(i, j)
int i, j;
{
	NODE_LIST_T *arg_p = node_p[i]->fan_in ;
	PARAMETER_LIST_T *para_p = node_p[i]->parameter;
	double c, a, b, x;
	double tmp1, tmp2;
	double denom;

	x = arg_p->content->value;
	a = para_p->content;
	b = para_p->next->content;
	c = para_p->next->next->content;
	if (a == 0)
		exit1("Error in dmf_dp3!");
	switch(j) {
		case 0: /* partial mf to partial a */
			if (x > c+b/2)
			return(-x+c+b/2);
```

```
        else if (x < c-b/2)
                        return(x-c+b/2);
                case 1: /* partial mf to partial b */
                        if ( (x < (c-b/2) ) || (x > (c + b/2) ) )
                                return(a/2);
                        else
                                return(0.0);
                case 2: /* partial mf to partial c */
                        if ( (x < (c-b/2) ) || (x > (c + b/2) ) )
                                return(a);
                        else
                                return(0.0);
                default:
                        exit1("Error in dmf_dp3!");
        }
}

double
dmf_dp4(i, j)
int i, j;
{
        NODE_LIST_T *arg_p = node_p[i]->fan_in ;
        PARAMETER_LIST_T *para_p = node_p[i]->parameter;
        double c, a, x;
        double tmp1, tmp2;
        double denom;

        x = arg_p->content->value;
        a = para_p->content;
        c = para_p->next->next->content;
        if (a == 0)
                exit1("Error in dmf_dp4!");
        switch(j) {
                case 0: /* partial mf to partial a */
                        if (x > c)
                                return(-x+c);
                        else if (x < c)
                                return(x-c);
                case 1: /* partial mf to partial b */
                                return(0.0);
                case 2: /* partial mf to partial c */
                        if ( (x < (c) ) || (x > (c) ) )
                                return(a);
                        else
                                return(0.0);
                default:
                        exit1("Error in dmf_dp4!");
        }
}
```

B.2.7. Rutina de estimación por mínimos cuadrados para aprendizaje fuera de línea (kalman.c)

```
#include "anfis.h"

static int k_p_n;     /* number of parameters to be identified
                              by kalman filter algorithm */

/* get the data to be used in kalman filter algorithm */
void
get_kalman_data(kalman_data, target)
double *kalman_data;
double target;
{
        /* 'first' is the index of the first node in layer 3 */
        int i, k, j = 0;
        int first = In_n + In_n*Mf_n + Rule_n;

        k_p_n = (In_n +1)*Rule_n;
        for (i = first; i < first + Rule_n; i++) {
                for (k = 0; k < In_n; k++)
                        kalman_data[j++] = (node_p[i]->value)*
                                                (node_p[k]->value);
                kalman_data[j++] = node_p[i]->value;
        }
        /*
        double x = node_p[0]->value;
        double y = node_p[1]->value;

        for (i = first; i < first + Rule_n; i++) {
                kalman_data[j++] = node_p[i]->value*x;
                kalman_data[j++] = node_p[i]->value*y;
                kalman_data[j++] = node_p[i]->value;
        }
        */
        kalman_data[j] = target;
}

/* sequential identification using kalman filter algorithm */
/* note that s[][] and p[] are static variables, such that
   their values are retained between every two function invocations */
void
kalman(zz, k, kalman_data, kalman_parameter)
int zz; /* (zz+1) is the epoch number */
int k; /* (k+1) is the times 'kalman' has been invoked in an epoch*/
double *kalman_data;
double *kalman_parameter;
{
        void s_mult_v();
        void s_mult_ma();
        double v_dot_v();
        void v_cross_v();
        void v_mult_m();
```

```
void m_mult_v();
void v_plus_v();
void ma_plus_ma();

int i, j;
double alpha = 1000000.0;
double denom, diff;
static double **s;
static double *p;
static double *x, y;
static double *tmp1, *tmp2;
static double **tmp_m;

/* initial values of s[][] and p[] if k = 0 */
if ((zz == 0) && (k == 0)) {
        s = (double **)create_matrix(k_p_n, k_p_n, sizeof(double));
        p = (double *)create_array(k_p_n, sizeof(double));
        x = (double *)create_array(k_p_n, sizeof(double));
        tmp1 = (double *)create_array(k_p_n, sizeof(double));
        tmp2 = (double *)create_array(k_p_n, sizeof(double));
        tmp_m = (double **)create_matrix(k_p_n, k_p_n, sizeof(double));
}
if (k == 0) {
        for (i = 0; i < k_p_n; i++)
                p[i] = 0;

        for (i = 0; i < k_p_n; i++)
                for (j = 0; j < k_p_n; j++)
                        if (i == j)
                                s[i][j] = alpha;
                        else
                                s[i][j] = 0;
}

for (i = 0; i < k_p_n; i++)
        x[i] = kalman_data[i];
y = kalman_data[k_p_n];

v_mult_m(x, s, tmp1);
denom = 1 + v_dot_v(tmp1, x);
m_mult_v(s, x, tmp1);
v_mult_m(x, s, tmp2);
v_cross_v(tmp1, tmp2, tmp_m);
s_mult_ma(-1/denom, tmp_m);
ma_plus_ma(s, tmp_m, s);

diff = y - v_dot_v(x, p);
m_mult_v(s, x, tmp1);
s_mult_v(diff, tmp1);
v_plus_v(p, tmp1, p);

for (i = 0; i < k_p_n; i++)
        kalman_parameter[i] = p[i];
```

```
}

/* matrix plus matrix */
void
ma_plus_ma(m1, m2, out)
double **m1, **m2, **out;
{
        int i, j;
        for (i = 0; i < k_p_n; i++)
                for (j = 0; j < k_p_n; j++)
                        out[i][j] = m1[i][j]+m2[i][j];
}

/* vector plus vector */
void
v_plus_v(v1, v2, out)
double v1[], v2[], out[];
{
        int i;
        for (i = 0; i < k_p_n; i++)
                out[i] = v1[i]+v2[i];
}

/* matrix multiplies vector */
void
m_mult_v(m, v, out)
double **m, v[], out[];
{
        int i, j;
        for (i = 0; i < k_p_n; i++) {
                out[i] = 0;
                for (j = 0; j < k_p_n; j++)
                        out[i] += m[i][j]*v[j];
        }
}

/* vector multiplies matrix */
void
v_mult_m(v, m, out)
double v[], **m, out[];
{
        int i, j;
        for (i = 0; i < k_p_n; i++) {
                out[i] = 0;
                for (j = 0; j < k_p_n; j++)
                        out[i] += v[j]*m[j][i];
        }
}

/* vector cross-products(?) vector */
void
v_cross_v(v1, v2, out)
double v1[], v2[], **out;
```

```
{
        int i, j;
        for (i = 0; i < k_p_n; i++)
                for (j = 0; j < k_p_n; j++)
                        out[i][j] = v1[i]*v2[j];
}

/* vector dot-products vector */
double
v_dot_v(v1, v2)
double v1[], v2[];
{
        int i;
        double out = 0;
        for (i = 0; i < k_p_n; i++)
                out += v1[i]*v2[i];
        return(out);
}

/* scalar multiplies matrix */
void
s_mult_ma(c, m)
double c;
double **m;
{
        int i, j;
        for (i = 0; i < k_p_n; i++)
                for (j = 0; j < k_p_n; j++)
                        m[i][j] = c*m[i][j];
}

/* scalar multiplies vector */
void
s_mult_v(c, v)
double c;
double v[];
{
        int i;
        for (i = 0; i < k_p_n; i++)
                v[i] = c*v[i];
}

/* put the parameters identified by kalman filter algorithm into GNN */
void
put_kalman_parameter(kalman_parameter)
double *kalman_parameter;
{
        /* 'first' is the index of the first node in layer 4 */
        int first = In_n + In_n*Mf_n + 2*Rule_n;
        PARAMETER_LIST_T *p;
        int i, j = 0;

        for (i = first; i < first + Rule_n; i++)
```

```
                for (p = node_p[i]->parameter; p != NULL; p = p->next)
                        p->content = kalman_parameter[j++];
}
```

B.2.8. Rutina de estimación por mínimos cuadrados para aprendizaje en línea (LSE_kalman.c)

```
#include "standard.h"

static double **S, **P, **a, **b, **a_t, **b_t;
static double **tmp1, **tmp2, **tmp3, **tmp4;
static double **tmp5, **tmp6, **tmp7;

/* create necessary matrices */
/* this function has to be invoked once and only  once
   in a program which invokes LSE_kalman */
/* an example can be found in ~/LSE/kalman/main.c */

void
initialize_kalman(in_n, out_n)
int in_n, out_n;
{
        S = (double **)create_matrix(in_n, in_n, sizeof(double));
        P = (double **)create_matrix(in_n, out_n, sizeof(double));
        a = (double **)create_matrix(in_n, 1, sizeof(double));
        b = (double **)create_matrix(out_n, 1, sizeof(double));
        a_t = (double **)create_matrix(1, in_n, sizeof(double));
        b_t = (double **)create_matrix(1, out_n, sizeof(double));

        tmp1 = (double **)create_matrix(in_n, 1, sizeof(double));
        tmp2 = (double **)create_matrix(1, 1, sizeof(double));
        tmp3 = (double **)create_matrix(1, in_n, sizeof(double));
        tmp4 = (double **)create_matrix(in_n, in_n, sizeof(double));
        tmp5 = (double **)create_matrix(1, out_n, sizeof(double));
        tmp6 = (double **)create_matrix(in_n, out_n, sizeof(double));
        tmp7 = (double **)create_matrix(in_n, out_n, sizeof(double));
}

/* reset values of S[][] and P[][] */
void
reset_kalman(in_n, out_n)
int in_n, out_n;
{
        int i, j;
        double alpha = 1e6;

        for (i = 0; i < in_n; i++)
                for (j = 0; j < out_n; j++)
                        P[i][j] = 0;

        for (i = 0; i < in_n; i++)
                for (j = 0; j < in_n; j++)
```

```
                        if (i == j)
                                S[i][j] = alpha;
                        else
                                S[i][j] = 0;
}

/* sequential identification using kalman filter algorithm */
/* kalman_data_pair is an array composed of desired input and output */
void
kalmanlse(in_n, out_n, kalman_data_pair, kalman_parameter)
int in_n, out_n;
double *kalman_data_pair;
double **kalman_parameter;
{
        int i, j;
        double denom;

        for (i = 0; i < in_n; i++)
                a[i][0] = kalman_data_pair[i];
        for (i = 0; i < out_n; i++)
                b[i][0] = kalman_data_pair[i+in_n];

        m_transpose(a, in_n, 1, a_t);
        m_transpose(b, out_n, 1, b_t);

        /* recursive formulas for S, covariance matrix */
        m_times_m(S, a, in_n, in_n, 1, tmp1);
        m_times_m(a_t, tmp1, 1, in_n, 1, tmp2);
        denom = 1 + tmp2[0][0];
        m_times_m(a_t, S, 1, in_n, in_n, tmp3);
        m_times_m(tmp1, tmp3, in_n, 1, in_n, tmp4);
        s_times_m(1/denom, tmp4, in_n, in_n, tmp4);
        m_minus_m(S, tmp4, in_n, in_n, S);

        /* recursive formulas for P, the estimated parameter matrix */
        m_times_m(a_t, P, 1, in_n, out_n, tmp5);
        m_minus_m(b_t, tmp5, 1, out_n, tmp5);
        m_times_m(a, tmp5, in_n, 1, out_n, tmp6);
        m_times_m(S, tmp6, in_n, in_n, out_n, tmp7);
        m_plus_m(P, tmp7, in_n, out_n, P);

        for (i = 0; i < in_n; i++)
                for (j = 0; j < out_n; j++)
                        kalman_parameter[i][j] = P[i][j];
}

/* same as kalman(), but with forgetting factor lambda */
void
new_kalman(in_n, out_n, kalman_data_pair, kalman_parameter, lambda)
int in_n, out_n;
double *kalman_data_pair;
double **kalman_parameter;
double lambda;
```

```
{
        int i, j;
        double denom;
        for (i = 0; i < in_n; i++)
                a[i][0] = kalman_data_pair[i];
        for (i = 0; i < out_n; i++)
                b[i][0] = kalman_data_pair[i+in_n];
        m_transpose(a, in_n, 1, a_t);
        m_transpose(b, out_n, 1, b_t);

        /* recursive formulas for S, covariance matrix */
        m_times_m(S, a, in_n, in_n, 1, tmp1);
        m_times_m(a_t, tmp1, 1, in_n, 1, tmp2);
        denom = lambda + tmp2[0][0];
        m_times_m(a_t, S, 1, in_n, in_n, tmp3);
        m_times_m(tmp1, tmp3, in_n, 1, in_n, tmp4);
        s_times_m(1/denom, tmp4, in_n, in_n, tmp4);
        m_minus_m(S, tmp4, in_n, in_n, S);
        s_times_m(1/lambda, S, in_n, in_n, S);
        /* recursive formulas for P, the estimated parameter matrix */
        m_times_m(a_t, P, 1, in_n, out_n, tmp5);
        m_minus_m(b_t, tmp5, 1, out_n, tmp5);
        m_times_m(a, tmp5, in_n, 1, out_n, tmp6);
        m_times_m(S, tmp6, in_n, in_n, out_n, tmp7);
        m_plus_m(P, tmp7, in_n, out_n, P);
        for (i = 0; i < in_n; i++)
                for (j = 0; j < out_n; j++)
                        kalman_parameter[i][j] = P[i][j];
}

/* find the least square solution of AX = B,
   using kalman filter algorithm */
/* return the RMSE (root mean squares error) */
/* A: data_n X in_n
   X: in_n X out_n
   B: data_n X out_n */
double
LSE_kalman(A, B, X, data_n, in_n, out_n, lambda)
double **A, **B, **X;
int data_n, in_n, out_n;
double lambda;
{
        double *kalman_data_pair;
        double **AX, **AX_minus_B;
        double root_square_error;
        int i;
        static int initialization;

        void get_kalman_data();
        void initialize_kalman();
        void reset_kalman();
        void kalmanlse(), new_kalman();
/*printf("entre \n");*/
```

```
        kalman_data_pair = (double *)calloc(in_n+out_n, sizeof(double));
        AX = (double **)create_matrix(data_n, out_n, sizeof(double));
        AX_minus_B = (double **)
                create_matrix(data_n, out_n, sizeof(double));
/*printf("saque Ax-B \n");*/
        /* initialize only once, at the first call of this function */
        if (initialization != 7527474) {
/*printf("inicialice \n");*/
                initialize_kalman(in_n, out_n);
                initialization = 7527474;
        }
        reset_kalman(in_n, out_n);
        for (i = 0; i < data_n; i++) {
                get_kalman_data_lse(A, B, i, kalman_data_pair, in_n, out_n);
                new_kalman(in_n, out_n, kalman_data_pair, X, lambda);
        }
        m_times_m(A, X, data_n, in_n, out_n, AX);
        m_minus_m(AX, B, data_n, out_n, AX_minus_B);
        root_square_error = m_norm(AX_minus_B, data_n, out_n);
        free_array(kalman_data_pair);
        free_matrix(AX, data_n);
        free_matrix(AX_minus_B, data_n);

        return(root_square_error/sqrt((double)(data_n)));
}

/* get the k-th data entry to be used in kalman filter algorithm */
void
get_kalman_data_lse(A, B, k, kalman_data_pair, in_n, out_n)
double **A, **B;
double *kalman_data_pair;
int k, in_n, out_n;
{
        int i;
        for (i = 0; i < in_n ; i++)
                kalman_data_pair[i] = A[k][i];
        for (i = 0; i < out_n ; i++)
                kalman_data_pair[i+in_n] = B[k][i];

}
```

B.2.9. Rutina de librerías de uso general (lib.c)

```
#include "standard.h"

/* Usage of create_cubic, create_matrix and create_array:
  1. Declaration in the calling program:
        char ***create_cubic();
        char **create_matrix();
        char *create_array();

  2. Examples of invocations:
```

```
        my_cubic = (float ***)create_cubic(row, col, height, sizeof(float));
        my_matrix = (int **)create_matrix(row, col, sizeof(int));
        my_array = (double *)create_array(row+col, sizeof(double));
*/

char ***
create_cubic(row, col, height, element_size)
int row, col, height, element_size;
{
        char ***cubic;
        int i, j;

        cubic = (char ***)malloc(sizeof(char **)*row);
        if (cubic == NULL) {
                printf("Error in create_cubic!\n");
                exit(1);
        }
        for (i = 0; i < row; i++) {
                cubic[i] = (char **)malloc(sizeof(char *)*col);
                if (cubic[i] == NULL) {
                        printf("Error in create_cubic!\n");
                        exit(1);
                }
                for (j = 0; j < col; j++) {
                        cubic[i][j] = (char *)malloc(element_size*height);
                        if (cubic[i][j] == NULL) {
                                printf("Error in create_cubic!\n");
                                exit(1);
                        }
                }
        }
        return(cubic);
}

char **
create_matrix(row_n, col_n, element_size)
int row_n, col_n, element_size;
{
        char **matrix;
        int i;

        matrix = (char **) malloc(sizeof(char *)*row_n);
        if (matrix == NULL) {
                printf("Error in create_matrix!\n");
                exit(1);
        }
        for (i = 0; i < row_n; i++) {
                matrix[i] = (char *) malloc(element_size*col_n);
                if (matrix[i] == NULL) {
                        printf("Error in create_matrix!\n");
                        exit(1);
                }
        }
```

```
	return(matrix);
}

char *
create_array(array_size, element_size)
int array_size, element_size;
{
	char *array;

	array = (char *)malloc(array_size*element_size);
	if (array == NULL) {
		printf("Error in create_array!\n");
		exit(1);
	}
	return(array);
}

/* an friendly interface to fopen() */
FILE *
open_file(file, mode)
char *file, *mode;
{
	FILE *fp, *fopen();

	if ((fp = fopen(file, mode)) == NULL){
		printf("Cannot open '%s'.\n", file);
		exit(1);
	}
	return(fp);
}

void
free_cubic(cubic, row, col)
char ***cubic;
int row, col;
{
	int i, j;

	for (i = 0; i < row; i++) {
		for (j = 0; j < col; j++)
			if (cubic[i][j] == NULL) {
				printf("free_cubic: already free!\n");
				exit(1);
			} else {
				free(cubic[i][j]);
				cubic[i][j] = NULL;
			}

		if (cubic[i] == NULL) {
			printf("free_cubic: already free!\n");
			exit(1);
		} else {
			free(cubic[i]);
```

```
                        cubic[i] = NULL;
                }
        }
        if (cubic == NULL) {
                printf("free_cubic: already free!\n");
                exit(1);
        } else {
                free(cubic);
                cubic = NULL;
        }
}

void
free_matrix(matrix, row_n)
char **matrix;
int row_n;
{
        int i;

        for (i = 0; i < row_n; i++)
                if (matrix[i] == NULL) {
                        printf("free_matrix: row %d is already free!\n", i);
                        exit(1);
                } else {
                        free(matrix[i]);
                        matrix[i] = NULL;
                }

        if (matrix == NULL) {
                printf("free_matrix: given matrix is already free!\n");
                exit(1);
        } else {
                free(matrix);
                matrix = NULL;
        }
}

void
free_array(array)
char *array;
{
        if (array == NULL) {
                printf("free_array: already free!\n");
                exit(1);
        } else {
                free(array);
                array = NULL;
        }
}

void
print_matrix(matrix, row_n, col_n)
double **matrix;
```

```
int row_n, col_n;
{
        int i, j;
        for (i = 0; i < row_n; i++) {
                for (j = 0; j < col_n; j++)
                        printf("%lf ", matrix[i][j]);
                printf("\n");
        }
}

void
print_array(array, size)
double *array;
int size;
{
        int i;
        for (i = 0; i < size; i++)
                printf("%lf ", array[i]);
        printf("\n");
}


/* write array to file 'file_name', using matlab format */
/* the matlab variable name is the baseanme of 'file_name' */
void
write_array(array, size, errmatrix)
double *array;
int size;
double errmatrix[size];
{
        int i;
        double tmp;

        for (i = 0; i < size; i++) {
    tmp = array[i];
    errmatrix[i] = tmp;
  }
}

void
exit1(s)
char *s;
{
        printf("%s\n", s);
        exit(1);
}

/* matrix operations */

/* matrix plus matrix */
void
m_plus_m(m1, m2, row, col, out)
double **m1, **m2, **out;
```

```
int row, col;
{
        int i, j;
        for (i = 0; i < row; i++)
                for (j = 0; j < col; j++)
                        out[i][j] = m1[i][j] + m2[i][j];
}

/* matrix minus matrix */
void
m_minus_m(m1, m2, row, col, out)
double **m1, **m2, **out;
int row, col;
{
        int i, j;
        for (i = 0; i < row; i++)
                for (j = 0; j < col; j++)
                        out[i][j] = m1[i][j] - m2[i][j];
}

/* matrix times matrix */
void
m_times_m(m1, m2, row1, col1, col2, out)
double **m1, **m2, **out;
int row1, col1, col2;
{
        int i, j, k;
        for (i = 0; i < row1; i++)
                for (j = 0; j < col2; j++) {
                        out[i][j] = 0;
                        for (k = 0; k < col1; k++)
                                out[i][j] += m1[i][k]* m2[k][j];
                }
}

/* scalar times matrix */
void
s_times_m(c, m, row, col, out)
double c;
double **m, **out;
int row, col;
{
        int i, j;
        for (i = 0; i < row; i++)
                for (j = 0; j < col; j++)
                        out[i][j] = c*m[i][j];
}

/* matrix transpose */
void
m_transpose(m, row, col, m_t)
double **m, **m_t;
int row, col;
```

```
{
        int i, j;
        for (i = 0; i < row; i++)
                for (j = 0; j < col; j++)
                        m_t[j][i] = m[i][j];
}

/* matrix L-2 norm */
double
m_norm(m, row, col)
double **m;
int row, col;
{
        int i, j;
        double total = 0;

        for (i = 0; i < col; i++)
                for (j = 0; j < row; j++)
                        total += m[i][j]*m[i][j];
        return(sqrt(total));
}
```

B.2.10. Rutinas necesarias para la actualización de parámetros (new_para.c)

```
#include "anfis.h"


/* update parameters in k-th layer*/
/* if k == -1, update all parameters */
void
update_parameter(k, step_size)
int k;
double step_size;
{
        int i;
        double length;
        PARAMETER_LIST_T *p;
        double gradient_vector_length();

        length = gradient_vector_length(k);
        if (length == 0) {
                printf("gradient vector length == 0!\n");
                return;
        }
        for (i = 0; i < Node_n; i++) {
                if (node_p[i]->parameter == NULL)
                        continue;
                if ((k != -1) && (node_p[i]->layer != k))
                        continue;
                for (p = node_p[i]->parameter; p != NULL; p = p->next)
                        p->content -= step_size*(p->de_dp)/length;
        }
```

```
}

/* calculate length of gradient vector of parameters in k-th layer */
/* if k == -1, calculate length of gradient vector of all parameters */
double
gradient_vector_length(k)
int k;
{
        int i;
        double total = 0;
        PARAMETER_LIST_T *p;

        for (i = 0; i < Node_n; i++) {
                if (node_p[i]->parameter == NULL)
                        continue;
                if ((k != -1) && (node_p[i]->layer != k))
                        continue;
                for (p = node_p[i]->parameter; p != NULL; p = p->next)
                        total += pow(p->de_dp, 2.0);
        }
        return(sqrt(total));
}
```

B.2.11. Rutinas para la actualización del paso (stepsize.c)

```
#include "anfis.h"

/* update step size */
void
update_step_size(error_array, i, step_size_p, decrease_rate, increase_rate)
double *error_array;
int i;
double *step_size_p;
double decrease_rate, increase_rate;
{
        static int last_decrease_ss, last_increase_ss;
        int check_increase_ss();
        int check_decrease_ss();

        if (i == 0)
                last_decrease_ss = last_increase_ss = 0;

        if (check_decrease_ss(error_array, last_decrease_ss, i)) {
                *step_size_p *= decrease_rate;
                printf("Ss decrease to %f after epoch %d.\n", *step_size_p, i+1);
                last_decrease_ss = i;
                return;
        }

        if (check_increase_ss(error_array, last_increase_ss, i)) {
                *step_size_p *= increase_rate;
                printf("Ss increase to %f after epoch %d.\n", *step_size_p, i+1);
```

```
                last_increase_ss = i;
                return;
        }
}

/* return 1 if the step size needs to be increased, 0 otherwise */
int
check_increase_ss(error_array, last_change, current)
double *error_array;
int last_change, current;
{
        if (current - last_change < 4)
                return(0);
        if ((error_array[current]     < error_array[current - 1]) &&
           (error_array[current - 1] < error_array[current - 2]) &&
           (error_array[current - 2] < error_array[current - 3]) &&
           (error_array[current - 3] < error_array[current - 4]))
                return(1);
        return(0);
}

/* return 1 if the step size needs to be decreased, 0 otherwise */
int
check_decrease_ss(error_array, last_change, current)
double *error_array;
int last_change, current;
{
        if (current - last_change < 4)
                return(0);
        if ((error_array[current]     < error_array[current - 1]) &&
           (error_array[current - 1] > error_array[current - 2]) &&
           (error_array[current - 2] < error_array[current - 3]) &&
           (error_array[current - 3] > error_array[current - 4]))
                return(1);
        return(0);
}
```

B.2.12. Rutinas para cálculos del error de los datos de entrenamiento (trn_err.c)

```
#include "anfis.h"

/* error_index[0] = RMSE --> root mean square error
   error_index[1] = NDEI --> non-dimensional error index
   error_index[2] = ARV  --> average relative variance
   error_index[3] = APE  --> average percentage error */
void
training_error_measure(data_matrix, anfis_output, data_n, error_index)
double **data_matrix, *anfis_output;
int data_n;
double error_index[];
{
        double RMSE, NDEI, ARV, APE;
```

```
	int i;
	static double target_total, abs_target_total, target_mean;
	static double target_VAR, target_STD, tmp;
	static int initialization;
	double total_abs_error, total_squared_error;
	double diff;

	/* initialization for the first call of this function */
	if (initialization != 7527474) {
		initialization = 7527474;
		target_total = 0;
		abs_target_total = 0;
		target_VAR = 0;
		target_STD = 0;
		for (i = 0; i < data_n; i++) {
			target_total += data_matrix[i][In_n];
			abs_target_total += ABS(data_matrix[i][In_n]);
		}
		target_mean = target_total/data_n;
		tmp = 0;
		for (i = 0; i < data_n; i++)
			tmp += (data_matrix[i][In_n] - target_mean)*
				(data_matrix[i][In_n] - target_mean);
		target_VAR = tmp/data_n;
		target_STD = sqrt(target_VAR);
	}

	total_abs_error = 0;
	total_squared_error = 0;
	tmp = 0;
	for (i = 0; i < data_n; i++) {
		diff = data_matrix[i][In_n] - anfis_output[i];
		total_abs_error += ABS(diff);
		total_squared_error += (diff*diff);
		tmp += (ABS(diff)/data_matrix[i][In_n]);
	}
	RMSE = sqrt(total_squared_error/data_n);
	NDEI = RMSE/target_STD;
	ARV = NDEI*NDEI;
	APE = total_abs_error/abs_target_total;
	error_index[0] = RMSE;
	error_index[1] = NDEI;
	error_index[2] = ARV;
	error_index[3] = APE;
	/*
	error_index[3] = tmp/data_n;
	*/
}
```

B.2.13. Rutinas para cálculos del error de los datos de chequeo (chk_err.c)

```
#include "anfis.h"

/* error_index[0] = RMSE --> root mean square error
   error_index[1] = NDEI --> non-dimensional error index
   error_index[2] = ARV  --> average relative variance
   error_index[3] = APE  --> average percentage error */
void
checking_error_measure(data_matrix, anfis_output, data_n, error_index)
double **data_matrix, *anfis_output;
int data_n;
double error_index[];
{
        double RMSE, NDEI, ARV, APE;
        int i;
        static double target_total, abs_target_total, target_mean;
        static double target_VAR, target_STD, tmp;
        static int initialization;
        double total_abs_error, total_squared_error;
        double diff;

        /* initialization for the first call of this function */
        if (initialization != 7527474) {
                initialization = 7527474;
                target_total = 0;
                abs_target_total = 0;
                target_VAR = 0;
                target_STD = 0;
                for (i = 0; i < data_n; i++) {
                        target_total += data_matrix[i][In_n];
                        abs_target_total += ABS(data_matrix[i][In_n]);
                }
                target_mean = target_total/data_n;
                tmp = 0;
                for (i = 0; i < data_n; i++)
                        tmp += (data_matrix[i][In_n] - target_mean)*
                                (data_matrix[i][In_n] - target_mean);
                target_VAR = tmp/data_n;
                target_STD = sqrt(target_VAR);
        }

        total_abs_error = 0;
        total_squared_error = 0;
        tmp = 0;
        for (i = 0; i < data_n; i++) {
                diff = data_matrix[i][In_n] - anfis_output[i];
                total_abs_error += ABS(diff);
                total_squared_error += (diff*diff);
                tmp += (ABS(diff)/data_matrix[i][In_n]);
        }
        RMSE = sqrt(total_squared_error/data_n);
        NDEI = RMSE/target_STD;
```

```
	ARV = NDEI*NDEI;
	APE = total_abs_error/abs_target_total;
	error_index[0] = RMSE;
	error_index[1] = NDEI;
	error_index[2] = ARV;
	error_index[3] = APE;
	/*
	error_index[3] = tmp/data_n;
	*/
}

void
epoch_checking_error(data_matrix, data_n, error_index)
double **data_matrix;
int data_n;
double error_index[];
{
	int j;
	static int initialization;
	static double *anfis_output;

	void put_input_data();
	void calculate_output();
	void checking_error_measure();
	if (data_n == 0) {
		printf("No data in given data matrix!\n");
		exit(1);
	}
	/* allocate array when first called */
	if (initialization != 7527474) {
		initialization = 7527474;
		anfis_output = (double *)calloc(data_n, sizeof(double));
	}
	for (j = 0; j < data_n; j++) {
		put_input_data(j, data_matrix);
		calculate_output(In_n, Node_n - 1);
		anfis_output[j] = node_p[Node_n - 1]->value;
	}

	checking_error_measure(data_matrix, anfis_output, data_n, error_index);
}
```

B.2.14. Rutina principal para la validación del modelo (checka.c)

```
#include "anfis.h"
#include "misc_def.h"

NODE_T **node_p;
int In_n, Mf_n, Rule_n, Node_n, funper;
void
  checka(In_n, Mf_n, checking_data_n, funper, abc, pqr, checking_data_matrix, anfis_output)
	int checking_data_n;
```

```
        double **abc, **pqr;
        double **checking_data_matrix;
        double *anfis_output;
{
        int i, j, k;
        double **checking_data_pointer;
        double desired_output;
        Rule_n = (int)pow((double)Mf_n, (double)In_n);
        Node_n = In_n + In_n*Mf_n + 3*Rule_n + 1;

        node_p = (NODE_T **)create_array(Node_n, sizeof(NODE_T *));
        checking_data_pointer = (double **)create_matrix
                        (checking_data_n, In_n + 1, sizeof(double));

        get_config();
        build_anfis();
        get_parameter(abc,pqr);
        get_data(checking_data_matrix, checking_data_n, checking_data_pointer);
        printf("Data_n\t Salida ANFIS \t Salida deseada\n");
        printf("------\t ------------ \t --------------\n");
        for (j = 0; j < checking_data_n; j++) {
                put_input_data(j, checking_data_pointer);
                calculate_output(In_n, Node_n - 1);
        anfis_output[j] = node_p[Node_n - 1]->value;
desired_output = checking_data_pointer[j][In_n];
printf("%3d \t %lf \t %lf\n", j+1,anfis_output[j],desired_output);
        }
}
```

B.2.15. Rutina principal para el aprendizaje en línea (hyonline.c)

```
#include "anfis.h"
#include "misc_def.h"

NODE_T **node_p;
int In_n, Mf_n, Rule_n, Node_n, funper;
void
  hyonline(In_n, Mf_n, training_data_n, funper, step_size, abc, pqr, training_data_matrix,
trn_evol_error)
    int training_data_n;
    double step_size;
    double **abc, **pqr;
    double **training_data_matrix;
        double **trn_evol_error;
{
        int i, j, k, debug;
        int parameter_n, first;
        double increase_rate, decrease_rate;/*agregado*/
        double min_RMSE = pow(2.0, 31.0) - 1;
        double *kalman_data, *kalman_parameter;
        double *trn_rmse_error;
        double de_dout, target, lambda;
```

```
        double **layer_1_to_3_output, **A,**B, **X;
        double **training_data_pointer;
        double *parameter_array, *step_size_array, *anfis_output;
        double *trn_error;
 double trn_evol[training_data_n], step_evol[training_data_n];
 decrease_rate = 0.9;
 increase_rate = 1.1;
 lambda=0.99;
debug=0;
        Rule_n = (int)pow((double)Mf_n, (double)In_n);
        Node_n = In_n + In_n*Mf_n + 3*Rule_n + 1;
        first = In_n + In_n*Mf_n + Rule_n;
        /* allocate matrices */
        node_p = (NODE_T **)create_array(Node_n, sizeof(NODE_T *));
        training_data_pointer = (double **)create_matrix
                        (training_data_n, In_n + 1, sizeof(double));
        layer_1_to_3_output = (double **)create_matrix
                (training_data_n, In_n*Mf_n + 2*Rule_n, sizeof(double));

        A = (double **)create_matrix(training_data_n, Rule_n, sizeof(double));
        B = (double **)create_matrix(training_data_n, 1, sizeof(double));
        X = (double **)create_matrix(Rule_n, 1, sizeof(double));

        trn_rmse_error = (double *)calloc(training_data_n, sizeof(double));
        kalman_data = (double *)
                    calloc((In_n + 1)*Rule_n + 1, sizeof(double));
        kalman_parameter = (double *)calloc
                            ((In_n + 1)*Rule_n, sizeof(double));
        step_size_array = (double *)calloc(training_data_n, sizeof(double));
        anfis_output = (double *)calloc(training_data_n, sizeof(double));
        trn_error = (double *)calloc(4, sizeof(double));

        get_config();
        build_anfis();
        parameter_n = set_parameter_mode();
        parameter_array = (double *)calloc(parameter_n, sizeof(double));
        get_parameter(abc,pqr);

 get_data(training_data_matrix, training_data_n, training_data_pointer);
        printf("epochs \t trn error \t step error\n");
        printf("------ \t --------- \t ----------\n");
        for (i = 0; i < training_data_n; i++) {
                step_size_array[i] = step_size;
                for (j = 0; j < training_data_n; j++) {
                        put_input_data(j, training_data_pointer);
                        /* get node outputs from layer 1 to layer 3 */
                        calculate_output(In_n, In_n + In_n*Mf_n + 2*Rule_n - 1);
                        /* put outputs of layer 1 to 3
                           into layer_1_to_3_output */
                        for (k = 0; k < Mf_n*In_n + 2*Rule_n; k++)
                                layer_1_to_3_output[j][k]
                                        = node_p[k + In_n]->value;
```

```
                for (k = first; k < first + Rule_n; k++)
                        A[j][k-first] = (node_p[k]->value);
                        B[j][0] = training_data_matrix[j][In_n];

                }
                LSE_kalman(A, B, X, training_data_n, Rule_n, 1, lambda);
                for (j = 0; j < Rule_n; j++)
                kalman_parameter[j]=X[j][0];
                clear_de_dp();
                for (j = 0; j < training_data_n; j++) {
                        put_input_data(j, training_data_pointer);
                        /* get output of layer 1 to 3
                           from layer_1_to_3_output */
                        for (k = 0; k < Mf_n*In_n + 2*Rule_n; k++){
                                node_p[k + In_n]->value
                                        = layer_1_to_3_output[j][k];
        }
                        /* calculate outputs of layer 4 and 5 */
                        calculate_output(In_n + In_n*Mf_n + 2*Rule_n, Node_n - 1);
                        anfis_output[j] = node_p[Node_n - 1]->value;
                        target = training_data_matrix[j][In_n];
                        de_dout = -2*(target - node_p[Node_n-1]->value);
                        calculate_de_do(de_dout);
                        update_de_dp();
                }
                training_error_measure(training_data_pointer, anfis_output,
                        training_data_n, trn_error);
                trn_rmse_error[i] = trn_error[0];
                        printf("%4d \t %lf\n", i+1,
                                trn_error[ERROR_TYPE]);
                if (trn_rmse_error[i] < min_RMSE) {
                        min_RMSE = trn_rmse_error[i];
                        record_parameter(parameter_array);
                }
                /* update parameters in 1st layer */
                update_parameter(1, step_size);

                update_step_size(trn_rmse_error, i, &step_size,
                        decrease_rate, increase_rate);
        }
        printf("Minimal training RMSE = %lf\n", min_RMSE);
        restore_parameter(parameter_array);
        write_parameter(abc,pqr);
        write_array(trn_rmse_error, training_data_n, trn_evol);
        write_array(step_size_array, training_data_n, step_evol);
  for (j = 0; j < training_data_n; j++) {
  trn_evol_error[j][0]=j+1;
    trn_evol_error[j][1]=trn_rmse_error[j];
      trn_evol_error[j][2]=step_size_array[j];
  }
}
```

B.2.16. Rutina paso hacia atrás (backward.c)

```
#include "anfis.h"

/* calculate de/do of each node */
/* for mean square error,
   de_dout = -2*(target - node_p[Node_n-1]->value) */
void
calculate_de_do(de_dout)
double de_dout;
{
        int i, j;
        NODE_LIST_T *p;
        double tmp1, tmp2;
        double derivative_o_o();
        double de_do;

        node_p[Node_n - 1]->de_do = de_dout;
        for (i = Node_n - 2; i >= In_n; i--) {
                de_do = 0;
                for (p = node_p[i]->fan_out; p != NULL; p = p->next) {
                        j = p->content->index;
                        tmp1 = p->content->de_do;
                        tmp2 = derivative_o_o(i, j);
                        de_do += tmp1*tmp2;
                }
                node_p[i]->de_do = de_do;
        }
}

/* calculate do(i)/do(j), where i and j are node indice */
double
derivative_o_o(i, j)
int i, j;
{
        double do2_do1(), do3_do2(), do4_do3(), do5_do4();
        int layer = node_p[i]->layer;
        switch (layer) {
                case 0:
                        exit1("Error in derivative_o_o!");
                case 1:
                        return(do2_do1(i, j));
                case 2:
                        return(do3_do2(i, j));
                case 3:
                        return(do4_do3(i, j));
                case 4:
                        return(do5_do4(i, j));
                default:
                        exit1("Error in derivative_o_o!");
        }
}
```

```
/* calculate do(j)/do(i), where node i is in layer 4, node j layer 5 */
double
do5_do4(i, j)
int i, j;
{
        return(1.0);
}

/* calculate do(j)/do(i), where node i is in layer 3, node j layer 4 */
double
do4_do3(i, j)
int i, j;
{
        NODE_LIST_T *arg_p = node_p[j]->fan_in;
        PARAMETER_LIST_T *para_p = node_p[j]->parameter;
        NODE_LIST_T *a_p;
        PARAMETER_LIST_T *p_p;
        int k;
        double x, a, total = 0;

        if ((j - i) != Rule_n)
                exit1("Error in do4_do3()!");
        a_p = arg_p;
        p_p = para_p;
        for (k = 0; k < In_n + 1; k++) {
                x = a_p->content->value;
                a = p_p->content;
                if (k == In_n)
                        break;
                total += x*a;
                a_p = a_p->next;
                p_p = p_p->next;
        }
        return(total + a);
        /*
        double a, b, c, x1, x2;
        x1 = arg_p->content->value;
        x2 = arg_p->next->content->value;
        a = para_p->content;
        b = para_p->next->content;
        c = para_p->next->next->content;
        return(a*x1 + b*x2 + c);
        */
}

/* calculate do(j)/do(i), where node i is in layer 2, node j layer 3 */
double
do3_do2(i, j)
int i, j;
{
        NODE_LIST_T *arg_p = node_p[j]->fan_in;
        NODE_LIST_T *p;
        double total = 0;
```

```
        for (p = arg_p; p != NULL; p = p->next)
                total += p->content->value;
        if (total == 0 )
                total = 0.00000001;
        if ((j - i) == Rule_n)
                return((total - node_p[i]->value)/(total*total));
        else
                return(-node_p[j - Rule_n]->value/(total*total));
}

/* calculate do(j)/do(i), where node i is in layer 1, node j layer 2 */
double
do2_do1(i, j)
int i, j;
{
        NODE_LIST_T *arg_p = node_p[j]->fan_in;
        NODE_LIST_T *p;
  double den;
        double product = 1.0;

  den = (node_p[i]->value) == 0 ? 0.0000001 : (node_p[i]->value);
        return((node_p[j]->value)/den);
}
```

B.2.17. Rutina principal para el aprendizaje fuera de línea (hybrid.c)

```
#include "anfis.h"
#include "misc_def.h"

NODE_T **node_p;
int In_n, Mf_n, Rule_n, Node_n, funper;
void
  hybrid(In_n, Mf_n, training_data_n, checking_data_n, funper, epoch_n, err_req, step_size, abc, pqr,
training_data_matrix, checking_data_matrix, trn_evol_error)
    int training_data_n, checking_data_n, epoch_n;
    double step_size, err_req;
    double **abc, **pqr;
    double **training_data_matrix, **checking_data_matrix;
        double **trn_evol_error;
{
        int i, j, k, debug;
        int parameter_n;/*agregado */
        double increase_rate, decrease_rate;/*agregado*/
        double min_RMSE = pow(2.0, 31.0) - 1;
        double *kalman_data, *kalman_parameter;
        double *trn_rmse_error,*chk_rmse_error;
        double de_dout, target;
        double **layer_1_to_3_output;
        double **training_data_pointer, **checking_data_pointer;
        double *parameter_array, *step_size_array, *anfis_output;
        double *trn_error, *chk_error;
```

```
  double trn_evol[epoch_n], chk_evol[epoch_n], step_evol[epoch_n];
  decrease_rate = 0.9;
  increase_rate = 1.1;
  debug=0;

        Rule_n = (int)pow((double)Mf_n, (double)In_n);
        Node_n = In_n + In_n*Mf_n + 3*Rule_n + 1;

        /* allocate matrices */
        node_p = (NODE_T **)create_array(Node_n, sizeof(NODE_T *));
        training_data_pointer = (double **)create_matrix
                        (training_data_n, In_n + 1, sizeof(double));
        checking_data_pointer = (double **)create_matrix
                        (checking_data_n, In_n + 1, sizeof(double));
        layer_1_to_3_output = (double **)create_matrix
                (training_data_n, In_n*Mf_n + 2*Rule_n, sizeof(double));
        trn_rmse_error = (double *)calloc(epoch_n, sizeof(double));
        chk_rmse_error = (double *)calloc(epoch_n, sizeof(double));
        kalman_data = (double *)
                    calloc((In_n + 1)*Rule_n + 1, sizeof(double));
        kalman_parameter = (double *)calloc
                            ((In_n + 1)*Rule_n, sizeof(double));
        step_size_array = (double *)calloc(epoch_n, sizeof(double));
        anfis_output = (double *)calloc(training_data_n, sizeof(double));
        trn_error = (double *)calloc(4, sizeof(double));
        chk_error = (double *)calloc(4, sizeof(double));

        get_config();
        build_anfis();
        parameter_n = set_parameter_mode();
        parameter_array = (double *)calloc(parameter_n, sizeof(double));
        get_parameter(abc,pqr);
printf("abc \n");
for (i=0; i<In_n*Mf_n;i++){
printf("%d\t %lf \t %lf \t %lf \n", i+1,abc[i][0], abc[i][1], abc[i][2]);
}

printf("pqr \n");
for (i=0; i<Rule_n-1;i++){
printf(" %lf \t %lf \t %lf \t %lf \n", pqr[i][0], pqr[i][1], pqr[i][2], pqr[i][3]);
}

      get_data(training_data_matrix, training_data_n, training_data_pointer);
        get_data(checking_data_matrix, checking_data_n, checking_data_pointer);
        if (debug != 0) debug_anfis();
        printf("epochs \t trn error \t chk error\n");
        printf("------ \t --------- \t ---------\n");
        for (i = 0; ((i < epoch_n) && (err_req <= min_RMSE)); i++) {



                step_size_array[i] = step_size;
                for (j = 0; j < training_data_n; j++) {
```

```
                put_input_data(j, training_data_pointer);
                /* get node outputs from layer 1 to layer 3 */
                calculate_output(In_n, In_n + In_n*Mf_n + 2*Rule_n - 1);
                /* put outputs of layer 1 to 3
                   into layer_1_to_3_output */
                for (k = 0; k < Mf_n*In_n + 2*Rule_n; k++)
                        layer_1_to_3_output[j][k]
                                = node_p[k + In_n]->value;
                target = training_data_matrix[j][In_n];

                get_kalman_data(kalman_data, target);
                kalman(i, j, kalman_data, kalman_parameter);
        }
        put_kalman_parameter(kalman_parameter);
        clear_de_dp();
        for (j = 0; j < training_data_n; j++) {
                put_input_data(j, training_data_pointer);
                /* get output of layer 1 to 3
                   from layer_1_to_3_output */
                for (k = 0; k < Mf_n*In_n + 2*Rule_n; k++){
                        node_p[k + In_n]->value
                                = layer_1_to_3_output[j][k];
}
                /* calculate outputs of layer 4 and 5 */
                calculate_output(In_n + In_n*Mf_n + 2*Rule_n, Node_n - 1);
                anfis_output[j] = node_p[Node_n - 1]->value;
                target = training_data_matrix[j][In_n];
                de_dout = -2*(target - node_p[Node_n-1]->value);
                calculate_de_do(de_dout);
                update_de_dp();
        }
        training_error_measure(training_data_pointer, anfis_output,
                training_data_n, trn_error);

        trn_rmse_error[i] = trn_error[0];
        if (checking_data_n != 0) {
                epoch_checking_error(checking_data_pointer,
                        checking_data_n, chk_error);
                chk_rmse_error[i] = chk_error[0];

                printf("%3d \t %lf \t %lf\n", i+1,
                        trn_error[ERROR_TYPE],
                        chk_error[ERROR_TYPE]);
        } else
                printf("%4d \t %lf\n", i+1,
                        trn_error[ERROR_TYPE]);
        if (trn_rmse_error[i] < min_RMSE) {
                min_RMSE = trn_rmse_error[i];
                record_parameter(parameter_array);
        }
        /* update parameters in 1st layer */
        update_parameter(1, step_size);
```

```
                update_step_size(trn_rmse_error, i, &step_size,
                        decrease_rate, increase_rate);
        }
        printf("Minimal training RMSE = %lf\n", min_RMSE);
        restore_parameter(parameter_array);
        write_parameter(abc,pqr);
        write_array(trn_rmse_error, epoch_n, trn_evol);
        if (checking_data_n != 0)
                write_array(chk_rmse_error, epoch_n, chk_evol);
        write_array(step_size_array, epoch_n, step_evol);
  for (j = 0; j < epoch_n; j++) {
  trn_evol_error[j][0]=j+1;
    trn_evol_error[j][1]=trn_rmse_error[j];
      trn_evol_error[j][2]=step_size_array[j];
  }
}
```

B.2.18. Rutina de la interfaz Fortran/C de la librería de chequeo (ccheck.c)

```
#include ".../scilab-5.1.1/include/scilab/core/machine.h"
#include "anfis.h"
#include "misc_def.h"

NODE_T **node_p;
int In_n, Mf_n, Rule_n, Node_n, funper;
void F2C(ccheck)(int *k, int *L, int *pchk, int *Fp, double *abcin, int *ma, int *na, double *pqrin, int
*mb, int *nb, double *checking_data_pointer, int *md, double *anfis_output)
{
  int i,j,s;
  int checking_data_n;
  double **abc, **pqr;
  double **checking_data_matrix;
  In_n=*k;
  Mf_n=*L;
  checking_data_n=*pchk;
  funper=*Fp;

        abc = (double **)create_matrix
                        (*ma, *na, sizeof(double));
        pqr = (double **)create_matrix
                        (*mb, *nb, sizeof(double));
        checking_data_matrix = (double **)create_matrix
                        (*pchk, *k + 1, sizeof(double));

s=0;j=0;
for ( i =0 ; i < (*na)*(*ma) ; i++){
       abc[s][j] = abcin[i];
       s=s++;
       if (s > *ma-1){
       s=0;j=j++;
       }
  }
```

```
s=0;j=0;
for ( i =0 ; i < (*mb)*(*nb) ; i++){
        pqr[s][j] = pqrin[i];
        s=s++;
        if (s > *mb-1)
        {
        s=0;j=j++;
        }
  }

s=0;j=0;
for ( i =0 ; i < checking_data_n*(In_n+1) ; i++){
        checking_data_matrix[s][j] = checking_data_pointer[i];
        s=s++;
        if (s > *md-1){
        s=0;j=j++;
        }
  }

checka(In_n, Mf_n, checking_data_n, funper, abc, pqr, checking_data_matrix, anfis_output);
}
```

B.2.19. Rutina de interfaz Fortran/C de la librería de aprendizaje en línea (chyontrain.c)

```
#include ".../scilab-5.1.1/include/scilab/core/machine.h"
#include "/root/Desktop/entrega/anfislab/interface/anfis.h"
#include "/root/Desktop/entrega/anfislab/interface/misc_def.h"
NODE_T **node_p;
int In_n, Mf_n, Rule_n, Node_n, funper;
double **abc, **pqr, **training_data_matrix;
void F2C(chyontrain)(int *k, int *L, int *pin, int *Fp, double *ka, double *abcin, int *ma, int *na,
double *pqrin, int *mb, int *nb, double *training_data_pointer, int *mc, double *trn_evol_error)
{
  int i,j,s;
  int training_data_n;
  double step_size;
  double **trn_evol;
  In_n=*k;
  Mf_n=*L;
  training_data_n=*pin;
  funper=*Fp;
  step_size=*ka;
printf("In_n %3d \t Mf_n %3d \t training_data_n %3d \t funper %3d \t step_size %3f\n", In_n, Mf_n,
training_data_n, funper, step_size);

        abc = (double **)create_matrix
                        (*ma, *na, sizeof(double));
        pqr = (double **)create_matrix
                        (*mb, *k + 1, sizeof(double));
        training_data_matrix = (double **)create_matrix
                        (*pin, *k + 1, sizeof(double));
```

```
            trn_evol = (double **)create_matrix
                                (*pin, *na, sizeof(double));

s=0;j=0;
for ( i =0 ; i < (*na)*(*ma) ; i++){
        abc[s][j] = abcin[i];
        s=s++;
        if (s > *ma-1){
        s=0;j=j++;
        }
   }

s=0;j=0;
for ( i =0 ; i < (*mb)*(*nb) ; i++){
        pqr[s][j] = pqrin[i];
        s=s++;
        if (s > *mb-1)
        {
        s=0;j=j++;
        }
   }

s=0;j=0;
for ( i =0 ; i < (*mc)*(*nb) ; i++){
        training_data_matrix[s][j] = training_data_pointer[i];
        s=s++;
        if (s > *mc-1){
        s=0;j=j++;
        }
   }

hyonline(In_n, Mf_n, training_data_n, funper, step_size, abc, pqr, training_data_matrix, trn_evol);
printf("Paso 1 \n");
s=0;j=0;
for ( i =0 ; i < (*na)*(*ma) ; i++){
        abcin[i] = abc[s][j];
        s=s++;
        if (s > *ma-1){
        s=0;j=j++;
        }
   }
printf("Paso 2 \n");
s=0;j=0;
for ( i =0 ; i < (*mb)*(*nb) ; i++){
        pqrin[i] = pqr[s][j];
        s=s++;
        if (s > *mb-1){
        s=0;j=j++;
        }
   }
printf("Paso 3 \n");
s=0;j=0;
for ( i =0 ; i < (*mc)*(*nb); i++) {
```

```
        trn_evol_error[i] = trn_evol[s][j];
        s=s++;
        if (s > *mc-1){
        s=0;j=j++;
        }
   }
printf("Paso 4 \n");
}
```

B.2.20. Rutina de interfaz Fortran/C de la librería de aprendizaje fuera de línea (chytrain.c)

```
#include "...scilab-5.1.1/include/scilab/core/machine.h"
#include "anfis.h"
#include "misc_def.h"
NODE_T **node_p;
int In_n, Mf_n, Rule_n, Node_n, funper;
double **abc, **pqr, **training_data_matrix, **checking_data_matrix;
void F2C(chytrain)(int *k, int *L, int *pin, int *pchk, int *Fp, int *itr, double *erp, double *ka, double
*abcin, int *ma, int *na, double *pqrin, int *mb, int *nb, double *training_data_pointer, int *mc,
double *checking_data_pointer, int *md, double *trn_evol_error)
{
   int i,j,s;
   int training_data_n,checking_data_n,epoch_n;
   double step_size,err_req;
   double **trn_evol;
   In_n=*k;
   Mf_n=*L;
   training_data_n=*pin;
   checking_data_n=*pchk;
   funper=*Fp;
   epoch_n=*itr;
   step_size=*ka;
   err_req=*erp;
printf("In_n %3d \t Mf_n %3d \t training_data_n %3d \t checking_data_n %3d \t funper %3d \t
epoch_n %3d \t step_size %3f\n", In_n, Mf_n, training_data_n, checking_data_n, funper, epoch_n,
step_size);

            abc = (double **)create_matrix
                              (*ma, *na, sizeof(double));
            pqr = (double **)create_matrix
                              (*mb, *k +1, sizeof(double));
            training_data_matrix = (double **)create_matrix
                              (*pin, *k + 1, sizeof(double));
            checking_data_matrix = (double **)create_matrix
                              (*pchk, *k + 1, sizeof(double));
            trn_evol = (double **)create_matrix
                              (*itr, *na, sizeof(double));

s=0;j=0;
for ( i =0 ; i < (*na)*(*ma) ; i++){
        abc[s][j] = abcin[i];
```

```
        s=s++;
        if (s > *ma-1){
        s=0;j=j++;
        }
   }

s=0;j=0;
for ( i =0 ; i < (*mb)*(*nb) ; i++){
        pqr[s][j] = pqrin[i];
        s=s++;
        if (s > *mb-1)
        {
        s=0;j=j++;
        }
   }

s=0;j=0;
for ( i =0 ; i < (*mc)*(*nb) ; i++){
        training_data_matrix[s][j] = training_data_pointer[i];
        s=s++;
        if (s > *mc-1){
        s=0;j=j++;
        }
   }

s=0;j=0;
for ( i =0 ; i < checking_data_n*(In_n+1) ; i++){
        checking_data_matrix[s][j] = checking_data_pointer[i];
        s=s++;
        if (s > *md-1){
        s=0;j=j++;
        }
   }

hybrid(In_n, Mf_n, training_data_n, checking_data_n, funper, epoch_n, err_req, step_size, abc, pqr,
training_data_matrix, checking_data_matrix, trn_evol);

s=0;j=0;
for ( i =0 ; i < (*na)*(*ma) ; i++){
        abcin[i] = abc[s][j];
        s=s++;
        if (s > *ma-1){
        s=0;j=j++;
        }
   }

s=0;j=0;
for ( i =0 ; i < (*mb)*(*nb) ; i++){
        pqrin[i] = pqr[s][j];
        s=s++;
        if (s > *mb-1){
        s=0;j=j++;
        }
```

```
  }

s=0;j=0;
for ( i =0 ; i < (*itr)*(*ma) ; i++) {
        trn_evol_error[i] = trn_evol[s][j];
        s=s++;
        if (s > *itr-1){
        s=0;j=j++;
        }
  }
}
```

B.2.21. Rutina de interfaz de Scilab/Fortran de la librería de chequeo (check.c)

```
#include "stack-c.h"
/*******************************************
 * SCILAB function : check, fin = 1
 *******************************************/

int intscheck(char *fname)
{
 int m1,n1,l1,m2,n2,l2,m3,n3,l3,m4,n4,l4,m5,n5,l5,m6,n6,l6,m7,n7,l7,un=1,mn8,l8;
 CheckRhs(7,7);
 CheckLhs(1,1);
 /*  checking variable k */
 GetRhsVar(1,"i",&m1,&n1,&l1);
 CheckScalar(1,m1,n1);
 /*  checking variable L */
 GetRhsVar(2,"i",&m2,&n2,&l2);
 CheckScalar(2,m2,n2);
 /*  checking variable pc */
 GetRhsVar(3,"i",&m3,&n3,&l3);
 CheckScalar(3,m3,n3);
 /*  checking variable FP */
 GetRhsVar(4,"i",&m4,&n4,&l4);
 CheckScalar(4,m4,n4);
 /*  checking variable ab */
 GetRhsVar(5,"d",&m5,&n5,&l5);
 /*  checking variable pq */
 GetRhsVar(6,"d",&m6,&n6,&l6);
 /*  checking variable Ec */
 GetRhsVar(7,"d",&m7,&n7,&l7);
 /* cross variable size checking */
 CheckDimProp(6,7,n6 != n7);
 CreateVar(8,"d",(un=1,&un),(mn8=*istk(l3),&mn8),&l8);/* named: anfout */
 C2F(ccheck)(istk(l1),istk(l2),istk(l3),istk(l4),stk(l5),&m5,&n5,stk(l6),&m6,&n6,stk(l7),&m7,stk(l8))
 ;
 LhsVar(1)= 8;
 return 0;
}
```

B.2.22. Rutina de interfaz de Scilab/Fortran de la librería de aprendizaje en línea (hyon.c)

```
#include "stack-c.h"
/*******************************************
 * SCILAB function : hyontrain, fin = 1
 *******************************************/

int intshyontrain(char *fname)
{
 int m1,n1,l1,m2,n2,l2,m3,n3,l3,m4,n4,l4,m5,n5,l5,m6,n6,l6,m7,n7,l7,m8,n8,l8,l9;
 CheckRhs(8,8);
 CheckLhs(1,3);
 /*  checking variable k */
 GetRhsVar(1,"i",&m1,&n1,&l1);
 CheckScalar(1,m1,n1);
 /*  checking variable L */
 GetRhsVar(2,"i",&m2,&n2,&l2);
 CheckScalar(2,m2,n2);
 /*  checking variable pi */
 GetRhsVar(3,"i",&m3,&n3,&l3);
 CheckScalar(3,m3,n3);
 /*  checking variable FP */
 GetRhsVar(4,"i",&m4,&n4,&l4);
 CheckScalar(4,m4,n4);
 /*  checking variable ka */
 GetRhsVar(5,"d",&m5,&n5,&l5);
 CheckScalar(5,m5,n5);
 /*  checking variable ab */
 GetRhsVar(6,"d",&m6,&n6,&l6);
 /*  checking variable pq */
 GetRhsVar(7,"d",&m7,&n7,&l7);
 /*  checking variable Ei */
 GetRhsVar(8,"d",&m8,&n8,&l8);
 /* cross variable size checking */
 CheckDimProp(7,8,n7 != n8);
 CreateVar(9,"d",&m8,&n6,&l9);/* named: Te */
 C2F(chyontrain)(istk(l1),istk(l2),istk(l3),istk(l4),stk(l5),stk(l6),&m6,&n6,stk(l7),&m7,&n7,stk(l8)
 ,&m8,stk(l9));
 LhsVar(1)= 6;
 LhsVar(2)= 7;
 LhsVar(3)= 9;
 return 0;
}
```

B.2.23. Rutina de interfaz de Scilab/Fortran de la librería de aprendizaje fuera de línea (hysci.c)

```
#include "stack-c.h"
/*******************************************
 * SCILAB function : hytrain, fin = 1
```

```
 ****************************************/

int intshytrain(char *fname)
{
 int
m1,n1,l1,m2,n2,l2,m3,n3,l3,m4,n4,l4,m5,n5,l5,m6,n6,l6,m7,n7,l7,m8,n8,l8,m9,n9,l9,m10,n10,l10,m11
 ,n11,l11,m12,n12,l12,l13,err=0;
 CheckRhs(12,12);
 CheckLhs(1,3);
 /*  checking variable k */
 GetRhsVar(1,"i",&m1,&n1,&l1);
 CheckScalar(1,m1,n1);
 /*  checking variable L */
 GetRhsVar(2,"i",&m2,&n2,&l2);
 CheckScalar(2,m2,n2);
 /*  checking variable pi */
 GetRhsVar(3,"i",&m3,&n3,&l3);
 CheckScalar(3,m3,n3);
 /*  checking variable pc */
 GetRhsVar(4,"i",&m4,&n4,&l4);
 CheckScalar(4,m4,n4);
 /*  checking variable FP */
 GetRhsVar(5,"i",&m5,&n5,&l5);
 CheckScalar(5,m5,n5);
 /*  checking variable it */
 GetRhsVar(6,"i",&m6,&n6,&l6);
 CheckScalar(6,m6,n6);
 /*  checking variable err */
 GetRhsVar(7,"d",&m7,&n7,&l7);
 CheckScalar(7,m7,n7);
 /*  checking variable ka */
 GetRhsVar(8,"d",&m8,&n8,&l8);
 CheckScalar(8,m8,n8);
 /*  checking variable ab */
 GetRhsVar(9,"d",&m9,&n9,&l9);
 /*  checking variable pq */
 GetRhsVar(10,"d",&m10,&n10,&l10);
 /*  checking variable Ei */
 GetRhsVar(11,"d",&m11,&n11,&l11);
 /*  checking variable Ec */
 GetRhsVar(12,"d",&m12,&n12,&l12);
 /* cross variable size checking */
 CheckDimProp(10,11,n10 != n11);
 CheckDimProp(11,12,n11 != n12);
 CreateVar(13,"d",istk(l6),&n9,&l13);/* named: Te */
 C2F(chytrain)(istk(l1),istk(l2),istk(l3),istk(l4),istk(l5),istk(l6),stk(l7),stk(l8),stk(l9),&m9,&n9
 ,stk(l10),&m10,&n10,stk(l11),&m11,stk(l12),&m12,stk(l13));
 if (err >  0) {
  Scierror(999,"%s: Internal Error \n",fname);
  return 0;
 };
 LhsVar(1)= 9;
 LhsVar(2)= 10;
```

```
 LhsVar(3)= 13;
 return 0;
}
```

B.2.24. Rutina principal del ejecutable (main.c)

```
#include "../anfis.h"
#include "../misc_def.h"

NODE_T **node_p;
int In_n, Mf_n, Rule_n, Node_n, funper;

main(argc, argv)
int argc;
char **argv;
{
  int i,j,s;
  int pin,pchk,epoch_n,parameter_n;
  double step_size,err_req;
  double **abc, **pqr, **training_data_matrix, **checking_data_matrix;
  double **trn_evol;
  double tmp;
switch(argc){
        case 1:
                printf("Por lo menos debe colocar el número de muestras de entrenamiento. \n Ej:
anfis 100 \n", argv[0]);
                exit(1);
        case 2:
                pchk=0;
                epoch_n=500;
                err_req=0;
                break;
        case 3:
                pchk = atoi(argv[2]);
                epoch_n=500;
                err_req=0;
                break;
        case 4:
                pchk = atoi(argv[2]);
                epoch_n = atoi(argv[3]);
                err_req=0;
                break;
        case 5:
                pchk = atoi(argv[2]);
                epoch_n = atoi(argv[3]);
                err_req = atoi(argv[4]);
                break;
        default:
                printf("Como maximo debe colocar:\n El número de muestras de entrenamiento \n El
número de muestras de validacion \n El número de iteraciones \n El error minimo deseado.\n Ej: anfis
100 0 500 0 \n", argv[0]);
        exit(1);
```

```
}
pin = atoi(argv[1]);
FILE *fp1, *open_file();
fp1 = open_file(INIT_PARA_FILE, "r");
        for (j = 0; j < 4; j++) {
                if (fscanf(fp1, "%lf", &tmp) != EOF) {
                        switch(j){
                                case 0:
                                        In_n = tmp;
                                case 1:
                                        Mf_n = tmp;
                                case 2:
                                        funper = tmp;
                                case 3:
                                        step_size = tmp;
                                }
                }
        }

Rule_n = (int) pow((double)Mf_n, (double)In_n);
Node_n = In_n + In_n*Mf_n + 3*Rule_n + 1;

abc = (double **)create_matrix
                ((In_n*Mf_n), 3, sizeof(double));
pqr = (double **)create_matrix
                (Rule_n, In_n + 1, sizeof(double));
training_data_matrix = (double **)create_matrix
                (pin, In_n + 1, sizeof(double));
checking_data_matrix = (double **)create_matrix
                (pchk, In_n + 1, sizeof(double));
trn_evol = (double **)create_matrix
                (epoch_n, 3, sizeof(double));


s=0;i=0;
        for (j = 4; j < (4 + 3*In_n*Mf_n); j++) {
                if (fscanf(fp1, "%lf", &tmp) != EOF) {
                        abc[s][i] = tmp;
                s=s++;
                if (s > In_n*Mf_n-1){
                      s=0;i=i++;
                        }
                }
        }

s=0;i=0;
        for ( j =(4 + 3*In_n*Mf_n) ; j < (4 + 3*In_n*Mf_n+Rule_n*(In_n+1))  ; j++){
        if (fscanf(fp1, "%lf", &tmp) != EOF) {
                        pqr[s][i] = tmp;
                s=s++;
                if (s > Rule_n-1){
                s=0;i=i++;
                }
```

```
                        }
                }

fclose(fp1);

        FILE *fp2, *open_file();
        fp2 = open_file(TRAIN_DATA_FILE, "r");
        for (i = 0; i < pin; i++)
                for (j = 0; j < In_n + 1; j++)
                        if (fscanf(fp2, "%lf", &tmp) != EOF)
                                training_data_matrix[i][j] = tmp;
                        else
                                exit1("No hay suficientes datos!");
        fclose(fp2);

if (pchk != 0){
        FILE *fp3, *open_file();
        fp3 = open_file(CHECK_DATA_FILE, "r");
        for (i = 0; i < pchk; i++)
                for (j = 0; j < In_n + 1; j++)
                        if (fscanf(fp3, "%lf", &tmp) != EOF)
                                checking_data_matrix[i][j] = tmp;
                        else
                                exit1("No hay suficientes datos!");
        fclose(fp3);
}

hybrid(In_n, Mf_n, pin, pchk, funper, epoch_n, err_req, step_size, abc, pqr, training_data_matrix,
checking_data_matrix, trn_evol);

        FILE *fp4, *open_file();
        fp4 = open_file(FINA_PARA_FILE, "w");

                for (j = 0; j < 4; j++) {
                        switch(j){
                                case 0:
                                        tmp = In_n;
                                        break;
                                case 1:
                                        tmp = Mf_n;
                                        break;
                                case 2:
                                        tmp = funper;
                                        break;
                                case 3:
                                        tmp = trn_evol[epoch_n-1][2];
                                        break;
                                }
                        fprintf(fp4, "%lf \n", tmp);
                        }

s=0;i=0;
                for (j = 4; j < (4 + 3*In_n*Mf_n); j++) {
```

```
                                tmp = abc[s][i];
                                fprintf(fp4, "%lf \n", tmp);
                        s=s++;
                        if (s > In_n*Mf_n-1){
                              s=0;i=i++;
                                }
                }

s=0;i=0;
                for ( j =(4 + 3*In_n*Mf_n) ; j < (4 + 3*In_n*Mf_n+Rule_n*(In_n+1))  ; j++){
                                tmp = pqr[s][i];
                                fprintf(fp4, "%lf \n", tmp);
                        s=s++;
                        if (s > Rule_n-1){
                        s=0;i=i++;
                        }
                }

        fclose(fp4);

        FILE *fp5, *open_file();
        fp5 = open_file(ARCH_EVOLUCION, "w");
                                fprintf(fp5, "iteracion \t error  \t paso \n");
                for ( j = 0 ; j < epoch_n  ; j++)
                                fprintf(fp5, "%lf \t %lf \t %lf \n", trn_evol[j][0], trn_evol[j][1],
trn_evol[j][2]);
        fclose(fp5);

}
```

B.2.25. Rutina MAKE para la compilación del ejecutable (Makefile)

```
# Make file for ANFIS

HPATH = .../interface

CFILES = $(HPATH)/ejecutable/main.c \
  $(HPATH)/hybrid.c $(HPATH)/trn_err.c \
        $(HPATH)/datastru.c $(HPATH)/backward.c \
        $(HPATH)/forward.c $(HPATH)/de_dp.c \
        $(HPATH)/new_para.c $(HPATH)/debug.c \
        $(HPATH)/kalman.c $(HPATH)/stepsize.c \
        $(HPATH)/input.c $(HPATH)/output.c \
        $(HPATH)/lib.c $(HPATH)/chk_err.c

COBJS = $(HPATH)/ejecutable/main.o \
  $(HPATH)/hybrid.o $(HPATH)/trn_err.o \
        $(HPATH)/datastru.o $(HPATH)/backward.o \
        $(HPATH)/forward.o $(HPATH)/de_dp.o \
        $(HPATH)/new_para.o $(HPATH)/debug.o \
        $(HPATH)/kalman.o $(HPATH)/stepsize.o \
        $(HPATH)/input.o $(HPATH)/output.o \
```

```
	$(HPATH)/lib.o $(HPATH)/chk_err.o

HFILES = $(HPATH)/standard.h $(HPATH)/anfis.h $(HPATH)/misc_def.h

TARGET  = anfis

CFLAGS  =  -O

.c.o : ; gcc -c -O -o $@ $*.c

$(TARGET) : ${COBJS}
	gcc -O -o $(TARGET) $(COBJS) -lm
#	gcc -O -shared -fPIC -mrtd -o hybrid.so $(COBJS) -lm

$(COBJS) : $(HFILES)


# for saber

saber_src: $(CFILES)
	#load $(CFLAGS) $(CFILES) -lm

saber_obj: $(COBJS)
	#load $(CFLAGS) $(COBJS) -lm
```

B.3. **Códigos PHP**

Aquí se muestran los códigos usados para el entorno web

B.3.1. Página principal (pa.php)

```
<?php

$Alerta= array();
$Alerta[0]='imagenes/aturquesa.gif';
$Alerta[1]='imagenes/averde.gif';
$Alerta[2]='imagenes/aamarilla.gif';
$Alerta[3]='imagenes/anaranja.gif';
$Alerta[4]='imagenes/aroja.gif';

$arrLines = file('data/datosweb.csv');
$lineas = 1;//manda la alerta de la ultima línea del arreglo
$numero_lineas = count($arrLines)-$lineas;
//Para el formato CSV "explode( ',',..." para texto normal "explode( ' ',..."
$arrResult = explode( ',', $arrLines[$numero_lineas]);
$presente[0]=intval($arrResult[4])-1;
if ($presente[0]>4)
$presente[0]=4;
$futuro[0]=intval($arrResult[5])-1;
if ($futuro[0]>4)
$futuro[0]=4;
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" lang="es">

<head>
        <TITLE>Proyecto PROCEDA</TITLE>
<META http-equiv=Content-Type content="text/html; charset=iso-8859-1">
<META http-equiv=EXPIRES content=0>
<META http-equiv=refresh content=500> <!-- refresca la página cada 500 seg. -->
<META http-equiv=pragma content=no-cache>
<META content=DOCUMENT name=RESOURCE-TYPE>
<META content="Andres Avendano" name=AUTHOR>
</head>

<body>
<body background="">

<table cellspacing=0 cellpadding=0 width="100%" border=0>

        <tbody>
                <tr>
                        <td colspan="3" rowspan="1" height="150px">
                        <div align="center"><table align="center" border="0" cellpadding="0"
```

```
cellspacing="0"
            width="60%">
            <tbody>
<div align="center"><H1>UNIVERSIDAD CENTRAL DE VENEZUELA</H1></div>
<div align="center"><H2>Modelo de inferencia hidrológico</H2></div>
<div align="center"><H3>Estación Macuto</H3></div>
   <!-- ImageReady Slices (delta4.psd) -->
<!-- primera columna -->
<tr>

<td width="137"> <table width="100%" border="0" cellspacing="0" cellpadding="0">
 <tr>
  <td><div align="center"><a href="http://www.ucv.ve"><img src="imagenes/l_ucv_p2.gif" alt="Web
de la Universidad Central de Venezuela" width="57" height="57" border="0"></a></div>
  </td>
  <td><div align="center"><a href="http://www.ing.ucv.ve"><img src="imagenes/l_it1.gif" alt="Web
de Ingeniería de la UCV" width="64" height="51" border="0"></a></div>
  </td>
  <td><div align="center"><a href="http://imf.ing.ucv.ve/"><img src="imagenes/logo_imf.gif"
alt="Web de Instituto de Mecanica de Fluidos" width="57" height="57" border="0"></a></div>
  </td>
  <td><div align="center"><a href="http://neutron.ing.ucv.ve/"><img src="imagenes/eielogo.gif"
alt="Web de la Escuela de Ingeniería Eléctrica" width="64" height="51" border="0"></a></div>
  </td>
 </tr>
</table></td>

 <table width="100%" border="0" cellspacing="0" cellpadding="0">

<td width="137"> <table width="100%" border="0" cellspacing="0" cellpadding="0">
  <td width="137"> <table width="100%" border="0" cellspacing="0" cellpadding="0">

<tr><td><div align="center"><a class=BarraTitle>Alerta Presente</a></div>
</td></tr>

<tr><td><div align="center"><img src= "<?php echo $Alerta[$presente[0]]; ?>" alt="Ultima alerta del
sistema" width="57" height="57" border="0"></div></td></tr>

<tr><td><div align="center"><a class=BarraTitle>Alerta Pronosticada </a></div>
</td></tr>

<tr><td><div align="center"><img src= "<?php echo $Alerta[$futuro[0]]; ?>" alt="Pronóstico del
modelo" width="57" height="57" border="0"></div></td></tr>

   <tr>
    <td><div align="center"><a class=BarraTitle>Sobre este pronóstico</a></div>
    </td>
   </tr>

   <tr>
    <td class=BarraTexto><div align="left"><img src="imagenes/arrow.gif" width="9" height="9"><a
href="colores.html"><font size="2" face="Verdana, Arial, Helvetica, sans-serif"> Codigo de
colores</font></a></div>
```

```
    </td>

   <tr>
    <td class=BarraTexto><div align="left"><img src="imagenes/arrow.gif" width="9" height="9"><a
href="descarga.php"><font size="2" face="Verdana, Arial, Helvetica, sans-serif"> Decargar
Datos</font></a></div>
    </td>

   </tr>
  </table></td>

</table></td>

<!-- segunda columna-->
 <td width="20"> 
<div align="center"><img name="fig1" src="acumulado.php"  alt="Grafica 1" width="800"
height="400" border="0"></div></td>
<tr>
 <td>
</td>
 <td width="20"> 
<div align="center"><img name="fig2" src="grafico_linea.php"  alt="Grafica 2" width="800"
height="400" border="0"></div></td>
   </tr>
</table>
</tr>

<!-- onMouseover="ddrivetip('<align =center><br><img src=imagenes/design/phone.jpg><br><br>',
246)" onMouseout="hideddrivetip()";><font size="2"-->

</body>
</html>
```

B.3.2. Gráfica 1 (acumulado.php)

```
<?php
require_once ("/usr/include/php/include/jpgraph-3.0.3/src/jpgraph.php");
require_once ("/usr/include/php/include/jpgraph-3.0.3/src/jpgraph_line.php");
require_once ("/usr/include/php/include/jpgraph-3.0.3/src/jpgraph_date.php");

$graph = new Graph(800,400);
$graph->SetScale('datlin');
$graph->img->SetAntiAliasing();
$graph->xgrid->Show();

DEFINE('NDATAPOINTS',360);
DEFINE('SAMPLERATE',240);

$i=0;
$lineas=48;//considera las ultimas 48 lineas del arreglo
$arrLines = file('data/datosweb.csv');
$numero_lineas = count($arrLines)-$lineas;
$matriz_datos = array_slice($arrLines,$numero_lineas);
foreach($matriz_datos as $line) {
```

```
$arrResult = explode( ',', $line);//Configuración para el formato CSV
$arrFecha = explode( '-', $arrResult[0]);//Separador de fecha
switch ($arrFecha[1]){
case 'Ene': $mes = 1;break;
case 'Feb': $mes = 2;break;
case 'Mar': $mes = 3;break;
case 'Abr': $mes = 4;break;
case 'May': $mes = 5;break;
case 'Jun': $mes = 6;break;
case 'Jul': $mes = 7;break;
case 'Ago': $mes = 8;break;
case 'Sep': $mes = 9;break;
case 'Oct': $mes = 10;break;
case 'Nov': $mes = 11;break;
case 'Dic': $mes = 12;break;
default: $mes = 1;break;
}

$arrhora = explode( ':', $arrResult[1]);
$xdata[$i]=mktime($arrhora[0],$arrhora[1],$arrhora[2],$mes,$arrFecha[0],$arrFecha[2]);
$data[$i]=$arrResult[2];
$i++;
}

$graph->SetMargin(40,40,30,110);
$graph->title->Set("Acumulado de precipitación");
$graph->xaxis->SetLabelAngle(90);

$line = new LinePlot($data,$xdata);
$line->SetFillColor('lightblue@0.5');
$graph->Add($line);
$graph->Stroke();
?>
```

B.3.3. Gráfica 2 (grafico_linea.php)

```
<?php
include ("/usr/include/php/include/jpgraph-3.0.3/src/jpgraph.php");
include ("/usr/include/php/include/jpgraph-3.0.3/src/jpgraph_line.php");

$graph = new Graph(800,400);
$graph->SetScale("linlin");
$graph->img->SetAntiAliasing();
$graph->xgrid->Show();

$i=0;
$lineas=96;//Considera las ultimas 96 horas
$arrResult = array();
$arrLines = file('data/datosweb.csv');
$numero_lineas = count($arrLines)-$lineas;
$matriz_datos = array_slice($arrLines,$numero_lineas);
foreach($matriz_datos as $line) {
```

```
$arrResult = explode( ',', $line);
$ydata1[$i]=$arrResult[4];
$i++;
}

$graph->img->SetMargin(40,20,20,40);
$graph->title->Set("Evolución del Nivel de Alerta");
$graph->xaxis->title->Set("PW72");
$graph->yaxis->title->Set("PW1,5");

$lineplot1=new LinePlot($ydata1);
$lineplot1->SetColor("black");


$graph->Add($lineplot1);
$graph->Stroke();
?>
```

B.3.4. Obtención y descarga de datos (descarga.php)

```
<?php
$path = 'data/datosweb.csv';
$file = 'datos.csv';
#
$type = '';
#

#
if (is_file($path)) {
#
   $size = filesize($path);

#
   if (function_exists('mime_content_type')) {
#
      $type = mime_content_type($path);
//echo $type;
#
   } else if (function_exists('finfo_file')) {
#
      $info = finfo_open(FILEINFO_MIME);
#
      $type = finfo_file($info, $path);
#
      finfo_close($info);
#
   }
#
   if ($type == '') {
#
      $type = "application/force-download";
#
```

```
    }
#
    // Set Headers
#
    header("Content-Type: $type");
#
    header("Content-Disposition: attachment; filename=$file");
#
    header("Content-Transfer-Encoding: binary");
#
    header("Content-Length: " . $size);
#
    // Download File
#
    readfile($path);
#
} else {
#
    die("File not exist !!");
#
}
?>
```

B.3.5. Página de información (colores.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" lang="es">

<head>
        <TITLE>CÓDIGO DE COLORES</TITLE>
<META http-equiv=Content-Type content="text/html; charset=iso-8859-1">
<META http-equiv=EXPIRES content=0>
<META content=DOCUMENT name=RESOURCE-TYPE>
<META content="Andres Avendano" name=AUTHOR>
</head>

<body>
<!-- <table  style="text-align: left;" border="0" width= "100%" cellpadding="5" cellspacing="1"> -->
<body background="">

<table cellspacing=0 cellpadding=0 width="100%" border=0>

        <tbody>
                <tr>
                        <td colspan="3" rowspan="1" height="150px">
                        <div align="center"><table align="center" border="0" cellpadding="0"
cellspacing="0"
        width="60%">
        <tbody>
<div align="center"><H1>Universidad Central de Venezuela</H1></div>
<div align="center"><H2>Escala de alerta por colores</H2></div>
```

```
   <!-- ImageReady Slices (delta4.psd) -->
<!-- primera columna -->
<tr>

<td width="137"> <table width="100%" border="0" cellspacing="0" cellpadding="0">
 <tr>
  <td><div align="center"><img src="imagenes/aturquesa.gif" width="57" height="57"
border="0"></div>
  </td>
   <td><div align="center"><a class=BarraTitle>Nivel 1: No existe riesgo, la precipitación acumulada
es mínima </a></div>
   </td>
 </tr>

<tr>
  <td><div align="center"><img src="imagenes/averde.gif" width="57" height="57"
border="0"></div>
  </td>
   <td><div align="center"><a class=BarraTitle>Nivel 2: No existe riesgo, la precipitación acumulada
es baja </a></div>
   </td>
 </tr>

<tr>
  <td><div align="center"><img src="imagenes/aamarilla.gif" width="57" height="57"
border="0"></div>
  </td>
   <td><div align="center"><a class=BarraTitle>Nivel 3: No existe riesgo para la población general,
aunque sí para un sector concreto </a></div>
   </td>
 </tr>

<tr>
  <td><div align="center"><img src="imagenes/anaranja.gif" width="57" height="57"
border="0"></div>
  </td>
   <td><div align="center"><a class=BarraTitle>Nivel 4: Existe un riesgo importante de ocurrencia de
un alud </a></div>
   </td>
 </tr>

<tr>
  <td><div align="center"><img src="imagenes/aroja.gif" width="57" height="57" border="0"></div>
  </td>
   <td><div align="center"><a class=BarraTitle>Nivel 5: El riesgo de ocurrencia de un alud es
extremo</a></div>
   </td>
 </tr>
</table></td>
</tr>

</body>
</html>
```