Resolución TP Programación 2

ADALINE – Series de Fourier - Polinomios

Alumno: Salomón Nicolás

Curso: Inteligencia Artificial aplicada a la Identificación y Control

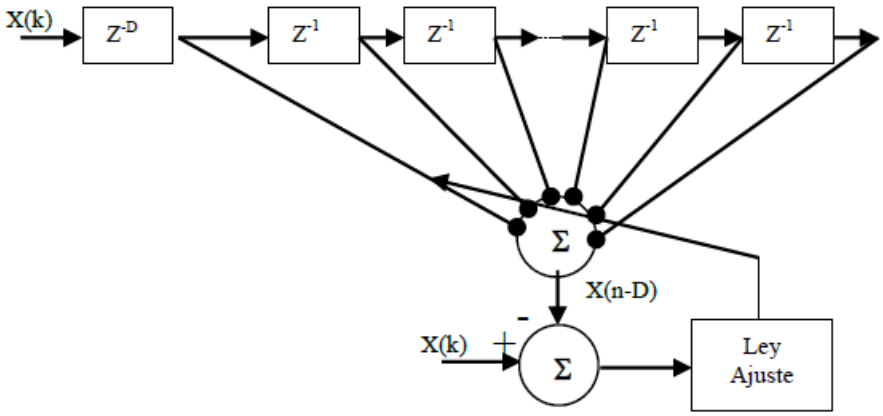
Profesor: Dr. Ing. H. Daniel Patiño

Año: 2022

1. **Diseñar un Predictor basado en ADALINE capaz de estimar la predicción en (3 y 5 pasos de muestreo) de una señal modulada en frecuencia del tipo:**

$$x(n) = \sin\left(n + \sin(n^2)\right)$$

Realizar simulaciones y reportar los resultados. Emplear cualquier ley de aprendizaje supervisado vista.



El código de resolución se encuentra adjunto en el archivo *Aproximacion.m*

Como primer paso se procede a graficar la señal modulada en frecuencia a aproximar:

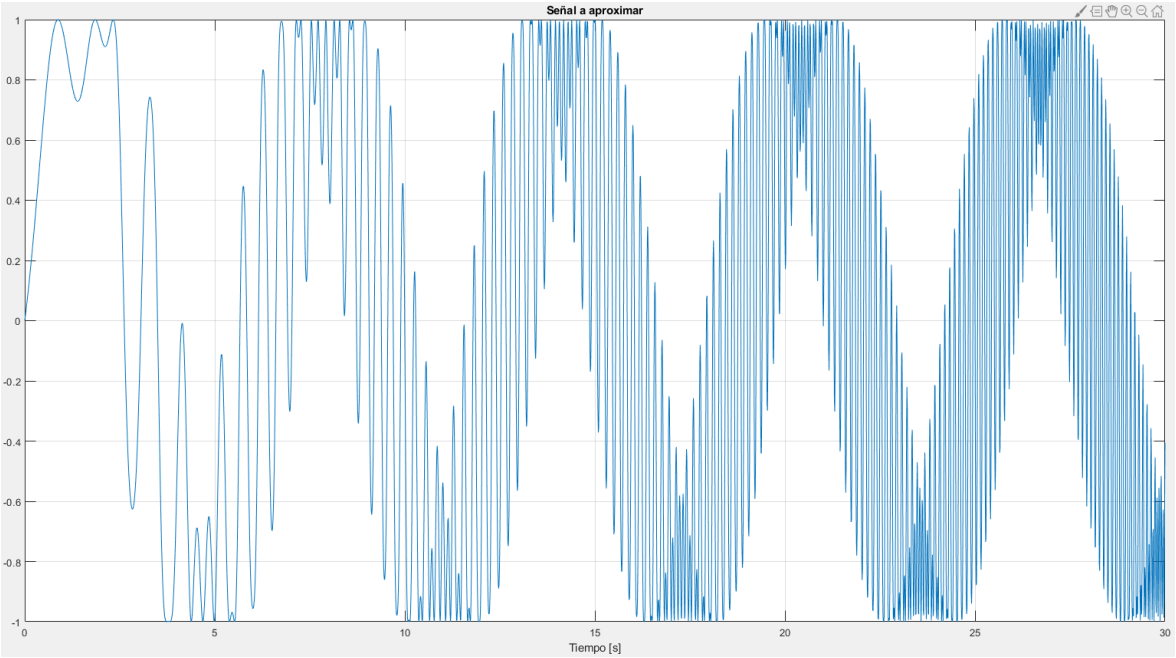


Figura 1. Señal original a aproximar.

Como paso siguiente, se procede a construir la red ADALINE con la estructura dada anteriormente.

La generación de los retardos de la red se llevó a cabo mediante la técnica de la convolución (convmtx (x, p) en Matlab, donde x es la señal de entrada y p es la cantidad de retardos). Por ejemplo, al realizar convmtx (1:8,3) obtenemos el siguiente resultado:

```
>> convmtx(1:8,3)

ans =

     1     2     3     4     5     6     7     8     0     0
     0     1     2     3     4     5     6     7     8     0
     0     0     1     2     3     4     5     6     7     8
```

Se observa como cada fila tiene la señal de entrada (en este ejemplo un vector de 1 a 8) desplazada un lugar con respecto al anterior. Por último, solo basta con quedarnos con todas las filas y las n-ésimas columnas (con n desde 1 a la longitud de la señal original).

El resultado obtenido al emplear 3 pasos de muestreo se observa en la Figura 2.

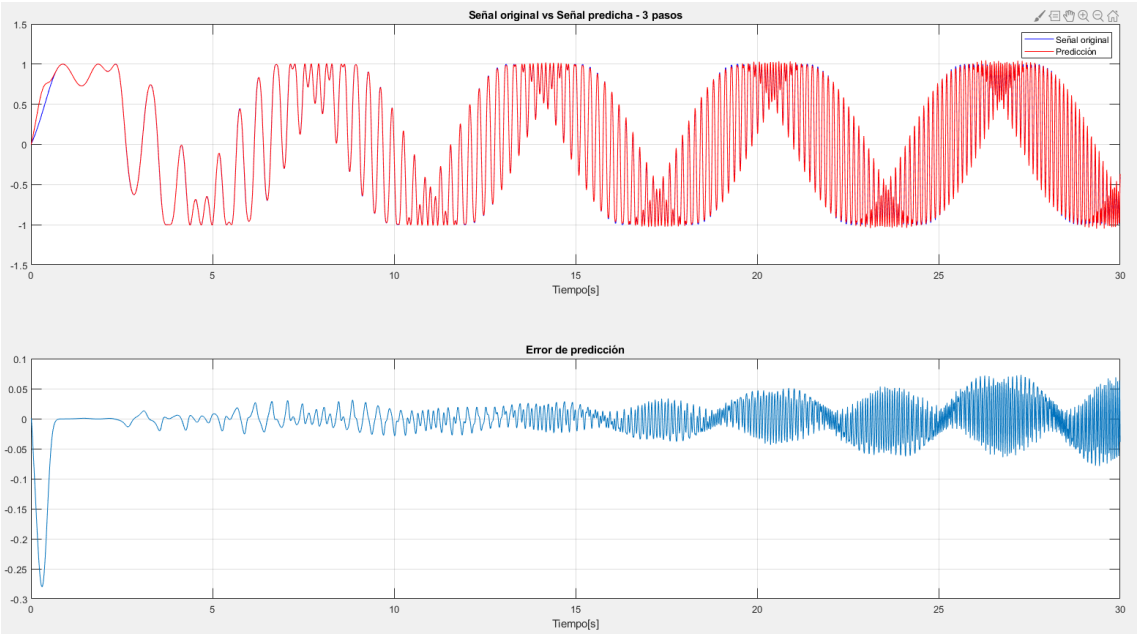


Figura 2. Señal predicha con 3 pasos de muestreo.

Se observa un pequeño error al comienzo de la predicción, pero luego el mismo disminuye hasta lograr predecir la señal casi perfectamente.

Como siguiente paso, se procede a aumentar el número de pasos de muestro a 5, obteniendo el resultado presentado en la Figura 3.

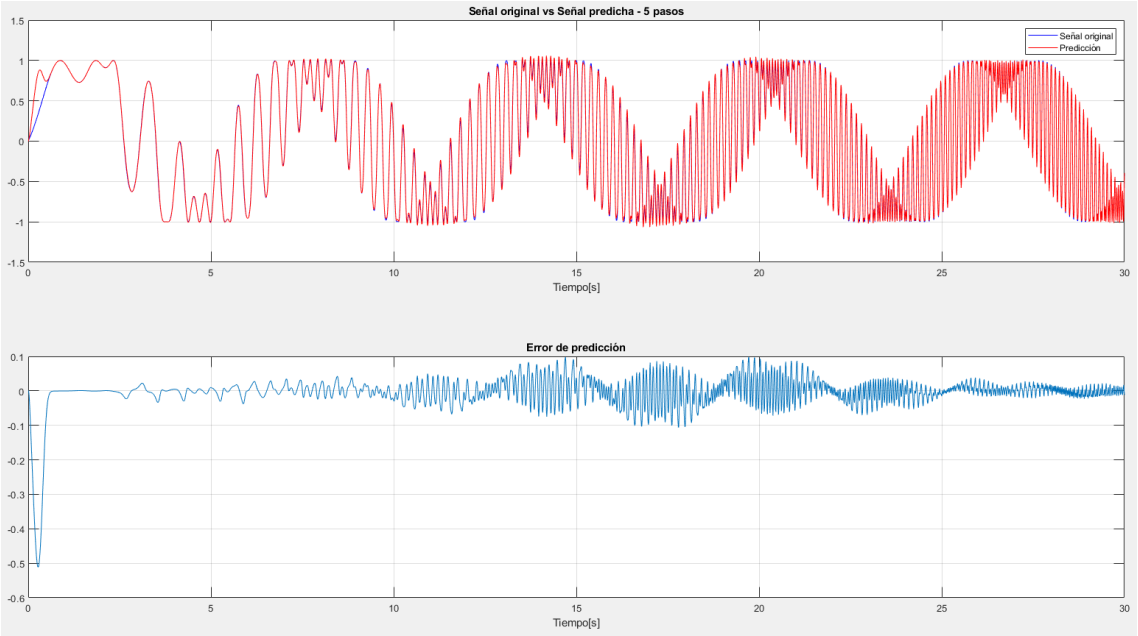


Figura 3. Señal predicha con 5 pasos de muestreo.

Se observa el mismo comportamiento que empleando 3 pasos de muestreo, con la diferencia de que el error disminuye al emplear un mayor número de pasos de muestreo, dado que la red ADALINE posee un mayor número de muestras pasadas, y por lo tanto mayor información, para predecir la señal futura.

2. **Identificar un sistema lineal (elegido arbitrariamente) basado en la ADALINE. Emplear al menos 2 leyes de aprendizaje y compararlas en término de velocidad de aprendizaje.**

El código de resolución se encuentra adjunto en el archivo *Identificacion.m*

Como primer paso, se procede a generar una señal aleatoria de entrada, según se muestra en la Figura 4.

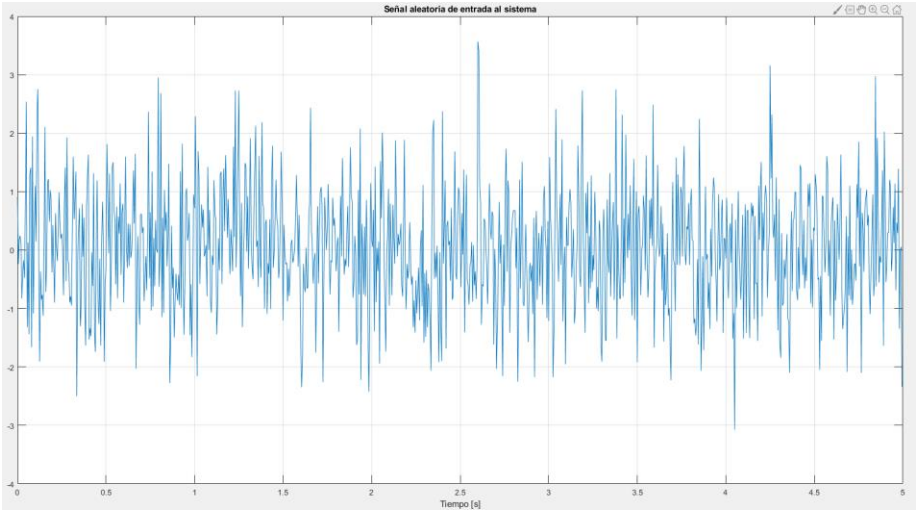


Figura 4. Señal de entrada al sistema desconocido.

El proceso de identificación del sistema se aprecia en la Figura 5.

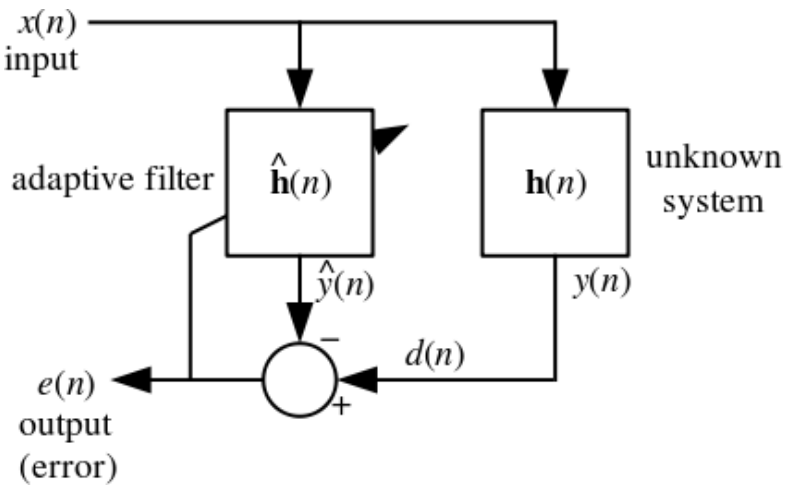


Figura 5. Red de identificación del sistema.

Como paso siguiente, consideraremos un sistema que posee tres parámetros b = [0.8 -0.2 0.5], dichos parámetros son desconocidos a priori y se buscará que la red estime los mismos. Dichos parámetros *b* se emplean para filtrar la señal de entrada generada anteriormente y obtener así, la salida del sistema desconocido. Luego, ingresan a la red ADALINE la señal de entrada junto con la salida de nuestro sistema desconocido.

Para realizar la identificación se empleó una red ADALINE con 3 pasos de muestreo y el algoritmo de Mínimos Cuadrados (LMS, Least Mean Squares). El tiempo total requerido para el entrenamiento fue de 0.02 [s] aproximadamente, y los resultados del mismo, junto con su error se pueden apreciar en las Figuras 6 y 7 respectivamente.

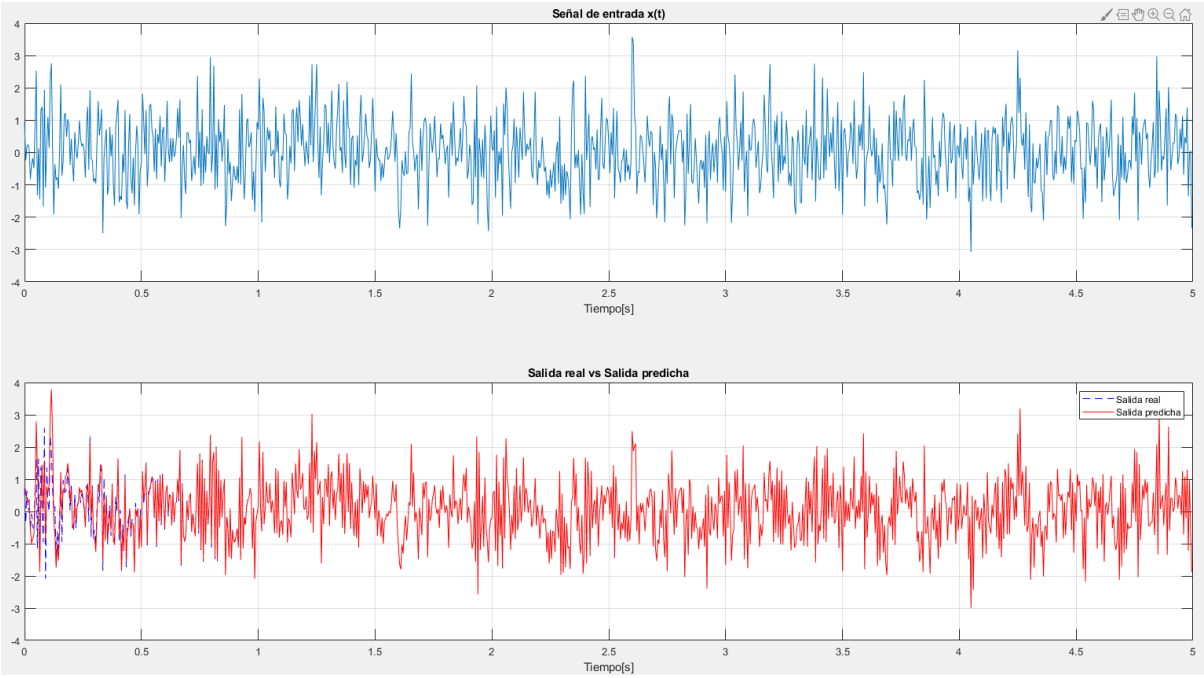


Figura 6. Señal de entrada vs Señal predicha con LMS.

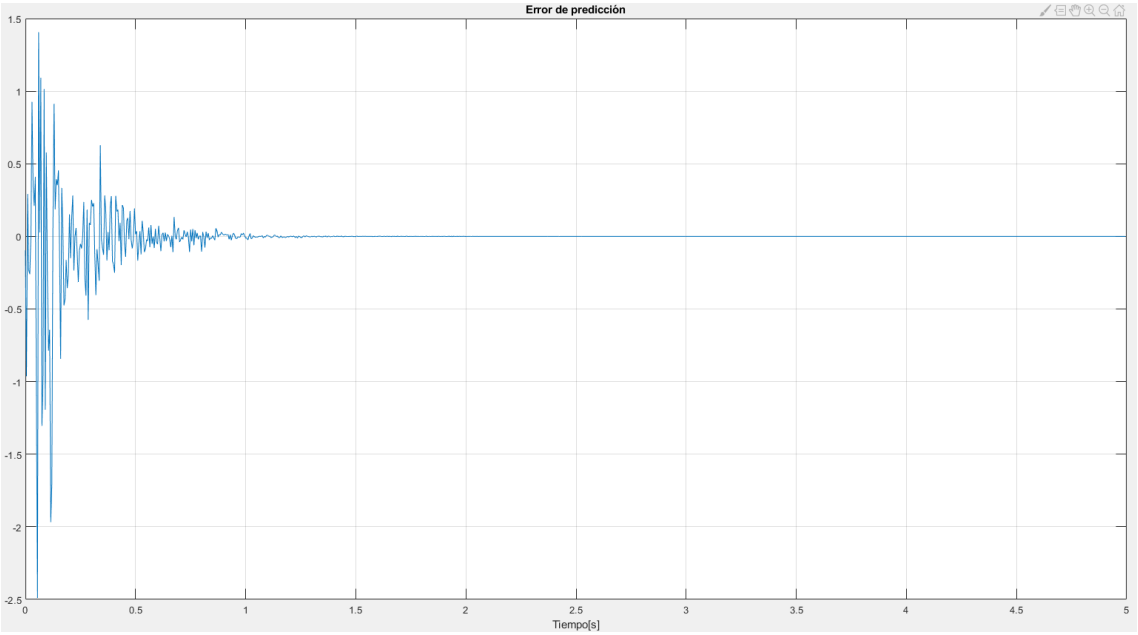


Figura 7. Error de predicción con LMS.

Por último, los parámetros reales vs los obtenidos por el sistema con ADALINE fueron:

```
Parametros reales durante t:
    0.8000   -0.2000    0.5000

Parametros predichos durante t:
    0.8000   -0.2000    0.5000
```

Se observa una identificación perfecta de los parámetros de nuestro sistema desconocido.

Para realizar una comparación empleando otro algoritmo de aprendizaje, entrenaremos la misma red ADALINE, con los mismos datos de entrada, empleando el algoritmo de Levenberg-Marquardt. Los resultados pueden apreciarse en las Figuras 8 y 9, mientras que el tiempo total de simulación fue de 0.85[s].

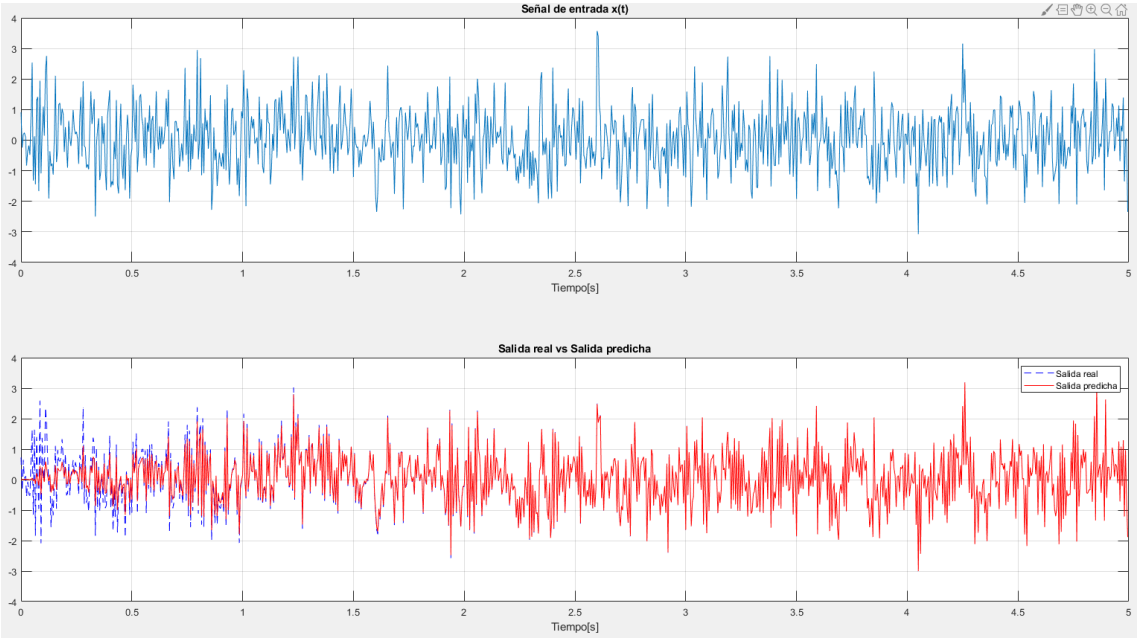


Figura 8. Señal real vs Señal predicha con Levenberg-Marquardt.

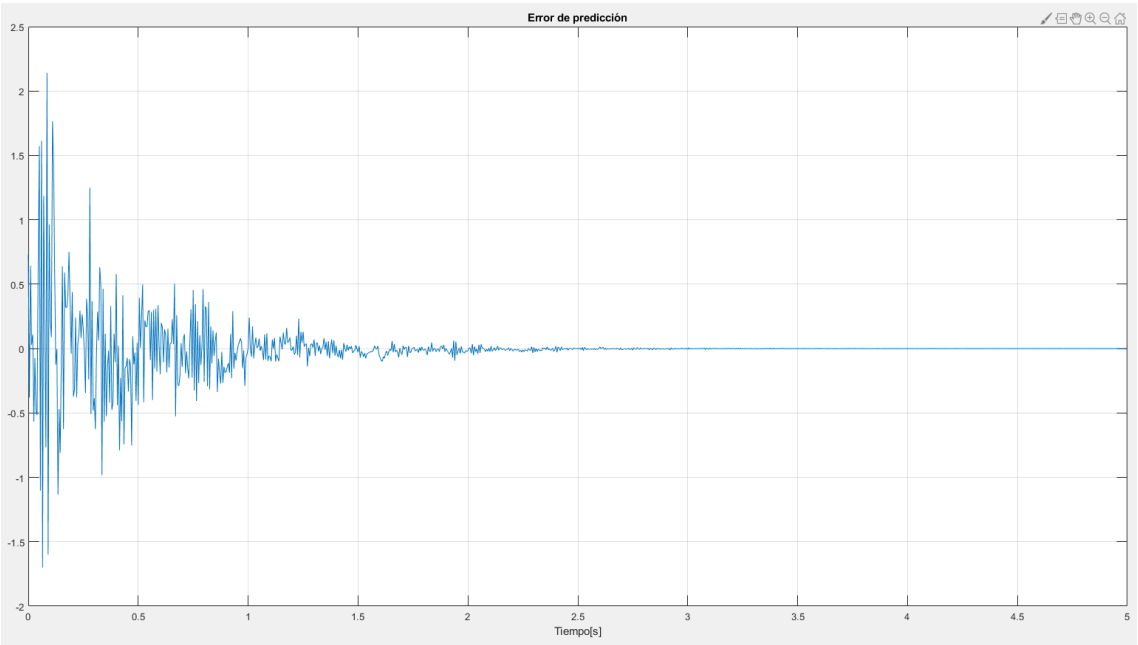


Figura 9. Error de predicción con Levenberg-Marquardt.

Por último, los parámetros reales vs los obtenidos por el sistema con ADALINE y Levenberg-Marquardt fueron:

```
Parametros reales durante t:
    0.8000   -0.2000    0.5000

Parametros predichos durante t:
    0.8000   -0.1999    0.5000
```

Al igual que el entrenamiento con LMS se observa una identificación perfecta de los parámetros del sistema desconocido, la principal diferencia radica en el tiempo necesario para la misma. Para esta simple demostración, el algoritmo LMS fue mucho más rápido que el algoritmo Levenberg-Marquardt, debido al poder de cómputo necesario para ejecutar este último.

3. **Diseñar un Filtro Supresor de Ruido basado en la ADALINE de una señal "inteligente" de tipo senoidal perturbada por ruido. Suponer poder medir la señal ruidosa.**

$$\boldsymbol{f(x) = \sin(2 * \pi * f * t) + [3 * (rand - 1)]}$$

El código de resolución se encuentra adjunto en el archivo *Noise_cancelling.m*

En primer lugar, se procede a generar y graficar la señal pedida, cuyo resultado se observa en la Figura 10.

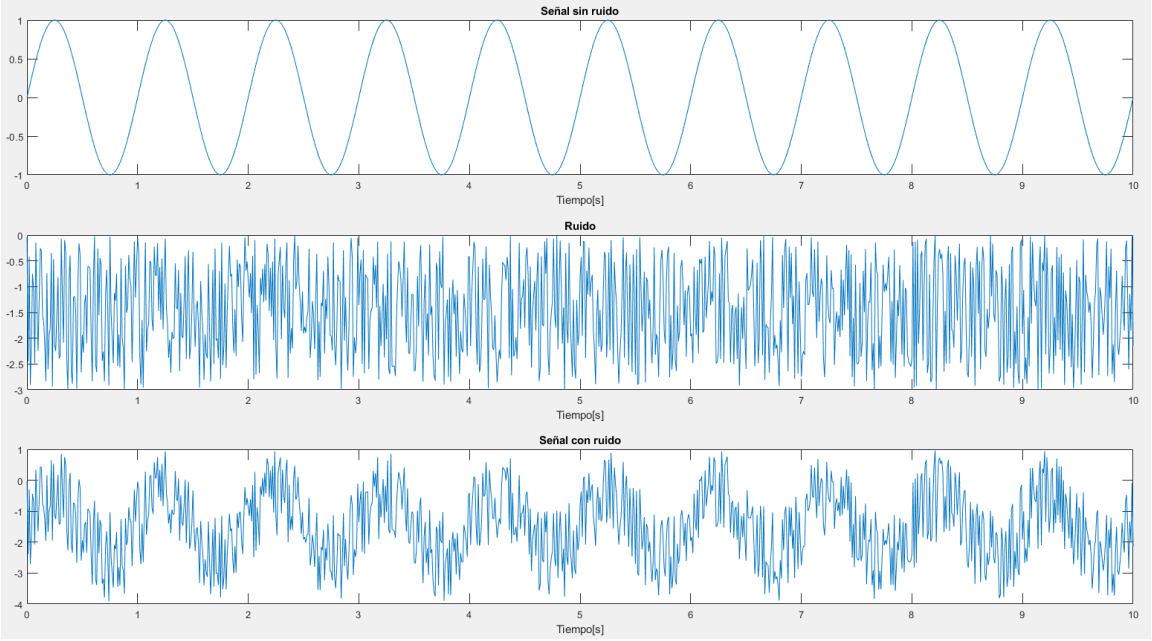


Figura 10. Señales de entrada.

Luego, se define la arquitectura de la red a emplear para filtrar el ruido presente en la señal. Dicha arquitectura se observa en la Figura 11, donde x(n) es la señal con ruido, mientras que n(n) es el ruido de la señal.

La idea de dicha arquitectura es que la red ADALINE pueda predecir el ruido presente en la señal para restarlo de la misma y así obtener la señal filtrada e(n).

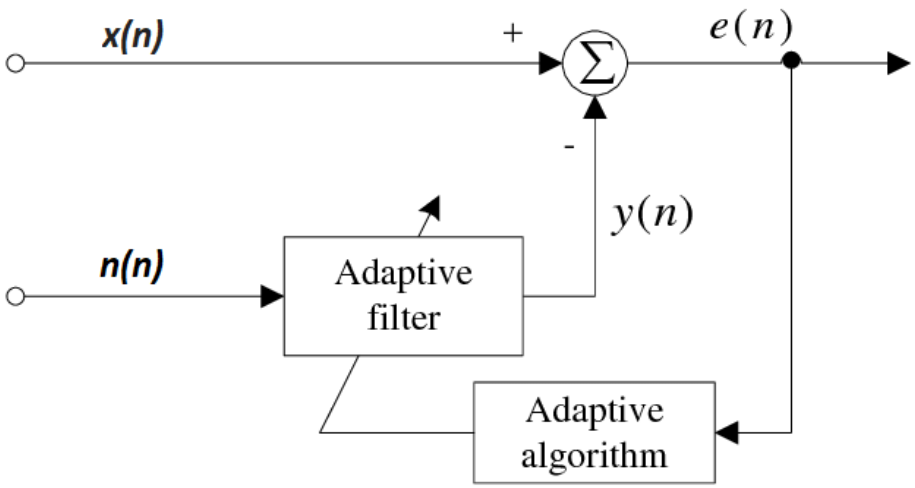


Figura 11. Arquitectura del Filtro Supresor de Ruido con ADALINE.

Se procedió a crear la red ADALINE empleando un único retardo (experimentalmente dicho número de retardo entregó los mejores resultados). Los resultados de la misma se pueden apreciar en la Figuras 12 y 13.

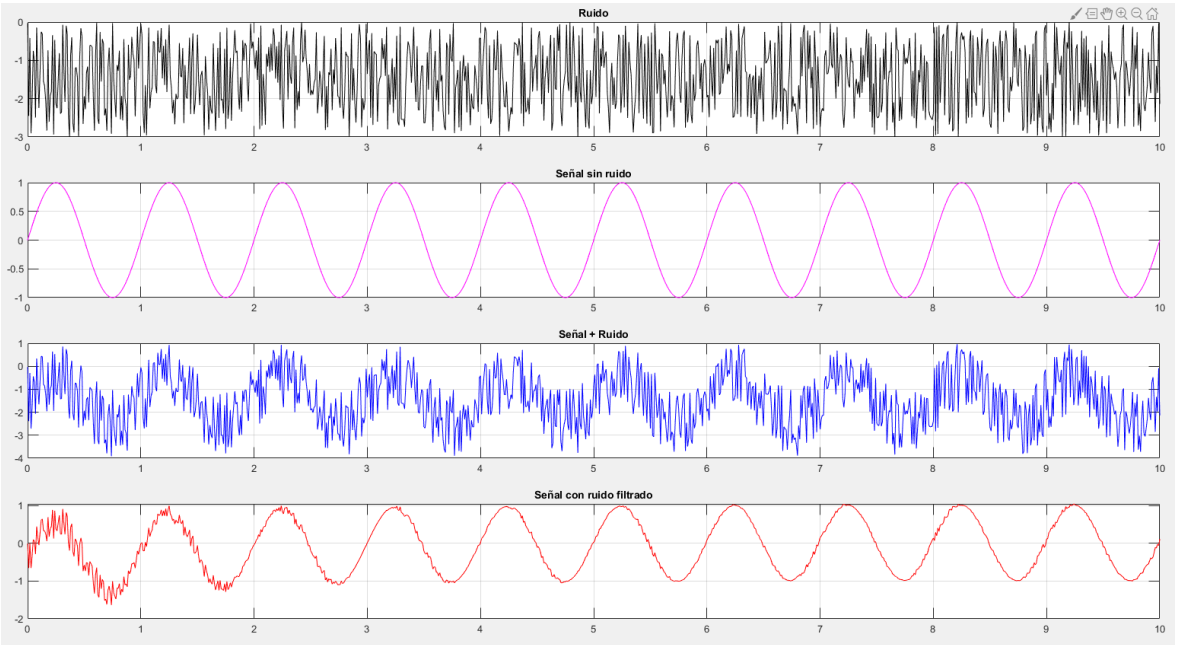


Figura 12. Resultado del filtrado de ruido con ADALINE.

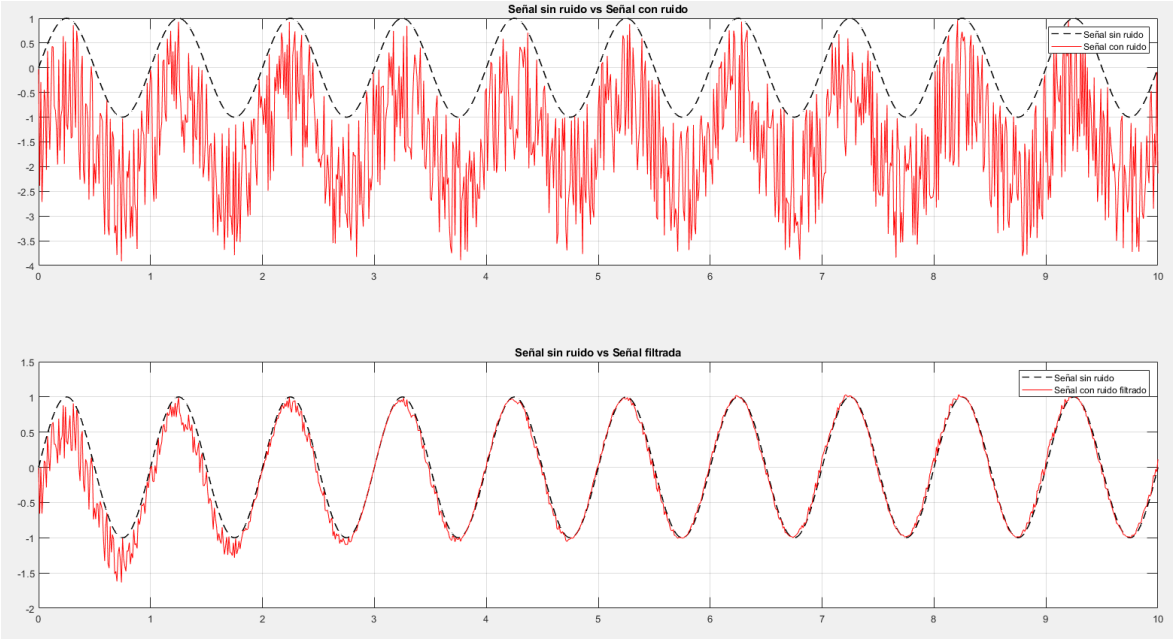


Figura 13. Señal con ruido vs Señal filtrada.

Se observa un óptimo resultado en el filtrado del ruido de la señal. Es importante destacar que la red logró filtrar no solamente la componente variacional del ruido, sino que también filtró la componente de continua que poseía el mismo, como puede apreciarse en la Figura 13. Al comenzar el filtrado la red no brinda un buen resultado ya que la misma necesita un tiempo de adaptación de sus parámetros (iniciados aleatoriamente), pero al superarse este tiempo, la aproximación es casi perfecta.

4. **Aproximar una función f(x), x ϵ [-2, 2] mediante una serie de Fourier y mediante un Polinomio de un orden n basado en una Ley de Aprendizaje Supervisado:**

$$\boldsymbol{f1(x) = 2 * x + 0.8 * x^3}$$

$$\boldsymbol{f2(x) = \frac{x}{1 + x^2} + 0.1 * \tanh(x)}$$

El código de resolución se encuentra adjunto en el archivo *TP_Programación_2 - Red de Fourier.ipynb* (Jupyter Notebook en lenguaje Python).

Aproximación mediante una Serie de Fourier

En primer lugar, aproximaremos la función f(x) empleando una Serie de Fourier de la forma:

$$f(x) = a_0 + \sum_{n=1}^{m} c_n * \sin(n * x + \theta_n)$$

Según la teoría matemática por detrás de la Serie de Fourier, cualquier función f(x) puede aproximarse a través de dicha serie, es decir, mediante una suma de señales senoidales.

En nuestro desarrollo, la función real será la dada por el enunciado, mientras que la red neuronal estimará los coeficientes de la Serie de Fourier que representa a la función f(x).

La arquitectura de la red neuronal a emplear (proveniente del análisis de la Serie de Fourier) será la siguiente:

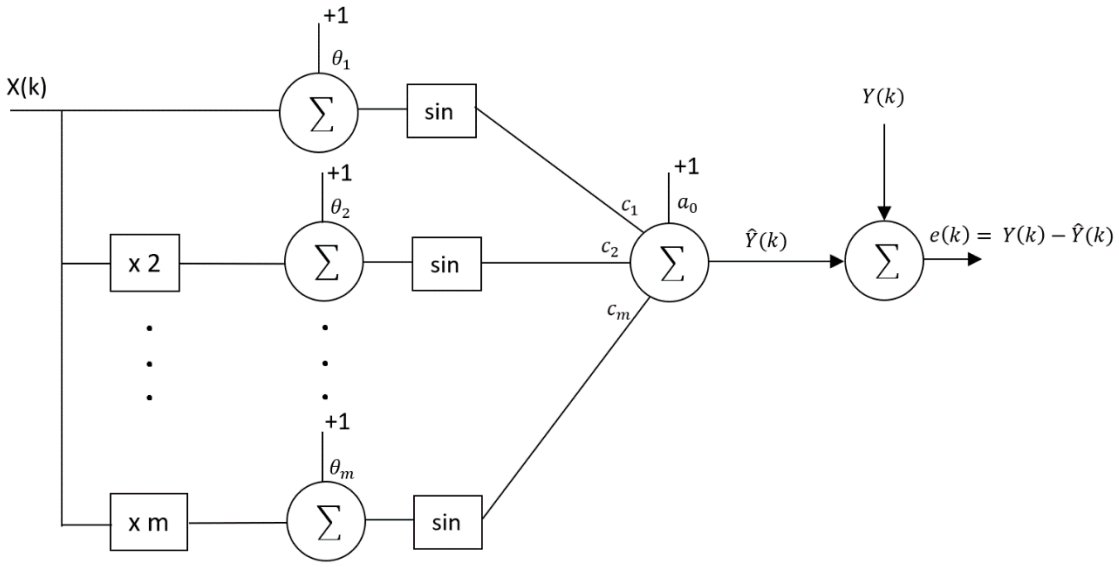


Figura 14. Arquitectura de la Red Neuronal.

En cuanto a la Ley de Aprendizaje empleada, se usó el Back-propagation para actualizar los pesos de la red, y se incluyó un límite de error para detener el entrenamiento de la red, una vez que dicho límite sea alcanzado.

Se realizó la aproximación de la función polinomial f1(x) empleando una Serie de Fourier variando el número de coeficientes. Para 3 coeficientes, junto con un learning rate = 0.01 y un umbral de error de 1e-6, se obtuvo el resultado de las Figuras 15 y 16 con 2081 iteraciones, mientras que, para 5 coeficientes, y el mismo learning rate y umbral de error, se obtuvo el resultado de las Figuras 17 y 18 en 1346 iteraciones.

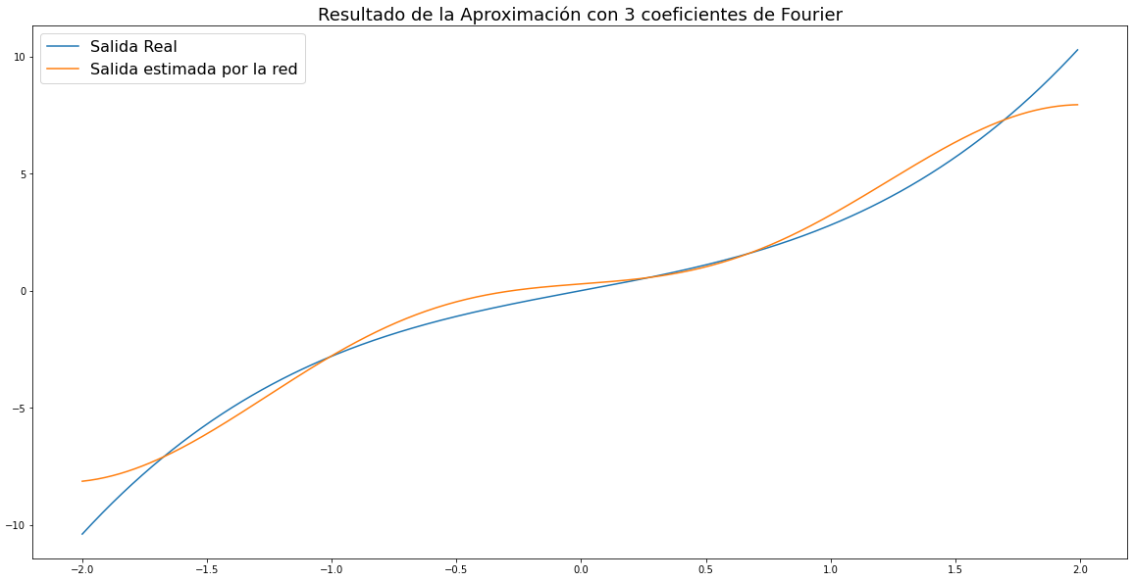


Figura 15. Aproximación por Serie de Fourier para 3 coeficientes.

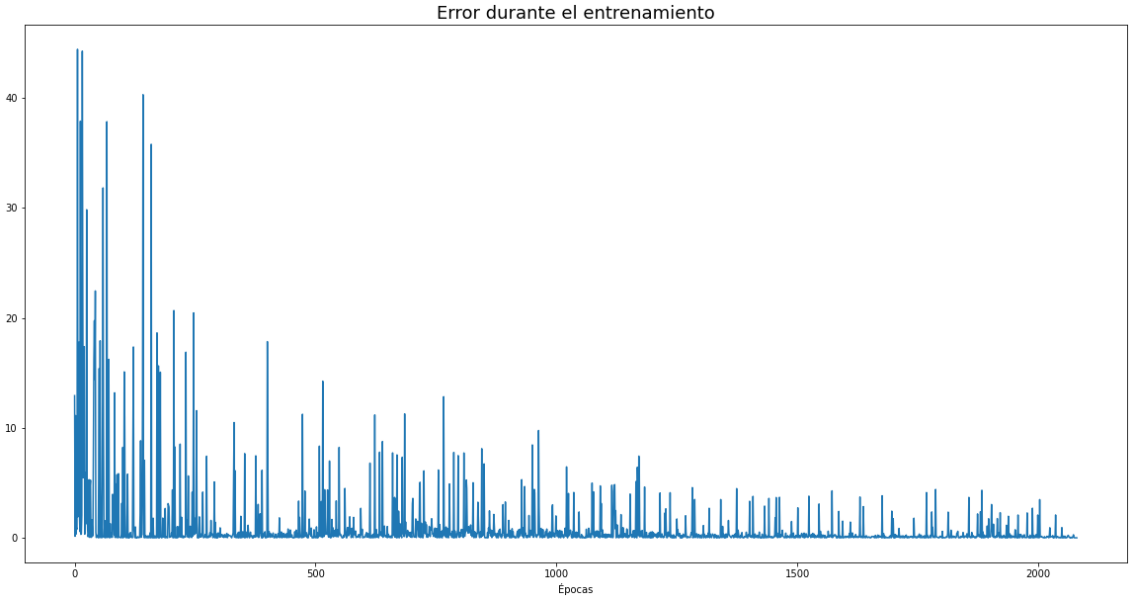


Figura 16. Error en aproximación por Serie de Fourier para 3 coeficientes.

Coeficientes:

- **cn = [6.55507889, -2.73682561, -0.09122486]**
- **a0 = [-0.02602577]**
- **θ_n = [0.06237493, 0.04270001, -0.22870773**

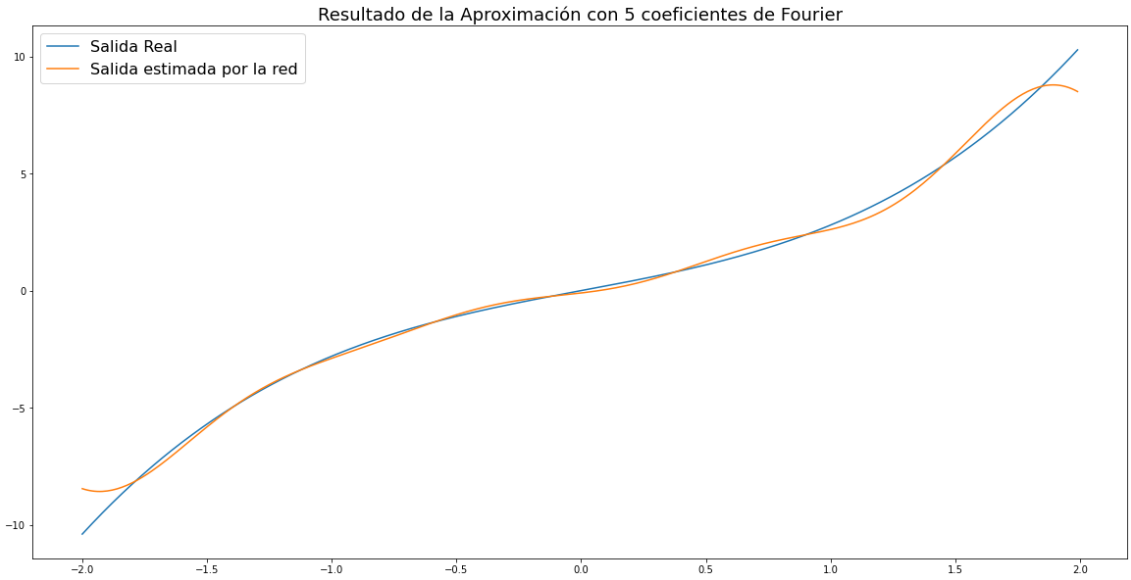


Figura 17. Aproximación por Serie de Fourier para 5 coeficientes.

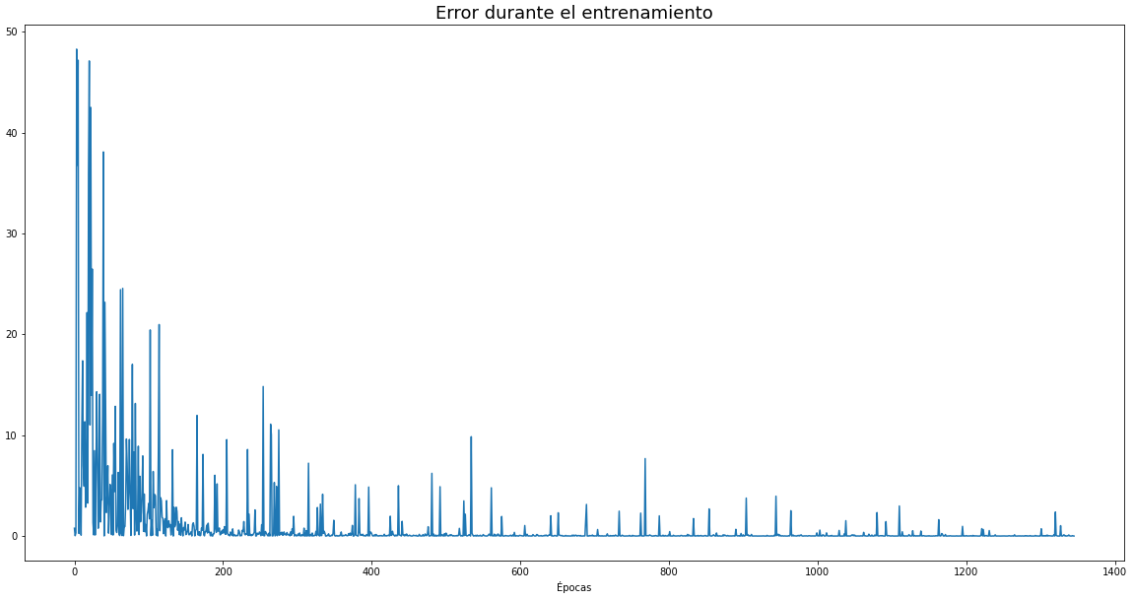


Figura 18. Error en aproximación por Serie de Fourier para 5 coeficientes.

Coeficientes:

- **cn = [5.25447144, -0.71085711, -2.17322497, 2.09475023, -0.91346795]**
- **a0 = [-0.20309184]**
- **θ_n = [0.06140236, 0.25503353, -0.02587187, 0.05195472, 0.22718683]**

Se observa como al aumentar el número de coeficientes, se logra una mejor aproximación de la señal, mientras que el error también es menor para cada época.

Luego, se realizó la aproximación de la función polinomial f2(x) empleando una Serie de Fourier variando el número de coeficientes. El mejor resultado, obtenido experimentalmente, se obtuvo con 5 coeficientes, junto con un learning rate = 0.1 y un umbral de error de 1e-9, se obtuvo el resultado de las Figuras 19 y 20 con 238 iteraciones.

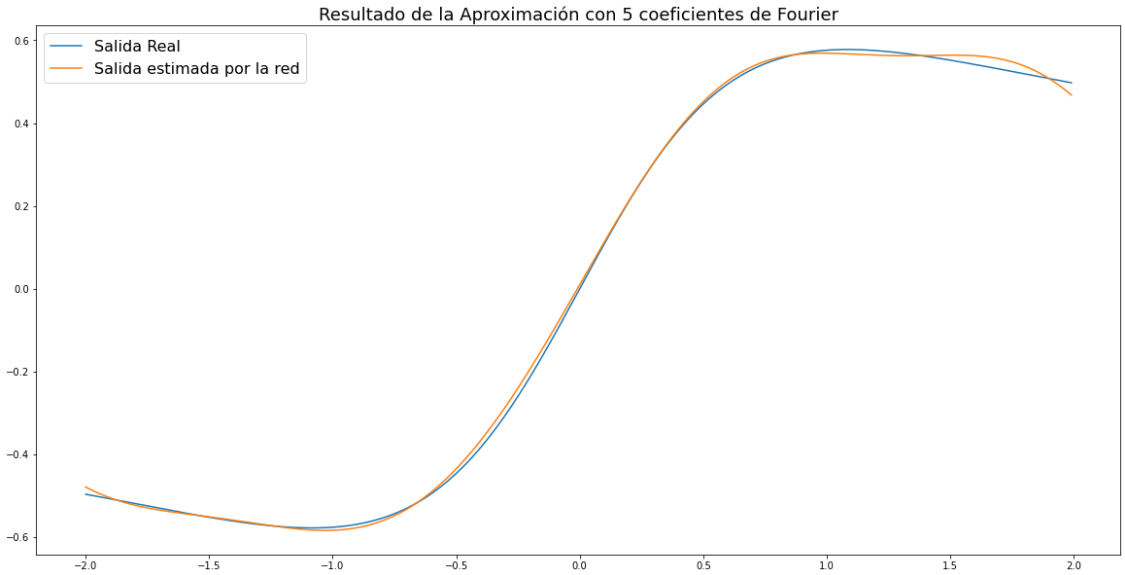


Figura 19. Aproximación por Serie de Fourier para 5 coeficientes

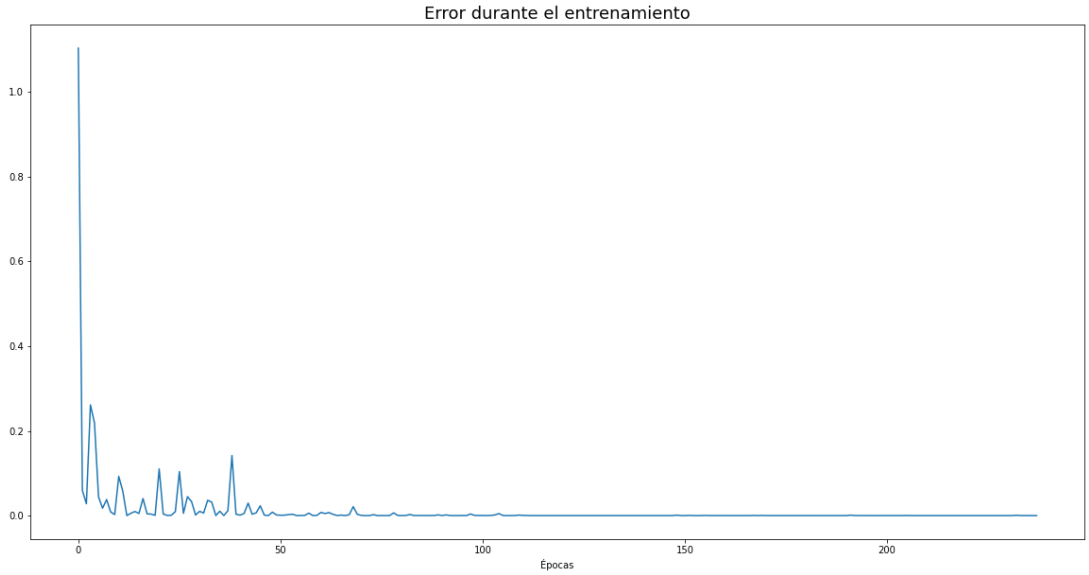


Figura 20. Error en aproximación por Serie de Fourier para 5 coeficientes.

Coeficientes:

- **cn = [0.56635033, 0.14825699, -0.00176923, 0.06332412, -0.01600443]**
- **a0 = [-0.00765099]**
- **θ_n = [0.03143701, -0.02049693, 0.7577387, 0.21673593, 0.65747743]**

Se observa una convergencia casi perfecta, en un menor tiempo de entrenamiento, con respecto a la señal f1(x) analizada anteriormente.

Aproximación mediante una función polinomial

De acuerdo al desarrollo matemático, sabemos que podemos aproximar una determinada función empleando otra función polinomial de un orden *n* tal que:

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n$$

Por lo tanto, la idea del desarrollo es emplear una red neuronal para aproximar los parámetros a_n de la función polinomial, para ello se empleó una red neuronal con la siguiente arquitectura:

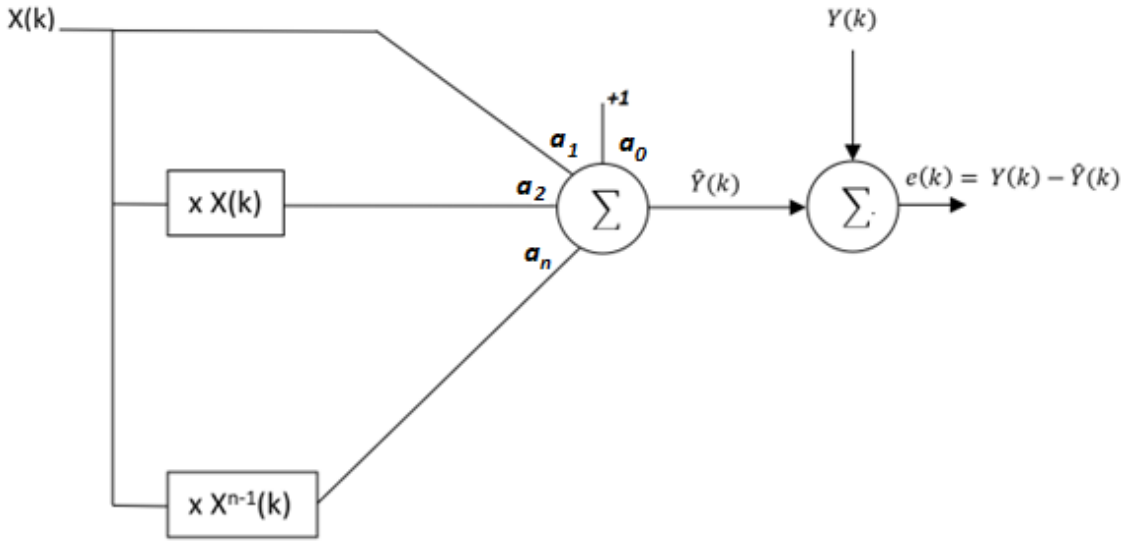


Figura 21. Arquitectura empleada para la aproximación por Polinomio.

Con el objetivo de aprovechar las funciones de optimización y leyes de aprendizaje más modernas, así como también evaluar su desempeño, se entrenó la red neuronal empleando el algoritmo de optimización Adam, con un learning rate de 1e-3, por 200 épocas para ambas funciones a aproximar.

En base a lo descripto anteriormente se aproximó la función $f1(x) = 2 * x + 0.8 * x^3$, con 3 coeficientes, obteniendo un error igual a 3,83e-13, y cuyo resultado se observa en la Figura 22.

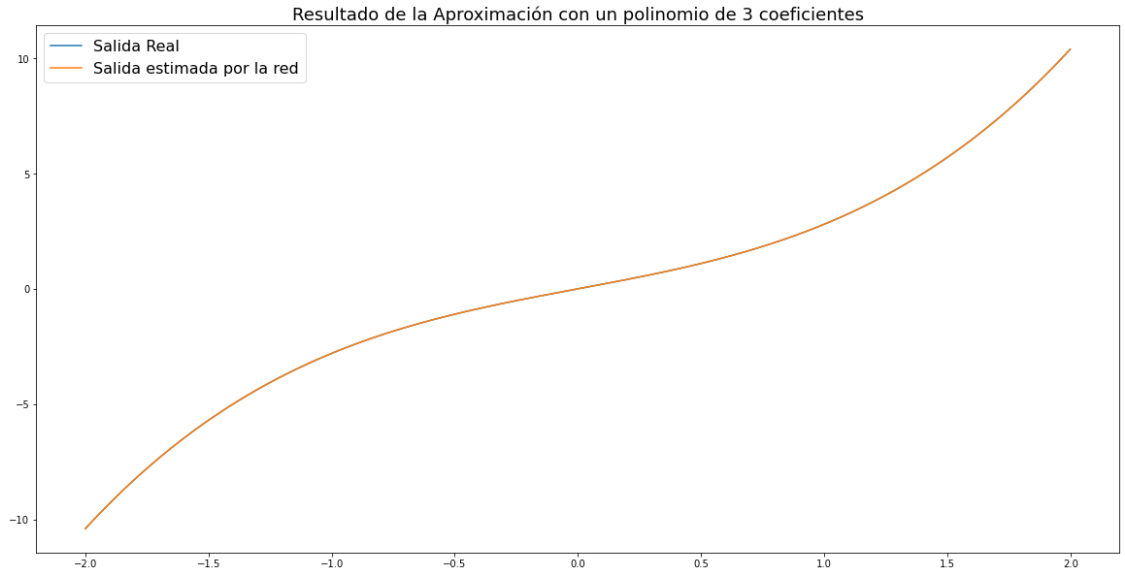


Figura 22. Resultado de la aproximación con 3 coeficientes.

Además, los coeficientes o pesos finales de la red fueron:

- **a_0 = -4.4048794e-02**
- **a_1 = 1.9999988e+00**
- **a_2 = 1.2535152e-07**
- **a_3 = 8 .0000037e-01**

Dichos coeficientes coinciden casi de forma exacta con los coeficientes reales dado es la ecuación de f1(x), demostrado además en la Figura 22.

De la misma forma se aproximó la función f2(x), obteniendo el mejor resultado al emplear 5 coeficientes, logrando un error de 6,65e-4. Los resultados de dicha aproximación se observan en la Figura 23.

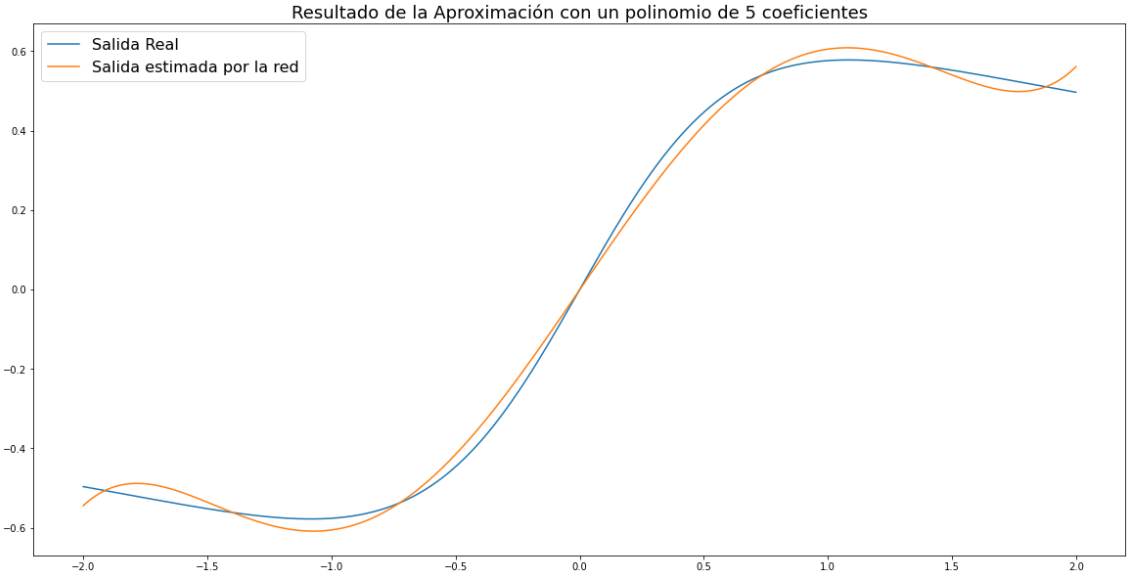


Figura 23. Resultado de la aproximación con 5 coeficientes.

Los coeficientes o pesos finales de la red fueron:

- **a_0 = 1.1232210e-01**
- **a_1 = 9.1563761e-01**
- **a_2 = -5.9719809e-05**
- **a_3 = -3.6047634e-01**
- **a_4 = 5.9540535e-04**
- **a_5 = 5.0171006e-02**

Se observa una aproximación bastante correcta considerando la complejidad de la función original f2(x).