Entrega Bloque Redes Neuronales: Modelado y Predicción

Ángel Manuel Ferrer Álvarez

23 de diciembre de 2025

Índice

1. Introducción

Este informe detalla el diseño y entrenamiento de diversos sistemas de redes neuronales utilizando la herramienta Xfuzzy, aplicados al modelado de funciones, problemas tridimensionales con Adalines y predicción de series temporales.

2. Apartado 1

2.1. Definición de la función y parámetros del sistema

Para este primer apartado se ha seleccionado la siguiente función no lineal de dos entradas y una salida:

$$F(x, y) = \sin(2\pi x) \cdot \sin(\pi y) \tag{1}$$

El objetivo es modelar su comportamiento en el rango $x, y \in [0, 1]$. Los datos se han generado mediante un script de Python con un mallado aleatorio de 1000 patrones.

Para el diseño del sistema en Xfuzzy, se han establecido los siguientes parámetros:

- **Entradas (X e Y):** Rango $[0, 1]$ con funciones de pertenencia (MFs) de tipo *triangular*.
- **Salida (OUT):** Rango $[-1, 1]$ utilizando funciones de pertenencia de tipo *singleton*.

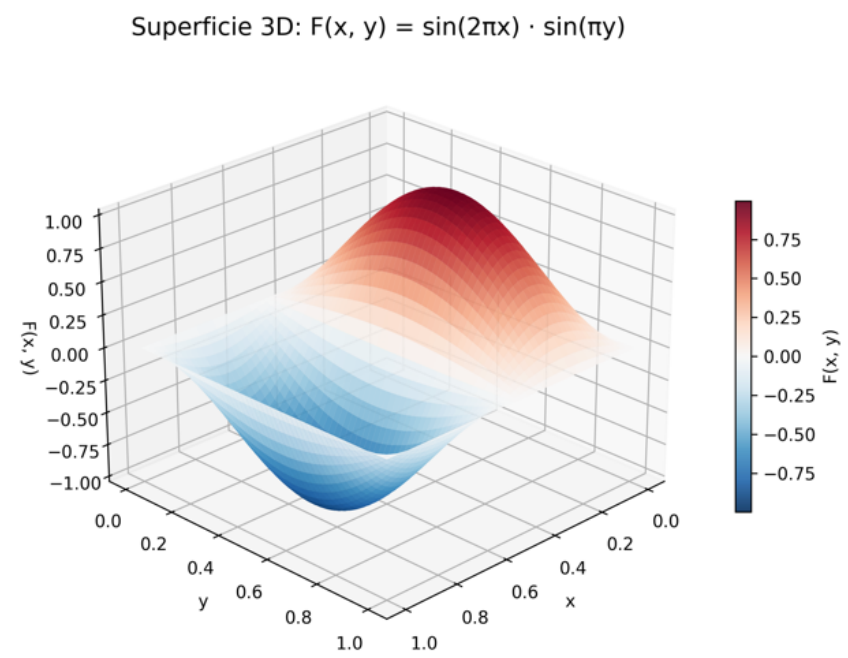

Figura 1: representación de la función.

2.2. Diseño con una única base de reglas: Incremental Grid y Marquardt-Levenberg

2.2.1. Fase 1: Identificación mediante Incremental Grid

Se ha utilizado el algoritmo Incremental Grid para generar una base de reglas inicial. Se estableció un límite de 7 MFs por variable y un objetivo de error de 0.01. Tras la ejecución, el algoritmo identificó automáticamente una estructura compuesta por 56 reglas capaz de aproximar la superficie de forma preliminar.

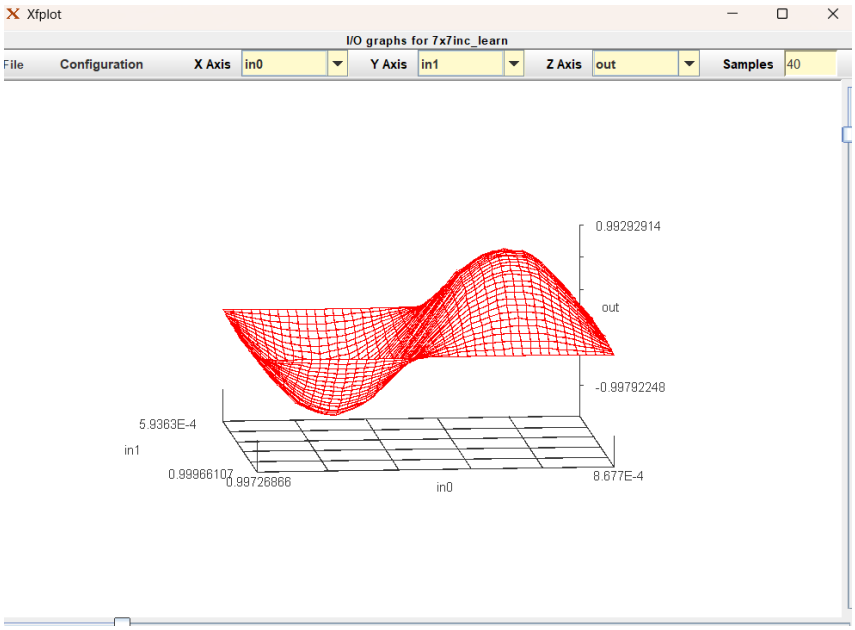

Figura 2: representación gráfica del sistema de base única.

2.2.2. Fase 2: Ajuste con Marquardt-Levenberg

Para refinar el modelo, se aplicó el algoritmo **Marquardt-Levenberg**. El proceso convergió alcanzando un RMSE de **0.00566**, logrando que la superficie de salida se adapte a las curvaturas de la función original.

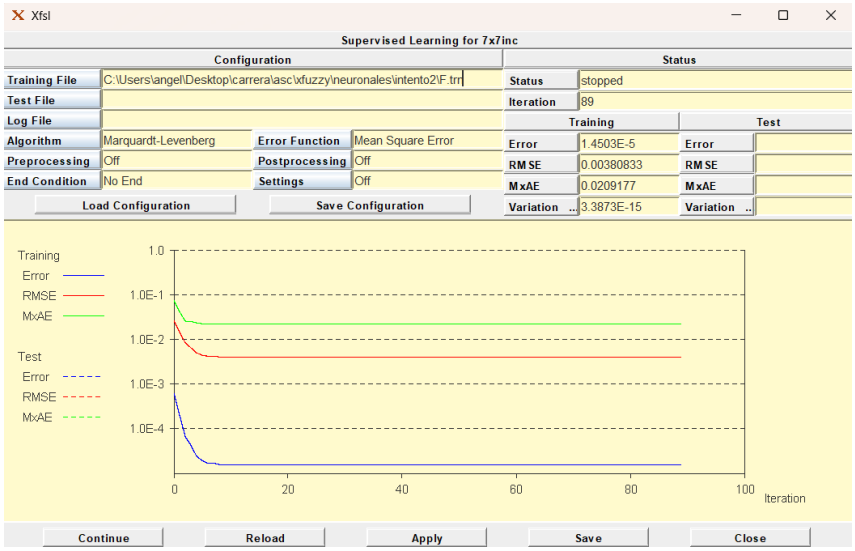

Figura 3: Curva de aprendizaje del sistema de base única.

2.3. Diseño con múltiples bases de reglas (Sistema Jerárquico)

En este apartado se propone una descomposición del problema mediante una arquitectura paralela que explota la naturaleza de la función objetivo, separando el aprendizaje de las componentes en X e Y.

2.3.1. Arquitectura del Sistema de 3 Bases

Se ha diseñado un sistema jerárquico compuesto por tres bloques funcionales interconectados:

- **rb_x y rb_y:** Actúan como extractores de características unidimensionales para cada eje.
- **rb_prod:** Actúa como un combinador que realiza el producto de las señales y ajusta el resultado final.

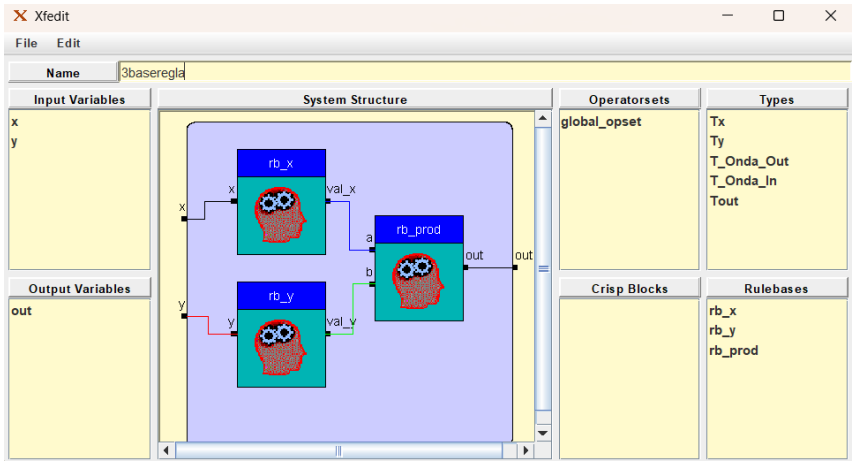

Figura 4: Estructura del sistema jerárquico con tres bases de reglas.

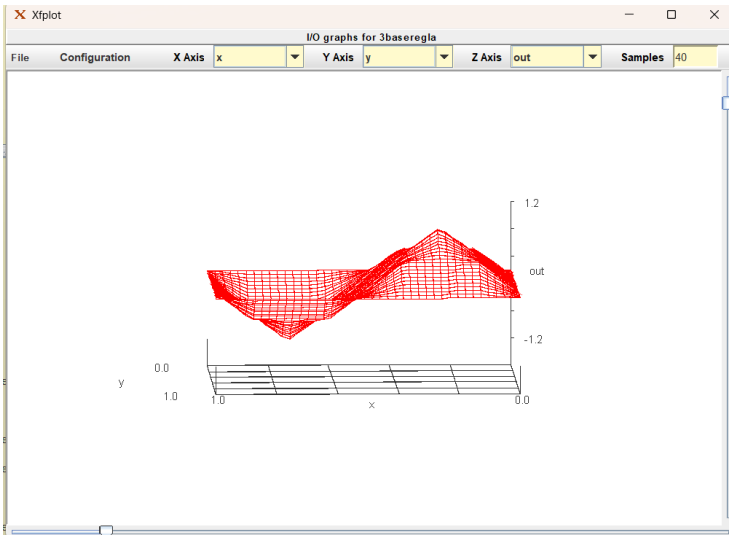

Figura 5: Superficie obtenida por el sistema jerárquico.

2.3.2. Resultados y Entrenamiento

Este sistema fue entrenado globalmente con el algoritmo Marquardt-Levenberg. Gracias a la especialización de las bases, se logró una convergencia muy eficiente, alcanzando un RMSE final de **0.00265**, lo que representa una mejora sustancial respecto al sistema de base única.

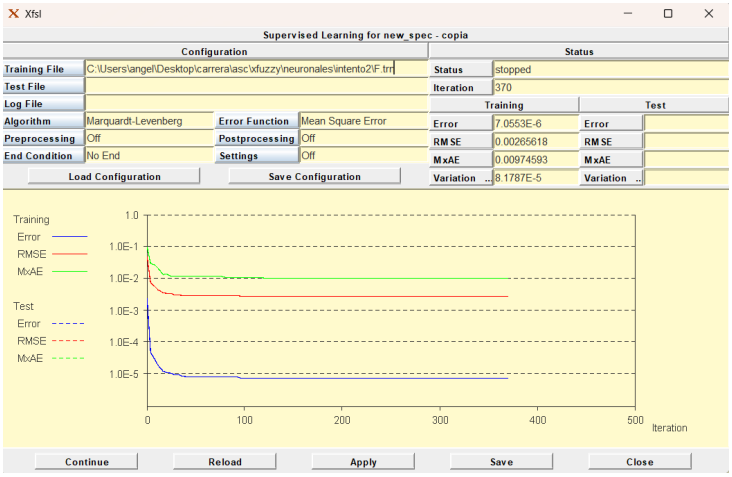

Figura 6: Curva de aprendizaje sistema jerárquico.

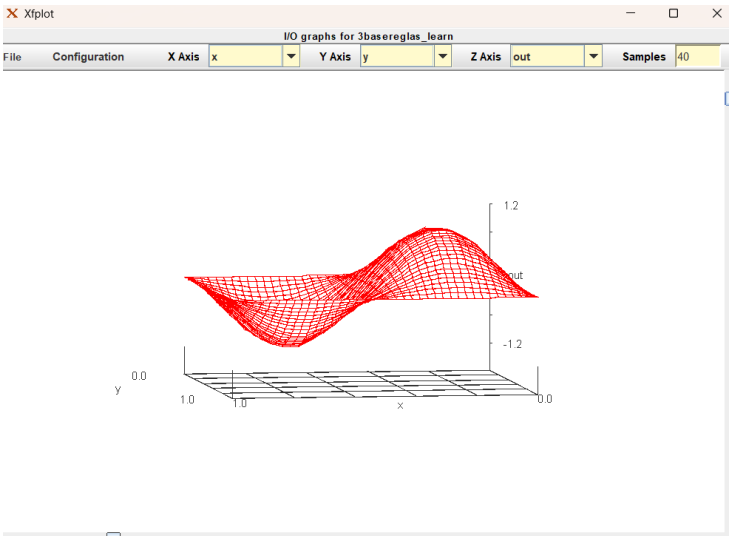

Figura 7: Superficie final obtenida.

2.4. Comparativa y Conclusiones del Apartado 1

Tras analizar ambos diseños, se presentan los resultados comparativos:

Conclusión: El sistema jerárquico ha demostrado ser la estrategia más efectiva para este problema de modelado. Al segmentar la lógica en tres bases especializadas, se ha conseguido reducir el número de reglas totales a 39 (frente a las 56 del sistema Grid), logrando al mismo tiempo un error RMSE significativamente menor. Esta reducción en la complejidad estructural, unida a una mayor precisión, valida la superioridad de los sistemas jerárquicos para aproximar funciones complejas de forma eficiente.

3. Apartado 2

En este bloque se aborda el modelado de un sistema con tres entradas (x_1, x_2, x_3) y una salida, cuya dinámica interna responde a una función sigmoidal combinada con términos lineales.

3.1. Definición del problema y análisis funcional

La función objetivo a aproximar viene definida por la siguiente expresión matemática:

$$out = \frac{1}{1 + e^{5(3x_1 - 2x_2)}} - 0{,}5x_3 \tag{2}$$

Del análisis de la ecuación se desprenden dos componentes críticas:

- **Componente No Lineal:** Una función sigmoide que depende de la combinación lineal de las variables x_1 y x_2.
- **Componente Lineal:** Un término puramente lineal dependiente de la variable x_3 con una pendiente de $-0{,}5$.

3.2. Sistema de base única (Sin jerarquía)

Siguiendo la metodología del ejercicio anterior, se ha diseñado inicialmente un sistema base compuesto por una única base de reglas utilizando el algoritmo de identificación **Incremental Grid**.

3.2.1. Identificación mediante Incremental Grid

Se estableció un límite de 5 funciones de pertenencia por variable. El algoritmo generó automáticamente un sistema de 72 reglas para intentar mapear el espacio tridimensional de forma directa. En las Figuras 8, 9 y 10 se muestran las representaciones gráficas iniciales.

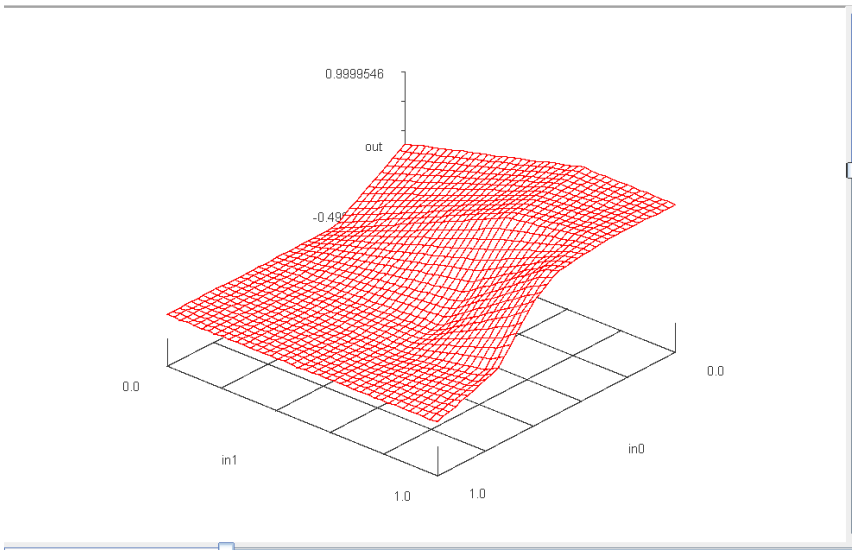

Figura 8: Representación gráfica inicial mediante Incremental Grid (ejes x1 y x2).

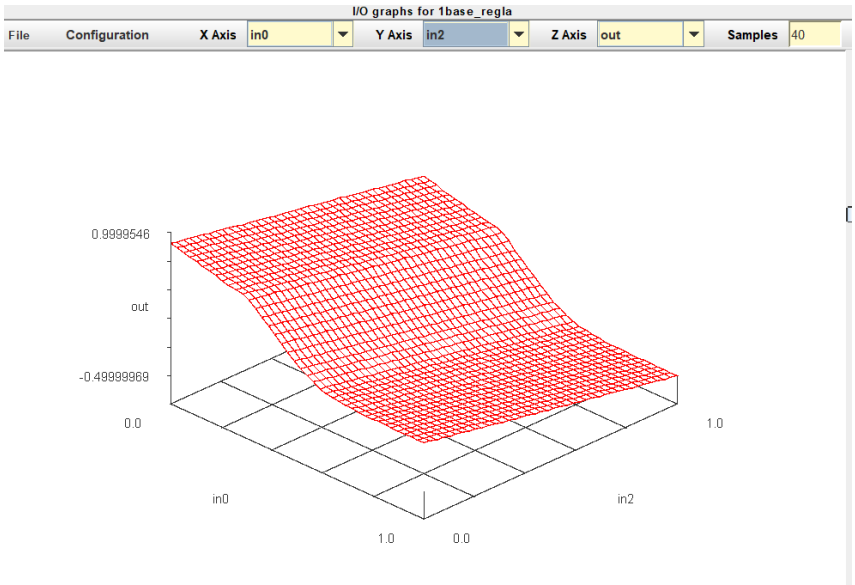

Figura 9: Representación gráfica inicial mediante Incremental Grid (ejes x1 y x3).

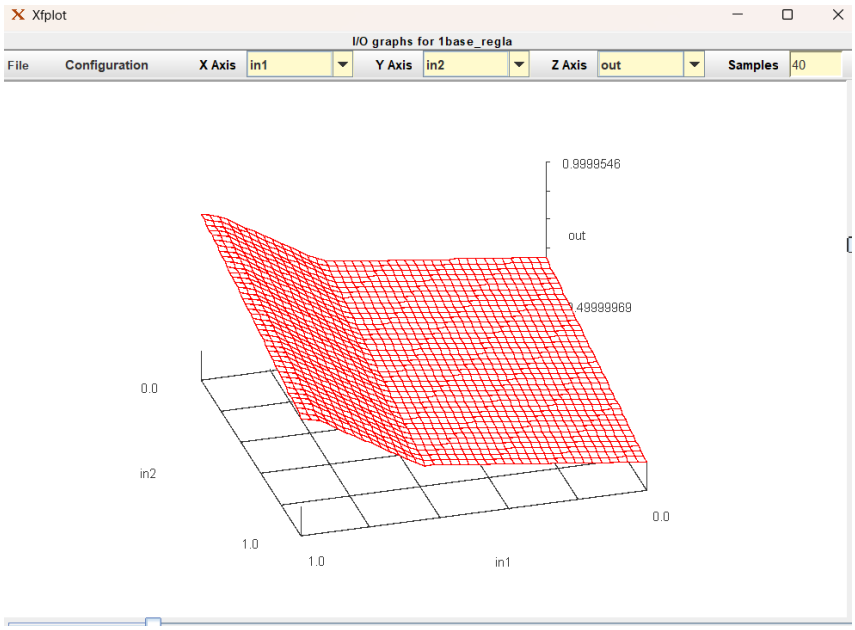

Figura 10: Representación gráfica inicial mediante Incremental Grid (ejes x2 y x3).

3.2.2. Entrenamiento con Marquardt-Levenberg

Para minimizar el error, el sistema se sometió a un aprendizaje supervisado utilizando el algoritmo Marquardt-Levenberg. Tras el entrenamiento, se consiguió un RMSE de 0.0297. El ajuste final se observa en las siguientes figuras:

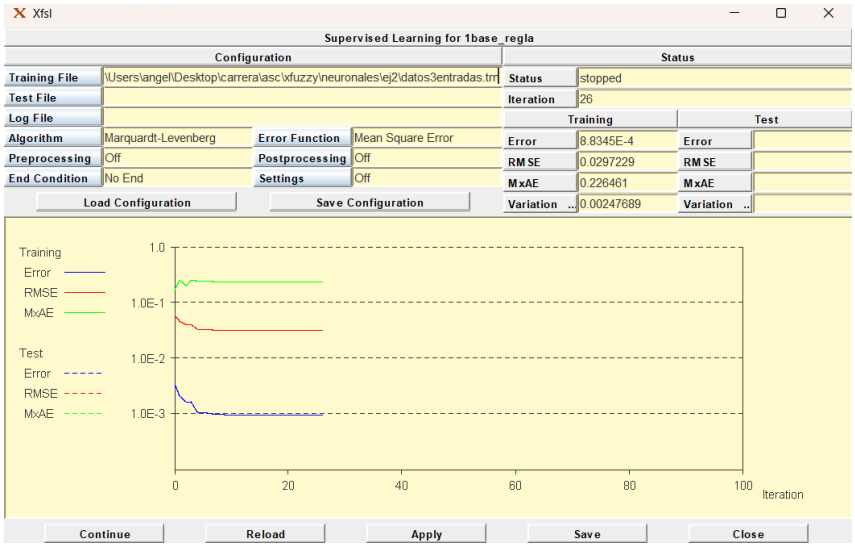

Figura 11: Curva de aprendizaje del sistema de base única (RMSE: 0.0297).

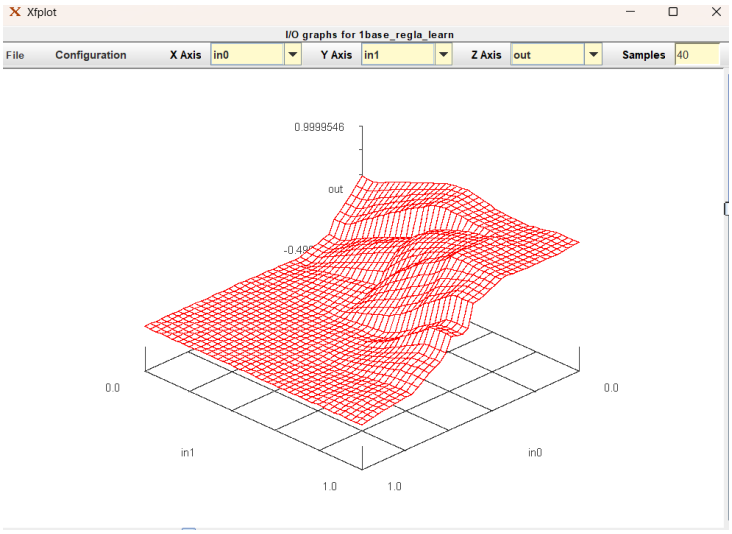

Figura 12: Superficie final entrenada (ejes x1 y x2).

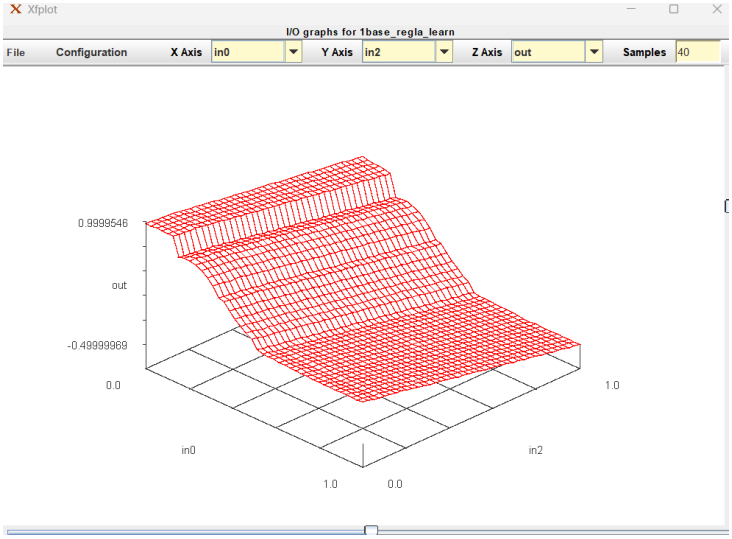

Figura 13: Superficie final entrenada (ejes x1 y x3).

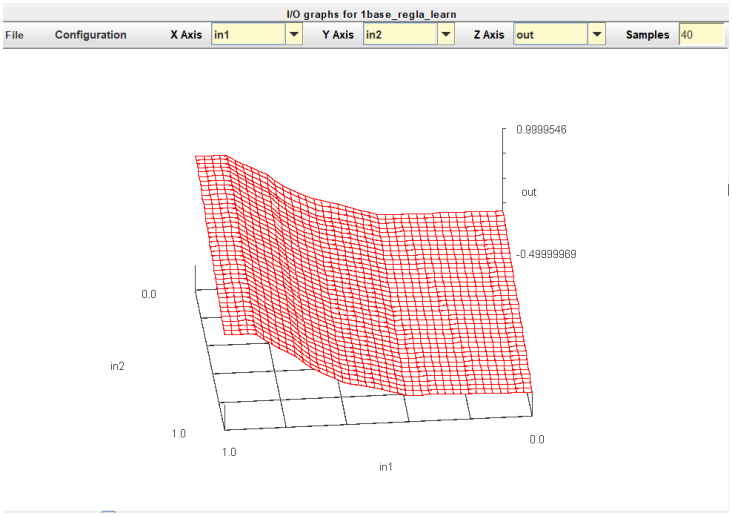

Figura 14: Superficie final entrenada (ejes x2 y x3).

3.3. Sistema Jerárquico Híbrido con Adaline (Takagi-Sugeno)

Para optimizar el diseño, se propone una arquitectura jerárquica que descompone el problema en dos etapas funcionales.

3.3.1. Arquitectura y Configuración

El diseño propuesto separa el aprendizaje en:

- **Etapa Difusa:** Un bloque de 25 reglas (5×5) que procesa x_1 y x_2 para modelar la sigmoide.
- **Etapa Lineal (Adaline):** Implementada mediante el modelo de **Takagi-Sugeno de orden 1**. Se utiliza una función de pertenencia `xfl.parametric` para definir los pesos ($w_{temp} = 1{,}0, w_{x3} = -0{,}5$) y el bias (0,0).

En las Figuras 15, 16 y 17 se muestra la salida del sistema antes del entrenamiento.

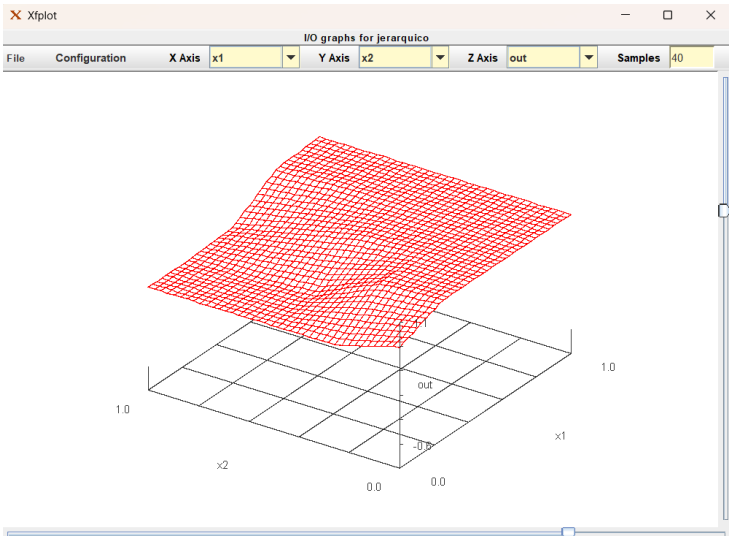

Figura 15: Representación del sistema jerárquico antes del entrenamiento supervisado (Ejes x1 y x2).

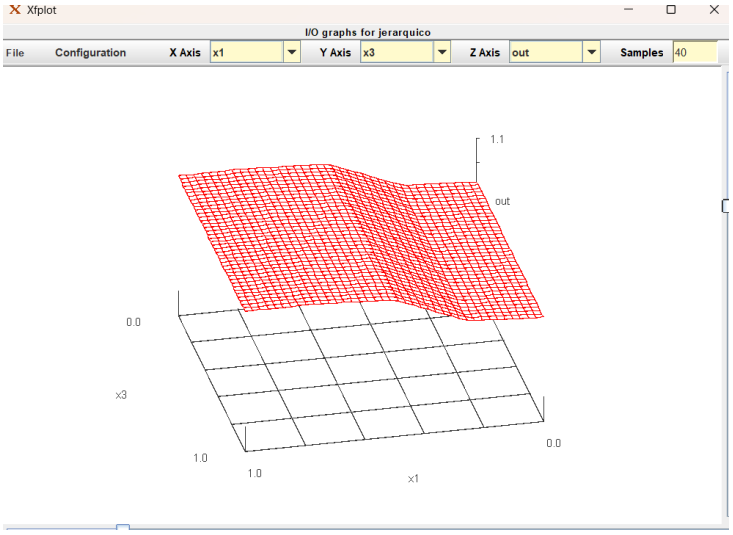

Figura 16: Representación del sistema jerárquico antes del entrenamiento supervisado (Ejes x1 y x3).

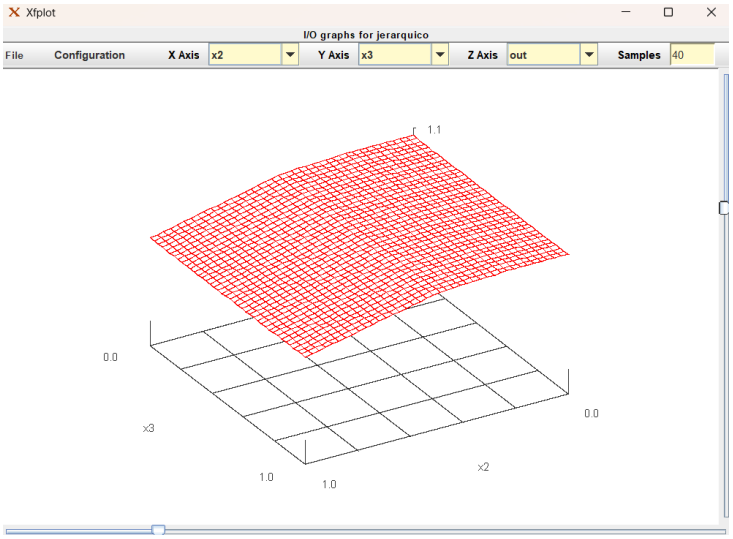

Figura 17: Representación del sistema jerárquico antes del entrenamiento supervisado (Ejes x2 y x3).

3.3.2. Resultados tras el entrenamiento

Tras aplicar el entrenamiento con Marquardt-Levenberg, el sistema jerárquico alcanzó un RMSE de 0.00265, logrando una precisión superior. A continuación, se presentan la curva de convergencia y las superficies finales para cada par de ejes:

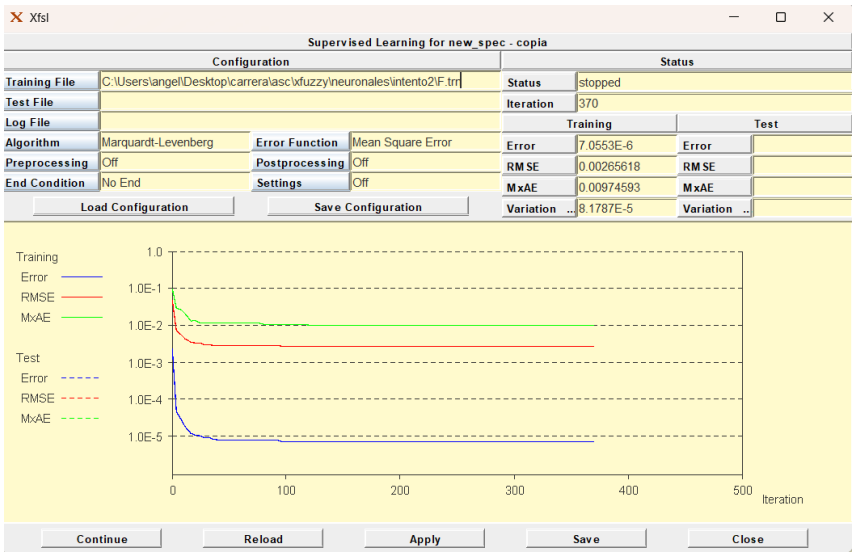

Figura 18: Curva de aprendizaje del sistema jerárquico (RMSE: 0.00265).

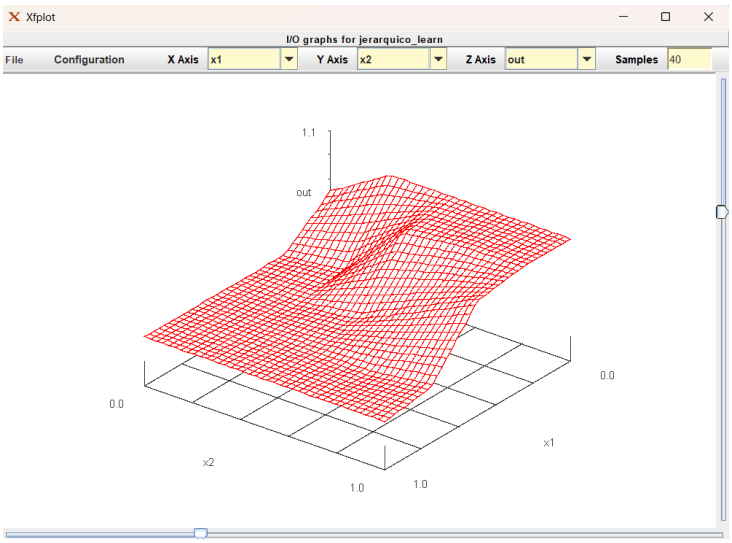

Figura 19: Superficie final del sistema jerárquico (ejes x1 y x2).

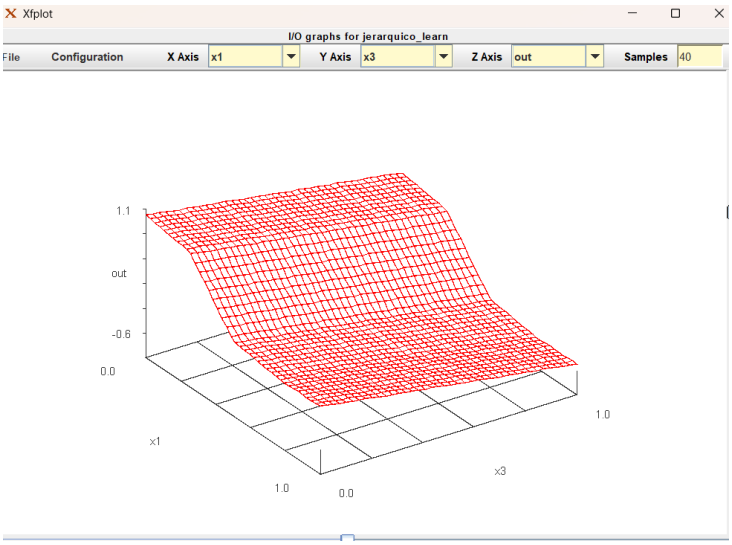

Figura 20: Superficie final del sistema jerárquico (ejes x1 y x3).

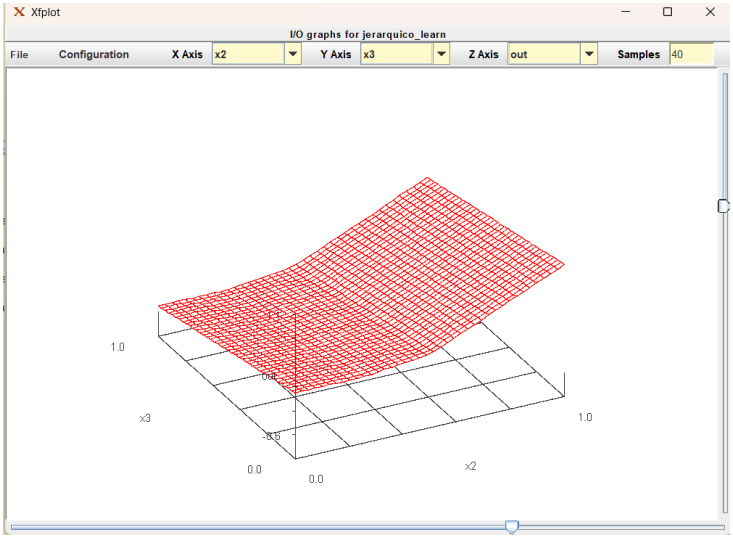

Figura 21: Superficie final del sistema jerárquico (ejes x2 y x3).

3.4. Conclusiones del Apartado 2

La comparativa confirma la superioridad de la arquitectura jerárquica. Se ha conseguido un error significativamente menor (RMSE 0.00265 frente a 0.0297) reduciendo la complejidad de 72 reglas a solo 26 reglas totales $(25 + 1)$. El uso de un Adaline para la componente lineal permite un aprendizaje más eficiente y preciso.

4. Apartado 3

En esta sección se aborda el diseño de una red neuronal lineal (Adaline) para la aproximación y predicción de una serie temporal. El objetivo es analizar cómo afectan la calidad y cantidad de datos al aprendizaje del sistema utilizando el error cuadrático medio (MSE) y su raíz (RMSE).

4.1. Generación de la Serie Temporal

Se ha definido una serie temporal sencilla basada en una función sinusoidal pura. La ecuación implementada para generar los datos es:

$$Y(t) = 10 \cdot \sin(0{,}1 \cdot t)$$

Siguiendo el enunciado, el sistema utiliza una estructura de ventana deslizante donde los valores pasados $Y(t-2)$ e $Y(t-1)$ actúan como entradas para predecir el valor actual $Y(t)$. Se han generado tres escenarios:

- **Normal:** 100 muestras limpias para un aprendizaje óptimo.
- **Ruido:** 100 muestras con ruido gaussiano para evaluar la robustez.
- **Escaso:** Solo 20 patrones para analizar la falta de información estadística.

4.2. Estructura del Sistema Adaline

El sistema se ha diseñado consta de dos variables de entrada $(x1, x2)$ que representan los retardos temporales y una variable de salida (`out`).

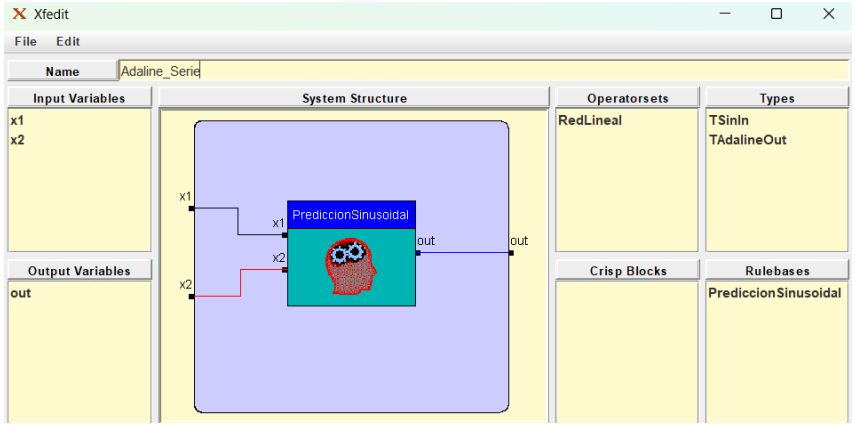

Figura 22: Arquitectura del sistema Adaline en Xfedit.

4.3. Entrenamiento Supervisado (Xfsl)

Se ha utilizado el algoritmo de Marquardt-Levenberg para ajustar los parámetros del Adaline.

4.3.1. Entrenamiento con Datos Normales

Con el conjunto `SIN_normal.trn`, el sistema alcanza una convergencia extremadamente precisa. El RMSE de entrenamiento es de $3{,}8845 \cdot 10^{-16}$ y el de test de $7{,}3925 \cdot 10^{-5}$.

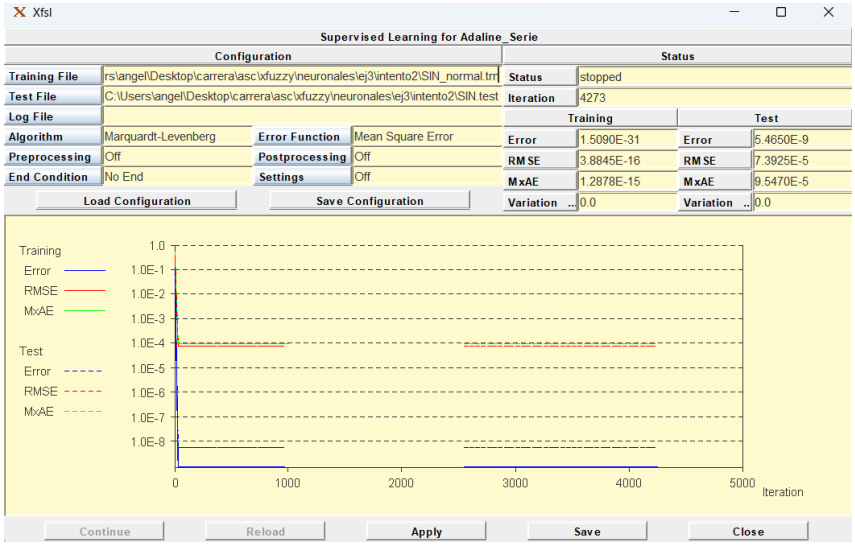

Figura 23: Curva de aprendizaje con datos normales (RMSE mínimo).

4.3.2. Entrenamiento con Datos con Ruido

Al introducir perturbaciones (`SIN_ruido.trn`), el error de entrenamiento aumenta considerablemente hasta un RMSE de 0,199463. Sin embargo, el error de test se mantiene bajo (0,0169641), lo que indica que el Adaline ha logrado extraer la tendencia lineal a pesar del ruido.

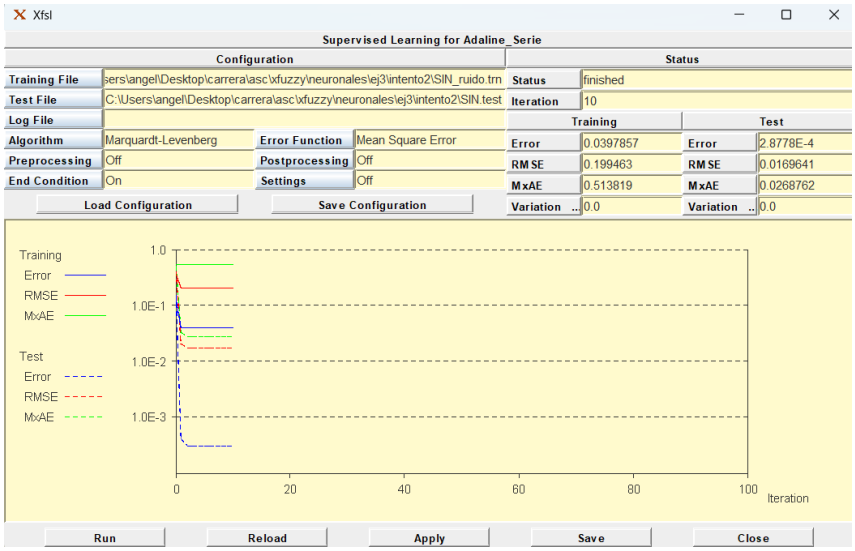

Figura 24: Efecto del ruido en el proceso de aprendizaje.

4.3.3. Entrenamiento con Datos Escasos

Con solo 20 puntos (`SIN_pocos.trn`), el sistema parece converger rápido (RMSE train $2{,}4699 \cdot 10^{-16}$), pero el error de test es el más alto de los tres casos sin ruido (0,0276694), demostrando una peor capacidad de generalización.

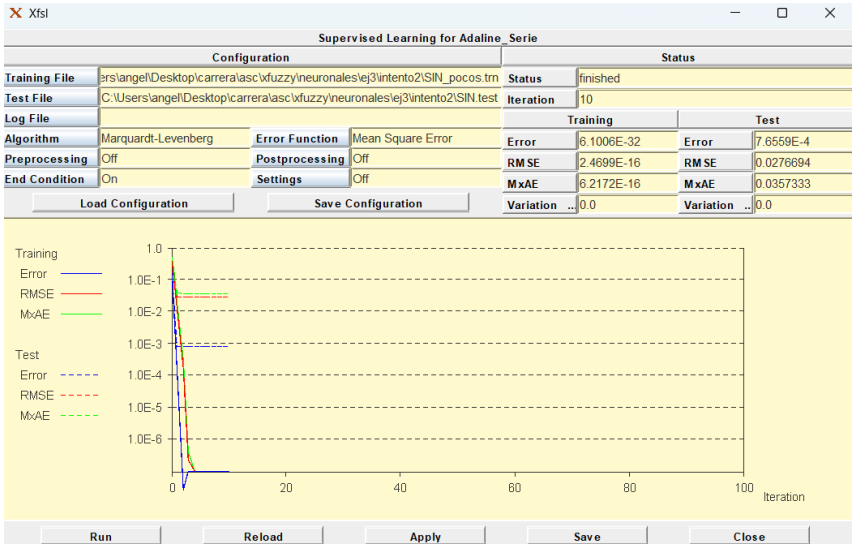

Figura 25: Resultado del entrenamiento con base de datos reducida.

4.4. Simulación y Verificación (Xfsim)

Se han realizado simulaciones dinámicas utilizando el modelo `ModeloSerieTemporal.class` para verificar la capacidad predictiva.

4.4.1. Validación del Modelo Normal

En la simulación del modelo normal, el sistema predice la serie con un comportamiento estable. Para $n = 100$, la salida se sitúa en $-0{,}5393$.

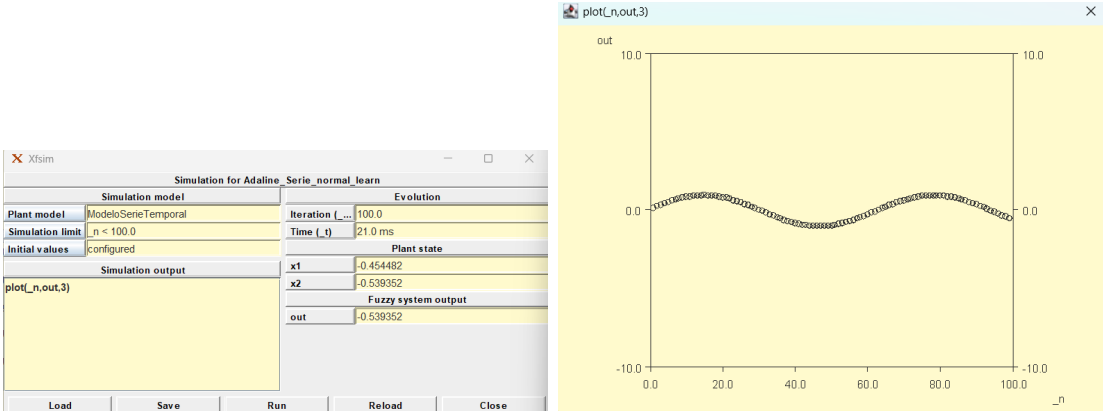

Figura 26: Simulación y gráfica del modelo entrenado con datos normales.

4.4.2. Validación del Modelo con Ruido

El modelo entrenado con ruido muestra una mayor desviación en la salida (0,5026 en $n = 100$) y una gráfica con una tendencia descendente que se aleja de la oscilación pura, reflejando la influencia del ruido en los pesos.

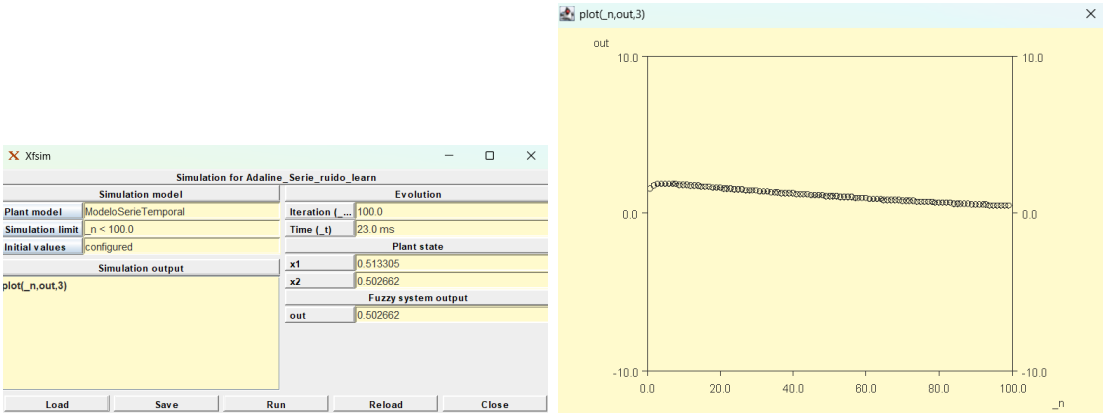

Figura 27: Estado y gráfica de salida para el modelo con ruido.

4.4.3. Validación del Modelo con Datos Escasos

Curiosamente, el modelo con datos escasos muestra visualmente una amplitud mayor en la simulación (salida $-5{,}5036$ en $n = 100$), siguiendo la forma senoidal pero con un desfase o error de amplitud respecto al objetivo teórico debido a la falta de muestras.

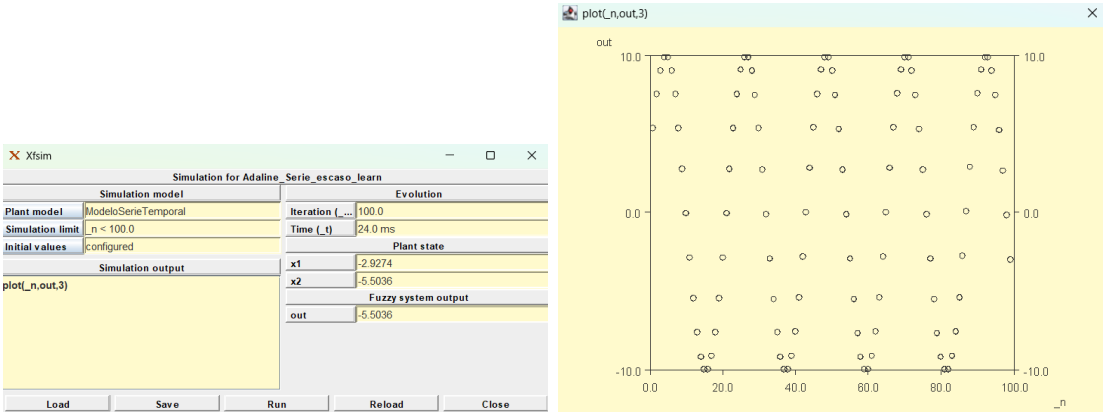

Figura 28: Simulación del modelo con datos escasos.

4.5. Conclusiones del Apartado 3

Tras el análisis del sistema `Adaline_Serie`, se extraen las siguientes conclusiones:

- **Precisión del Adaline:** El modelo lineal es excelente para aproximar funciones sinusoidales puras cuando los datos son abundantes y limpios, alcanzando errores de test del orden de 10^{-5}.
- **Robustez frente al Ruido:** El sistema es capaz de aprender la señal principal incluso con ruido, aunque esto penaliza el RMSE de entrenamiento y deforma la predicción a largo plazo en la simulación dinámica.
- **Importancia del volumen de datos:** La reducción de patrones (caso escaso) provoca que el sistema no ajuste correctamente la amplitud o fase de la señal, confirmando que la densidad de muestras en el fichero `.trn` es crítica para la fidelidad del modelo.