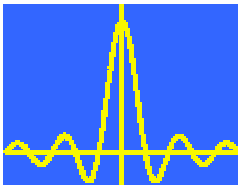
# INGENIERÍA DE TELECOMUNICACIONES

## DEPARTAMENTO DE INGENIERIA ELECTRÓNICA

## ÁREA DE TEORÍA DE LA SEÑAL Y COMUNICACIONES



Proyecto Fin de Carrera

# “FILTROS ADAPTATIVOS POLINOMIALES”

Realizado por: **Pedro Pérez Peña**

Tutor: **José Ignacio Acha Catalina**

# ***INDICE***

# CAPÍTULO 1.

# INTRODUCCIÓN

En este proyecto vamos a hablar sobre filtros adaptativos no lineales provistos con modelos polinomiales para la no linealidad. En estos sistemas no lineales, se pueden relacionar la salida y la entrada a través de una expansión en serie de Volterra o con una ecuación en diferencias no lineal y recursiva. La expansión en serie de Volterra puede modelar un amplio abanico de sistemas no lineales y es interesante en aplicaciones de filtros adaptativos porque dicha expansión es una combinación lineal de funciones no lineales de la señal de entrada. Las ecuaciones en diferencias recursivas y no lineales son interesantes porque son capaces de aproximar muchos sistemas no lineales con el uso de muy pocos coeficientes. También se verá lo básico de una estructura de lattice no lineal.

El objetivo final será comparar los algoritmos LMS y RLS para un sistema de Volterra.

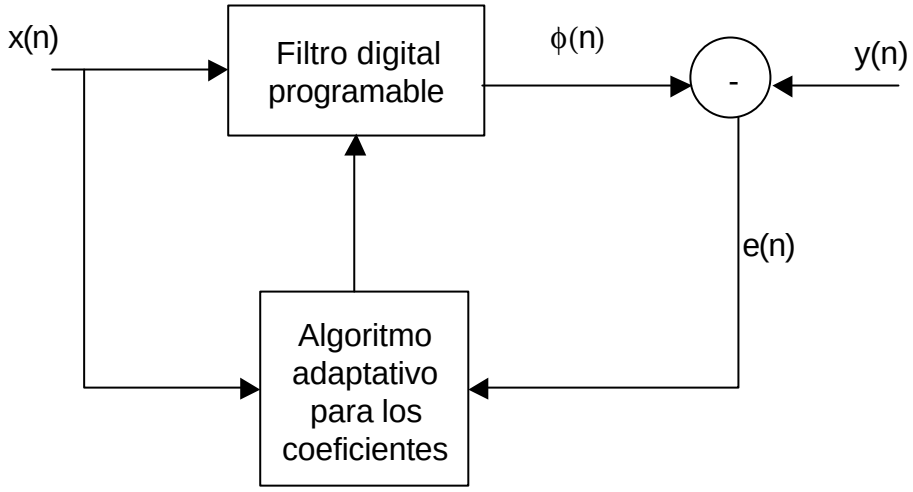
Para empezar, vamos a ver un poco sobre filtros adaptativos en general y los algoritmos que se usan.

Un filtrado es adaptativo si sus parámetros, los coeficientes, varían de acuerdo con un criterio específico conforme nueva información va llegando. Un filtro adaptativo se dice lineal si la estimación de la cantidad de interés se obtiene adaptativamente (a la salida del filtro) como una combinación lineal del conjunto de observaciones aplicadas a la entrada del filtro. En otro caso, el filtro adaptativo es no lineal.

Las técnicas de filtrado adaptativo se han usado con éxito durante los últimos años. Conforme se ha ganado experiencia en las aplicaciones y el procesado de señal madura, estas técnicas son más refinadas y sofisticadas. Pero para hacer un mejor uso del potencial de estas técnicas, los

usuarios deben aprender con detalle como funcionan realmente, y no aplicar simplemente algoritmos. Además, el número de algoritmos disponibles ha crecido mucho: no es raro encontrar una docena de ellos. Encontrar el mejor es un problema crucial en la ingeniería.

El esquema de un filtro adaptativo sería: [4]




Es decir, la salida de un filtro digital programable con coeficientes variables se resta de una señal de referencia, y(n), para producir una secuencia de error, e(n), que se usa junto con elementos de la secuencia de entrada, x(n), para adaptar los coeficientes del filtro, siguiendo un criterio de optimización.

Los filtros adaptativos se clasifican de acuerdo a las opciones que se tomen en las siguientes áreas: [4]

- ***Criterio de optimización:*** en general se toma de la familia LS (Least Squares) para poder trabajar con operaciones lineales. Sin embargo, en algunos casos, donde la simplicidad de la implementación y la robustez son importantes, el criterio del valor absoluto mínimo (LAV) puede ser atractivo; además, no está restringido a una optimización de fase mínima.

- ***Algoritmo para adaptar los coeficientes:*** dependen totalmente del criterio de optimización aunque a menudo es el algoritmo el que decide la elección del criterio de

optimización y no al revés. En términos generales, el criterio del menor valor cuadrático (LMS-Least Mean Squares) se asocia con el algoritmo del gradiente, el criterio LAV corresponde a algoritmos de Signo, y el criterio LS exacto se asocia con una familia de algoritmos recursivos, siendo el más eficiente de éstos últimos el algoritmo rápido de mínimos cuadrados (Fast RLS).

- ***Estructura del filtro programable:*** El filtro puede ser de tipo FIR o IIR y, en principio, puede tener cualquier estructura: forma directa, cascada, lattice, escalera o filtro de onda. Al igual que con los filtros de coeficientes fijos, la complejidad computacional varía con la estructura. Lo característico de los filtros adaptativos es que la estructura influye en la complejidad del algoritmo. La estructura en forma directa FIR, o transversal, es la más simple para estudiar e implementar y, por tanto, es la más popular.

- ***Tipo de las señales procesadas (mono- o multidimensional):*** las señales multidimensionales pueden usar los mismos algoritmos y estructuras que las monodimensionales. Sin embargo, la complejidad computacional y las limitaciones del hardware generalmente reducen las opciones a las técnicas más simples.

Fundamentalmente, existen dos técnicas para el desarrollo de los algoritmos: [3]

- ***Técnica del gradiente estocástico:***

Los filtros adaptativos basados en las técnicas del gradiente forman una clase muy apreciada en la ingeniería por su simplicidad, flexibilidad y robustez. Además, son fáciles de diseñar y su funcionamiento está bien caracterizado. Es la más usada en numerosos campos, particularmente en comunicaciones y control.

Se suele utilizar con un filtro transversal y hay que minimizar el error cuadrático medio. (La función que hay que optimizar se llama función de coste). Se utiliza una estimación del vector gradiente, que se obtiene con la matriz de correlación de las entradas y el vector de correlación

cruzada entre las entradas y la respuesta deseada (ya lo veremos más adelante).

El algoritmo que resulta es conocido como Least Mean Square (LMS). Es simple y actúa satisfactoriamente bajo ciertas condiciones. Sus mayores limitaciones son:

- una relativamente lenta tasa de convergencia, que se define como el número de iteraciones necesarias para que el algoritmo, en respuesta a entradas estacionarias, converja lo suficientemente cerca a la solución óptima en el sentido cuadrático medio. Una tasa alta permite al algoritmo adaptarse rápidamente a ambientes estacionarios con estadísticos desconocidos; y

- una sensibilidad a las variaciones en el número de condición de la matriz de correlación de las entradas.
El número de condición se define como el cociente entre los autovalores más grande y más pequeño.

También se usa con una estructura lattice.

➢ ***Estimación de mínimos cuadrados:***

Se minimiza una función de coste que se define como una suma ponderada del error al cuadrado. El método se formula con una estimación por bloques o una estimación recursiva. Normalmente, el estimador recursivo requiere menos almacenamiento que uno por bloques, y por eso en la práctica se usa mucho más el primero.

Se clasifica la familia RLS (Recursive Least Squares) en tres categorías:

- Algoritmos RLS estándar: Se usa un filtro transversal y el lema de inversión de matrices. Sus limitaciones: falta de robustez numérica y excesiva complejidad computacional. Debido a estas dos limitaciones, aparecen las otras dos categorías.

- Algoritmos RLS raíz cuadrada: Se basan en una descomposición QR de la matriz de datos.

- Algoritmos RLS rápidos: La complejidad computacional de los anteriores puede llegar a ser inaceptable desde el punto de vista de la implementación hardware. Ésta se consigue reducir haciendo uso de la redundancia en la estructura Toeplitz de la matriz de entrada

a través del uso de la predicción lineal de mínimos cuadrados tanto hacia delante como hacia atrás. El algoritmo resultante se llama RLS Rápido y hay dos tipos dependiendo de la estructura que se use para el filtro:

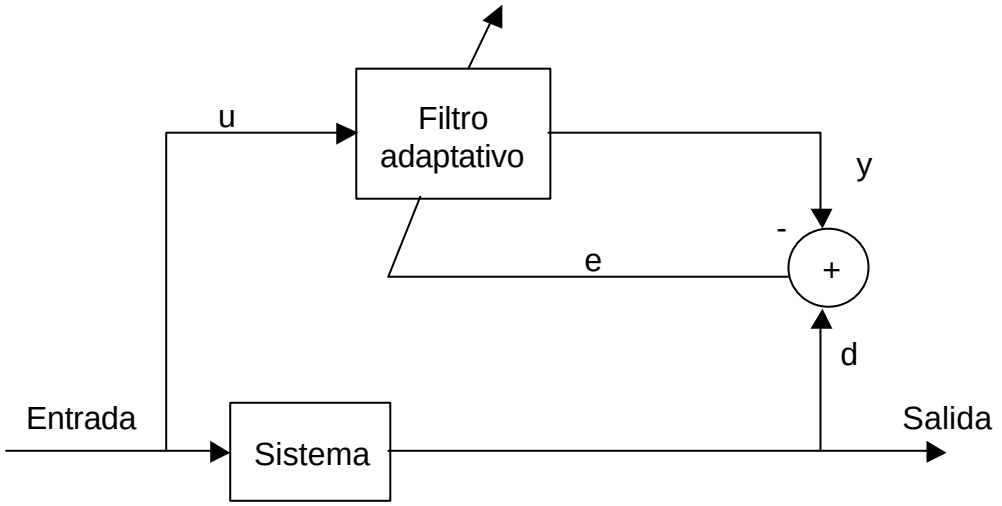
- Filtros adaptativos de orden recursivo: se basan en una estructura lattice para hacer las predicciones hacia delante y hacia atrás.

- Filtros transversales rápidos: las predicciones hacia delante y hacia atrás se hacen usando filtros transversales por separado.

## 1.1.- Aplicaciones

La habilidad de un filtro adaptativo para operar satisfactoriamente en un ambiente desconocido y seguir las variaciones temporales de los estadísticos de entrada hacen del filtro adaptativo una herramienta poderosa para aplicaciones en el procesado de señal y de control. De hecho, se han utilizado con éxito en campos diversos como las comunicaciones radar, sonar, sismología e ingeniería biomédica. Aunque estas aplicaciones son bastante diferentes, tienen un hecho en común: la entrada y la respuesta deseada se usan para obtener un error estimado, el cual a su vez se usa para controlar los valores de un conjunto de coeficientes ajustables del filtro. Estos coeficientes dependerán de la estructura de filtro empleada. La diferencia esencial entre las distintas aplicaciones es la manera en la que se obtiene la respuesta deseada. Vamos a ver cuatro clases básicas de aplicaciones: [3]

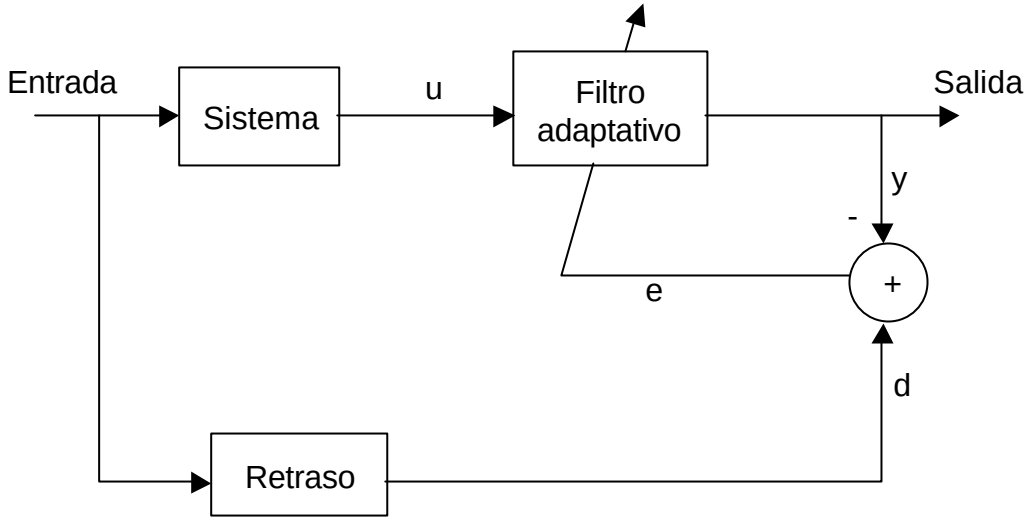
- ***Identificación:***




Se usa el filtro adaptativo para obtener un modelo lineal que represente de la mejor manera (en algún sentido) a un sistema desconocido (sistema). La salida del sistema suministra la respuesta deseada para el filtro adaptativo.

Como aplicaciones concretas estarían: identificación de sistema y modelado estratificado de la Tierra.

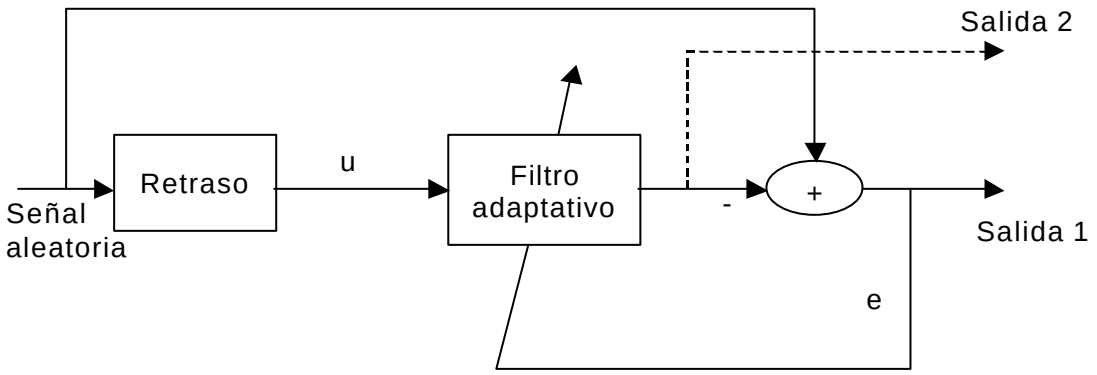
- ***Modelado inverso:***




La función del filtro adaptativo es proveer un modelo inverso que represente de la mejor manera (en algún sentido) a un sistema desconocido ruidoso. Idealmente, en el caso de un sistema lineal el modelo inverso tiene una función de transferencia igual a la inversa de la del sistema. Una versión retrasada de la entrada del sistema constituye la respuesta deseada para el filtro adaptativo.

Posibles aplicaciones concretas: deconvolución predictiva, ecualización adaptativa y ecualización ciega.

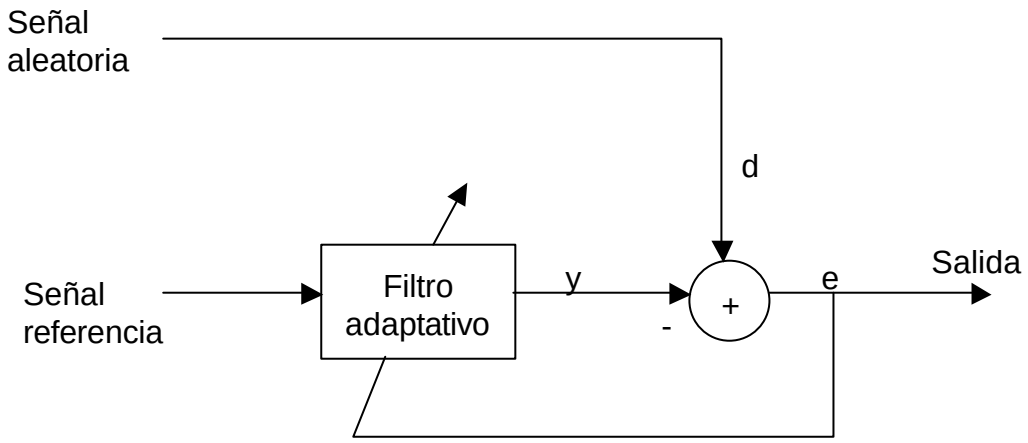
- ***Predicción:***



La función es suministrar la mejor predicción (en algún sentido) del valor actual de la señal aleatoria. El valor actual de la señal sirve para la respuesta deseada para el filtro adaptativo. Dependiendo de la aplicación en cuestión, la salida del filtro adaptativo o el error estimado pueden servir como salidas del sistema. En el primer caso, el sistema actúa como un predictor; en el último caso, opera como un filtro de error de predicción.

Algunas de las aplicaciones concretas serían: código predictivo lineal, modulación de pulso diferencial adaptativo, análisis del espectro autorregresivo y detección de señales.

- ***Cancelación de interferencia:***




Se usa para cancelar una interferencia desconocida contenida en una señal primaria. La cancelación debe ser óptima en algún sentido. La señal primaria sirve como respuesta deseada para el filtro adaptativo. Se emplea una señal de referencia como entrada al filtro adaptativo.

Aplicaciones concretas: cancelación de ruido adaptativo, cancelación de eco y conformación de haz adaptativa.

Para cerrar esta introducción, se va a explicar lo que se trata en cada capítulo del proyecto.

En el siguiente capítulo se estudian los fundamentos teóricos de este tipo de estructuras. Se comienza con una introducción al filtrado adaptativo no lineal. Después se analizan por separado los dos modelos polinomiales que nos sirven para modelar la no linealidad. Primero, los filtros adaptativos usando expansiones en serie truncada de Volterra; y, a continuación, los filtros adaptativos usando ecuaciones en diferencias no lineales y recursivas. Finalmente, se comenta un poco sobre un tipo concreto de estructura: la lattice. Es un tipo de estructura a la que se le pueden aplicar los algoritmos anteriores y que se basa en la ortogonalización de las señales de entrada.

En el tercer capítulo se analizan dos algoritmos (LMS y RLS) para el tratamiento de los filtros adaptativos de Volterra. Éste es el único modelo que se analiza con profundidad. Tras estudiarlos por separado, se comparan las ventajas y desventajas de cada uno de ellos.

En el siguiente capítulo se comentan algunas de las aplicaciones de los filtros de Volterra de segundo orden (que son los que se analizan). Nos centramos un poco más en el modelado de sistemas no lineales puesto que es la aplicación que luego se utiliza en las simulaciones.

En el quinto capítulo se analizan dos ejemplos. Se simulan multitud de casos utilizando los dos algoritmos que se han visto antes.

En el sexto capítulo se comentan algunas aplicaciones futuras y por donde van las investigaciones en el filtrado adaptativo no lineal.

Finalmente se muestran los programas de Matlab que se han utilizado en las simulaciones y se comenta un poco sobre el futuro de los filtros adaptativos polinomiales.

Por último, se hace referencia a la bibliografía utilizada.

# CAPITULO 2.

# FUNDAMENTOS TEÓRICOS

## 2.1.- Introducción filtrado adaptativo no lineal

Como ya sabemos, la gran ventaja de los filtros lineales es su simplicidad. Sin embargo, hay muchas situaciones en las que el comportamiento de éstos es inaceptable. Un ejemplo simple de un tipo de no linealidad es la saturación. Intentar utilizar modelos lineales para estos sistemas suele dar resultados erróneos.

Los problemas con sistemas no lineales tienen soluciones difíciles desde un punto de vista analítico y/o computacional y no hay tantas técnicas disponibles como para los sistemas lineales. Estas dificultades son mucho mayores en el caso de sistemas no lineales adaptativos.

Al contrario que en los sistemas lineales, para los no lineales no existe una respuesta impulsiva para describir sistemas no lineales arbitrarios. Debido a esto, las investigaciones sobre estos sistemas se ven restringidas a ciertos modelos que son menos generales. Algunos filtros no lineales desarrollados usando estos modelos son: [1]

- *Filtros de orden estadístico:* son interesantes debido a su robustez y a su simplicidad computacional. Se basan en el orden de los estadísticos de la señal de entrada al filtro. El más usado es el *median filter*. Tienen un amplio margen en el que mantienen una serie de propiedades y son muy útiles para la eliminación de ruido impulsional aditivo (en general, ruido perteneciente a distribuciones con una larga cola) de las señales de entrada. Principalmente tiene aplicaciones en el procesado de imágenes.

➢ ***Filtros homomórficos:*** son los más antiguos y tienen aplicaciones en el realce de imágenes, en el procesado de señales sísmicas y en la supresión de ruido multiplicativo de las señales de entrada. En aplicaciones de codificación de imágenes se han usado mucho modelos de sistemas de visión humanos basados en este tipo de filtros.

➢ ***Filtros morfológicos:*** utilizan las formas geométricas de las señales de entrada y se emplean en aplicaciones que incluyen reconocimiento de formas, detección de bordes y otras.

➢ ***Filtros basados en Volterra y otras descripciones polinomiales para las no linealidades:*** son más generales que los anteriores. Nos centraremos en éstos y estudiaremos dos casos: filtros adaptativos empleando una representación en serie truncada de Volterra para los sistemas no lineales; y otros usando ecuaciones en diferencias recursivas y no lineales para relacionar las señales de entrada y salida del sistema. Es posible ver la representación en serie truncada de Volterra como un caso particular de la representación de sistemas recursivos no lineales aunque nosotros los veremos por separado. El modelado de Volterra es muy popular en filtrado no lineal adaptativo y ha desarrollado una identidad propia en los últimos años. Sin embargo, el uso de modelos de realimentación está en sus inicios y, aunque estos sistemas son muy interesantes desde un punto de vista de implementación, existen numerosos problemas para los cuales no se han encontrado todavía soluciones. Trataremos con más profundidad algoritmos basados en una expansión en serie truncada de Volterra.

Antes hemos visto algunas aplicaciones de los filtros adaptativos en general. Algunas aplicaciones concretas usando estructuras no lineales serían:

- Igualadores de canal no lineales: los canales de comunicación de alta velocidad los suelen necesitar. En transmisiones telefónicas, las no linealidades aparecen principalmente por inexactitudes en la compresión de la señal. En los enlaces digitales por satélite, los amplificadores de éste se llevan cerca del punto de saturación y ahí hay características grandemente no lineales.

- Cancelación de eco.

- Análisis de sistemas de transmisión de datos.

- Cancelación de ruido adaptativa.

- Detección de funciones no lineales de procesos gausianos.

- Modelado de fenómenos biológicos.

- Procesado de señales mioeléctricas.

- Caracterización de dispositivos semiconductores.

- Procesado de imágenes.

## 2.2.- Filtros adaptativos usando expansiones en serie truncada de Volterra

### 2.2.1.- Series de Volterra

Un sistema se puede definir en un marco matemático como una regla mediante la cual una entrada, x, se transforma en una salida, y, mediante un operador, S, y se representa por: [2]

$$y = S[x]$$

Normalmente, la entrada y la salida son funciones de variables independientes tales como el tiempo o la posición; si son sólo funciones del tiempo, t, la ecuación anterior es:

$$y(t) = S[x(t)]$$

Se sabe que un sistema invariante en el tiempo (TI) se caracteriza por la propiedad:

$$y(t+\tau) = S[x(t+\tau)]$$

y que la salida de un sistema lineal e invariante en el tiempo (LTI) viene dada por la integral de convolución:

$$y(t) = \int_{-\infty}^{+\infty} h(\tau)\,x(t-\tau)\,d\tau$$

donde h(t) es la respuesta impulsiva unitaria del sistema que caracteriza completamente al sistema LTI.

Un sistema se dice causal si la salida en cualquier instante no depende de valores futuros de la entrada. Un sistema LTI es causal si y sólo si

$$h(\tau) = 0 \qquad \tau < 0$$

La contribución a la salida de una entrada T segundos antes del valor actual de la salida se determina por h(T), es decir, la respuesta impulsiva unitaria representa la memoria del sistema LTI.

Si el sistema LTI no tiene memoria, entonces

$$h(t) = c\,u_0(t)$$

donde $u_0(t)$ es el impulso unitario, y c es una constante; la respuesta del sistema es evidente

$$y(t) = c\,x(t)$$

Si un sistema sin memoria es no lineal, su salida se puede expresar, con ciertas restricciones, con una serie de Taylor:

$$y(t) = \sum_{k=0}^{\infty} c_k\, x^k(t)$$

La extensión de esta expresión a sistemas no lineales con memoria es la serie de Volterra

$$y(t) = h_0 + \int_{-\infty}^{+\infty} h_1(\boldsymbol{t})\,x(t-\boldsymbol{t})\,d\boldsymbol{t}_1 + \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} h_2(\boldsymbol{t}_1,\boldsymbol{t}_2)\,x(t-\boldsymbol{t}_1)\,x(t-\boldsymbol{t}_2)\,d\boldsymbol{t}_1\,d\,\boldsymbol{t}_2 +$$

$$+ \ldots + \int_{-\infty}^{+\infty}\ldots\int_{-\infty}^{+\infty} h_k(\boldsymbol{t}_1,\ldots,\boldsymbol{t}_k)\,x(t-\boldsymbol{t}_1)\ldots x(t-\boldsymbol{t}_k)\,d\,\boldsymbol{t}_1\ldots d\,\boldsymbol{t}_k + \ldots \qquad (2.1)$$

Las funciones $h_k(\tau_1,\ldots,\tau_k)$, llamadas núcleos de Volterra, caracterizan completamente al sistema no lineal. Por analogía con los sistemas lineales, un sistema no lineal representado por una serie de Volterra es causal si y sólo si

$$h_k(\boldsymbol{t}_1,\ldots,\boldsymbol{t}_k) = 0 \qquad \forall\ \boldsymbol{t}_j<0,\quad j=1,\ldots,k$$

Sin pérdida de generalidad, podemos asumir que los núcleos de Volterra son simétricos (es decir, $h_k(\tau_1,\ldots,\tau_k)$ se mantiene constante ante las k! posibles permutaciones de los índices $\tau_i$).

Una relación más compacta se puede obtener definiendo los términos de las integrales como el operador de Volterra de orden k, $\overline{H}_k$. La serie de Volterra se expresa entonces de la forma:

$$y(t) = h_0 + \sum_{k=1}^{\infty} \overline{H}_k[x(t)] \qquad (2.2)$$

Conviene observar que la serie de Volterra es una serie de potencia respecto a un factor multiplicador aplicado a la entrada x(t): la salida, y(t), será en este caso

$$y(t) = h_0 + \sum_{k=1}^{\infty} \overline{H}_k[c\,x(t)] = h_0 + \sum_{k=1}^{\infty} c^k\, \overline{H}_k[x(t)]$$

Además, es una serie con memoria puesto que las integrales tienen la forma de convoluciones multidimensionales. Como consecuencia de su carácter de serie de potencia, hay algunas limitaciones asociadas con las aplicaciones de la serie de Volterra a problemas no lineales:

- Dificultades en la convergencia cuando se modelan sistemas no lineales con saturación. El mismo problema existe con la representación en serie de Taylor de una función fuertemente no lineal. Esta similitud era de esperar porque la serie de Volterra es una serie de Taylor con memoria.

- En contraposición con el caso lineal, la respuesta impulsiva no caracteriza completamente un sistema no lineal.

Dos aspectos importantes de la expansión de Volterra se obtienen de su definición formal [(2.1)-(2.2)]:

- La salida del filtro es lineal respecto a los elementos de los núcleos de Volterra.

- Las funcionales $\overline{H}_k[x(t)]$ se pueden interpretar como convoluciones multidimensionales.

Las implicaciones de estas dos propiedades se pueden explotar también en el caso discreto. De hecho, se puede introducir una clase de sistemas discretos no lineales TI con memoria (también llamados filtros polinomiales o de Volterra), descritos por una serie discreta de Volterra derivada de (2.2) en la forma:

$$y(n) = h_0 + \sum_{p=1}^{\infty} \overline{h}_p[x(n)] \qquad (2.3)$$

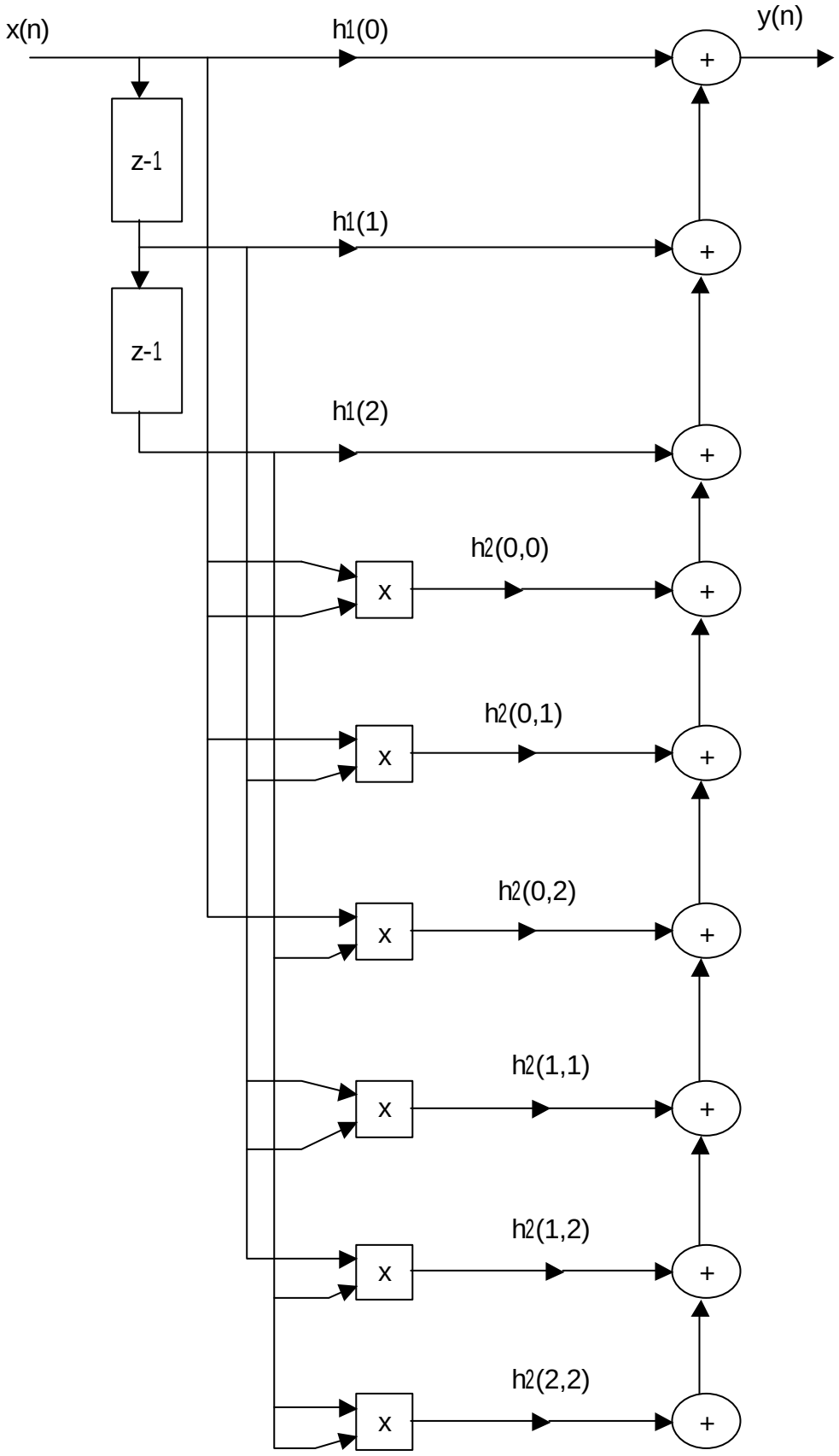
donde y(n) y x(n) son las versiones muestreadas de y(t) y x(t), y

$$\overline{h}_p[x(n)] = \sum_{m_1=0}^{\infty} \dots \sum_{m_p=0}^{\infty} h_p(m_1,\dots,m_p)\,x(n-m_1)\dots x(n-m_p) \qquad (2.4)$$

donde $h_0$ es un término de offset, $h_1(m_1)$ es la respuesta impulsiva de un filtro digital IIR, y $h_p(m_1,\ldots,m_p)$ se considera como la respuesta impulsiva generalizada de orden p que caracteriza el comportamiento no lineal del filtro. El término $h_0$ a menudo se estima fuera de la estructura básica. Por eso, sin pérdida de generalidad, se asume que $h_0=0$.(Más adelante se verá un ejemplo en el que $h_0$ es distinto de cero).

Un filtro cuadrático (de orden dos) completo se describe entonces con los tres primeros términos de la serie de Volterra. El límite superior de los sumatorios de (2.4) indica memoria infinita del sistema. Sin embargo, a menudo la memoria requerida es finita. En este caso, más útil, se limitan todos los sumatorios en (2.4) a un valor finito N-1, es decir, definiendo las variables independientes discretas $m_1,\ldots,m_p$ en un dominio finito. En este caso $h_1(m_1)$ representa un filtro FIR.

Como ejemplo, una estructura básica para un sistema de orden 2 (p=2) y con N-1=2 elementos de retraso sería como se muestra a continuación:

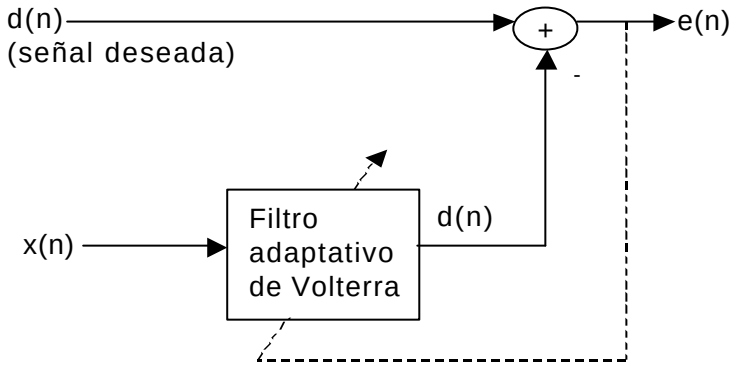

x(n)
y(n)
h1(0)
h1(1)
h1(2)
z-1
z-1
h2(0,0)
h2(0,1)
h2(0,2)
h2(1,1)
h2(1,2)
h2(2,2)
x
+

Este sistema es lineal en la señal de entrada para cada coeficiente y esto simplifica enormemente los problemas de diseño que incluyen representaciones en serie de Volterra.

Sin embargo, vemos que el número de coeficientes en esta expansión polinomial es proporcional a $N^p$ (a partir de ahora lo expresaremos como $Pr(N^p)$). Luego, una gran desventaja de estos modelos es que puede ser muy complicado la implementación de estos filtros incluso para valores moderados de N y p. Debido a esto, la mayoría de las aplicaciones prácticas de sistemas empleando expansiones en serie de Volterra se basan en modelos de bajo orden.

### 2.2.2.- Filtros adaptativos

Vamos a utilizar la expansión en serie de Volterra que se ha visto antes para diseñar un filtro adaptativo no lineal. La linealidad de la salida de los filtros de Volterra respecto a los núcleos ($h_i(m_1,m_2,\ldots,m_p)$) permite la extensión de la teoría del filtrado lineal óptimo al filtrado de Volterra no lineal óptimo, de acuerdo al criterio de minimizar el error cuadrático medio(MSE). Un diagrama de bloques de un filtro adaptativo de Volterra sería:




Sean x(n) y d(n) dos procesos aleatorios estacionarios con parámetro discreto n. Como se ve en el dibujo, x(n) es la entrada al filtro no lineal y d(n) es la señal de referencia. El objetivo, como siempre que se trata de filtrado adaptativo, es encontrar los coeficientes del filtro de tal manera que se minimice el MSE entre d(n) y la salida del filtro $\hat{d}(n)$.

El MSE tiene un único mínimo global y se puede encontrar mediante el cálculo de variaciones, o, alternativamente, aplicando una extensión del principio de proyección ortogonal. De hecho, el error residual de un filtro de Volterra MSE mínimo es ortogonal no sólo a la entrada sino también a todos los posibles productos de las entradas. Si el filtro no lineal es de orden dos con un término lineal, la salida se puede poner:

$$\hat{d}(n) = \sum_{m_1=0}^{N-1} h_1(m_1)x(n-m_1) + \sum_{m_1=0}^{N-1}\sum_{m_2=0}^{N-1} h_2(m_1,m_2)x(n-m_1)x(n-m_2) \qquad (2.5)$$

El principio de proyección ortogonal, [2], se expresa:

$$\mathrm{E}\left[\left\{d(n)-\hat{d}(n)\right\}x(n-m_1)\right] = 0 \qquad (2.6a)$$

$$\mathrm{E}\left[\left\{d(n)-\hat{d}(n)\right\}x(n-m_1)x(n-m_2)\right] = 0 \qquad (2.6b)$$

Sustituyendo la expresión (2.5) en (2.6), se obtiene un sistema lineal con N + N(N+1)/2 ecuaciones y con el mismo número de incógnitas y teniendo en cuenta la simetría del núcleo de Volterra:

$$\sum_{m_1=0}^{N-1} h_1(m_1)\mathrm{E}[x(n-m_1)x(n-l)] + \sum_{m_1=0}^{N-1}\sum_{m_2=0}^{N-1} h_2(m_1,m_2)\mathrm{E}[x(n-m_1)x(n-m_2)x(n-l)] =$$

$$= \mathrm{E}[d(n)x(n-l)], \qquad l = 0,1,\ldots,N-1 \qquad (2.7)$$

$$\sum_{m_1=0}^{N-1} h_1(m_1)\mathrm{E}[x(n-m_1)x(n-l)x(n-t)] + \sum_{m_1=0}^{N-1}\sum_{m_2=0}^{N-1} h_2(m_1,m_2)\mathrm{E}[x(n-m_1)x(n-m_2)x(n-l)\cdot$$

$$\cdot x(n-t)\,] = \mathrm{E}[d(n)x(n-l)x(n-t)], \qquad l = 0,1,\ldots,N-1; \quad t = l,\ldots,N-1$$

La solución de este sistema da los coeficientes $h_1(m_1)$ y $h_2(m_1,m_2)$ de un filtro cuadrático óptimo de acuerdo con el criterio MSE.

Si además se añade que la señal de entrada y la de referencia son de media cero, se debe considerar también la condición

$$\mathrm{E}\left[d(n)-\hat{d}(n)\right] = 0 \qquad (2.8)$$

por lo que se añade un término constante (un offset) $h_0$ a la representación del filtro cuadrático en (2.5), el cual se obtiene usando (2.8) de la forma

$$h_0 = -\sum_{m_1=0}^{N-1}\sum_{m_2=0}^{N-1} h_2(m_1,m_2)r_x(m_1,m_2)$$

donde $r_x(m_1) = \mathrm{E}[x(n)x(n-m_1)]$ es la función de autocorrelación de x(n). Es posible incluir el término de offset en la parte cuadrática, y entonces aplicar el principio de proyección ortogonal como en (2.7).

En general, la solución directa del sistema (2.7) requiere obtener una matriz inversa cuyos elementos están en función de los momentos de segundo, tercer y cuarto orden de x(n). La inversión de la matriz requiere un número de operaciones del orden de $N^6$ (Pr $N^6$), lo que supone una elevada carga computacional para valores altos de N. Además, el sistema (2.7) muestra que en general existe acoplamiento entre los coeficientes lineales y cuadráticos del filtro.

Si la señal de entrada es gausiana, este acoplamiento no existe puesto que los momentos de tercer orden de esta entrada son cero. Así, la matriz del sistema (2.7) se puede separar en dos submatrices, permitiendo soluciones distintas para los coeficientes lineales y cuadráticos.

Como, además, los momentos de cuarto orden de la entrada se pueden expresar en función de tres momentos de segundo orden, es posible demostrar que los coeficientes lineales se pueden obtener usando la conocida ecuación de Wiener-Hopf:

$$H_1 = R_x^{-1} R_{yx} \qquad (2.9)$$

donde $H_1$ es el vector de coeficientes $h_1(m_1)$ del filtro lineal, $R_x$ es la matriz de autocorrelación (NxN) de la entrada x(n), y $R_{yx}$ es el vector de correlación cruzada (Nx1) entre d(n) y x(n). Los coeficientes cuadráticos $h_2(m_1,m_2)$, agrupados en una matriz $H_2$, se obtienen entonces de la ecuación

$$H_2 = (1/2)R_x^{-1} T_{yx} R_x^{-1} \qquad (2.10)$$

donde $T_{yx}$ está formado con términos del tipo

$$t_{yx}(m_1,m_2) = \mathrm{E}[d(n)x(n-m_1)x(n-m_2)]$$

La componente lineal del filtro es exactamente la misma que la que se obtiene en el caso lineal puro, mientras que para la componente cuadrática sólo se requiere la inversa de la matriz $R_x$. Como $R_x$ es Toeplitz, sólo son necesarias Pr($N^2$) operaciones. Por tanto, se consigue una importante reducción en la complejidad computacional. Se puede observar también que, si x(n) y d(n) son ambas gausianas, los elementos de $T_{yx}$

son todos ceros. En este caso, el filtro lineal óptimo es el mejor filtro posible.

Por analogía con el caso lineal, un filtro adaptativo no lineal se puede usar para aproximar los coeficientes de un filtro óptimo.

Una importante consecuencia de la linealidad de los filtros de Volterra respecto a sus coeficientes es que todos los algoritmos de adaptación válidos en el caso lineal pueden, en principio, extenderse a los filtros de Volterra, y en particular a los cuadráticos (orden dos).

Como acabamos de ver, los algoritmos que se utilizan son análogos a los que se usan en el filtrado adaptativo lineal. Entre los más utilizados están el algoritmo LMS (Least Mean Square) y sus variaciones y los algoritmos RLS (Recursive Least Square). Éstos últimos son al menos un orden de magnitud más complejo que los de tipo LMS.

En el siguiente capítulo, se estudian estos dos algoritmos por separado y luego se comparan.

## 2.3.- Filtros adaptativos usando ecuaciones en diferencias no lineales y recursivas

### 2.3.1.- Introducción

El principal problema que hay con la representación en series de Volterra de sistemas no lineales es el gran número de coeficientes que hacen falta para caracterizar estos sistemas. Por eso, se buscan representaciones alternativas que usen un menor número de coeficientes. Uno de estos modelos es el que emplea una ecuación en diferencias no lineal y recursiva para definir la relación entrada/salida. Así, se tiene: [1]

$$y[n] = \sum_{i=1}^{M} P_i\left(y[n-1], y[n-2], \ldots, y[n-N+1], x[n], x[n-1], \ldots, x[n-N+1]\right)$$

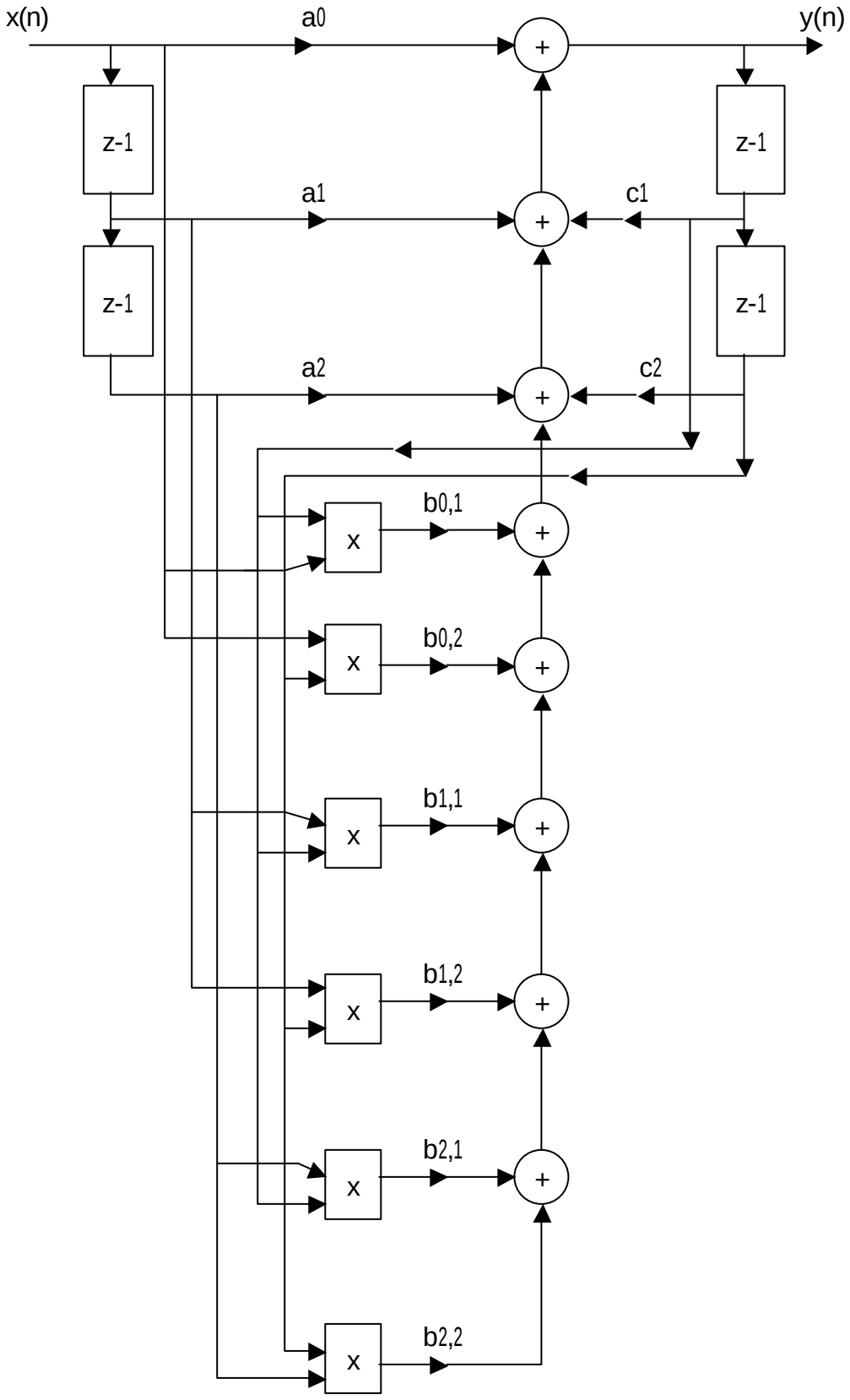
donde $P_i(\bullet,\bullet,\ldots,\bullet)$ es un polinomio de orden i-ésimo en las cantidades de dentro del paréntesis.

Seguramente, el sistema más simple de esta clase sea el sistema bilineal, cuya relación entrada/salida viene dada por:

$$y[n] = \sum_{i=1}^{N-1} c_i\, y[n-i] + \sum_{i=0}^{N-1}\sum_{j=1}^{N-1} b_{i,j}\, y[n-j]x[n-i] + \sum_{i=0}^{N-1} a_i\, x[n-i]$$

A pesar de su simplicidad, es un modelo no lineal importante porque, bajo unas condiciones moderadas, se puede demostrar que una gran cantidad de sistemas no lineales, incluyendo los de Volterra, se pueden representar usando estos modelos de sistemas bilineales con un número finito de coeficientes. Además, la mayoría de las ideas de los sistemas bilineales se pueden extender fácilmente para modelos más generales.

El diagrama de bloques de un sistema bilineal para N=3 sería:


x(n)
a0
y(n)
z-1
z-1
a1
c1
z-1
z-1
a2
c2
b0,1
b0,2
b1,1
b1,2
b2,1
b2,2
x
+

La gran ventaja de estos sistemas, como ya se ha dicho antes, es que es posible representar un gran número de sistemas no lineales con menos coeficientes que los sistemas de Volterra. Sin embargo, y como era de esperar, existen inconvenientes. Los más importantes son dos: debido a la realimentación, hay que monitorizar continuamente la adaptación para que no haya problemas con la estabilidad; y, además, cualquier ruido en las señales de entrada aparecerá multiplicado en el modelo y afectará al comportamiento de estos sistemas adaptativos.

Al igual que en el caso de filtros lineales adaptativos IIR, hay dos formas diferentes de resolver estos problemas: ecuación de error y error de salida.

### 2.3.2.- Algoritmos de ecuación de error

Se desarrollan directamente. La superficie de la estimación del error cuadrático medio (función de coste) tiene un único mínimo aunque éste podría no ser la solución correcta al problema si hay ruido en la señal deseada. Además, no hay garantías de que las soluciones al filtro adaptativo sean estables todo el tiempo. La idea es usar muestras de la señal de entrada x[n] y de la respuesta deseada d[n] para obtener la estimación del filtro adaptativo como:

$$\hat{d}[n] = \sum_{i=1}^{N-1} c_i[n]d[n-i] + \sum_{i=0}^{N-1}\sum_{j=1}^{N-1} b_{i,j}[n]d[n-j]x[n-i] + \sum_{i=0}^{N-1} a_i[n]x[n-i]$$

donde $c_i[n]$, $b_{i,j}[n]$ y $a_i[n]$ son los coeficientes del filtro adaptativo en la n-ésima iteración. Estos coeficientes se pueden adaptar usando el algoritmo del gradiente, una solución RLS o cualquier otra técnica apropiada. Lo que se pretende minimizar es:

$$E\left\{\left[d[n] - \hat{d}[n]\right]^2\right\}$$

Las ecuaciones de adaptación del gradiente son:

$$c_i[n+1] = c_i[n] + \mu_c\, d[n-i]e[n]$$

$$a_i[n+1] = a_i[n] + \mu_a\, x[n-i]e[n]$$

$$b_{i,j}[n+1] = b_{i,j}[n] + \mu_b\, d[n-j]x[n-i]e[n]$$

donde $e[n] = d[n] - \hat{d}[n]$ es la estimación del error y $\mu_a$, $\mu_b$ y $\mu_c$ son constantes que controlan la razón de convergencia del filtro adaptativo.

Hay que hacer notar que si d[n] contiene ruido, los estadísticos de la señal de entrada al filtro adaptativo serán diferentes a los de la señal deseada ideal y esto provocará una peor estimación. Esto complica considerablemente el problema cuando se compara con el filtrado lineal IIR. Debido a la presencia de términos con productos x[n-i]d[n-j] en la obtención de la salida de los sistemas bilineales adaptativos, habrá componentes de ruido multiplicado en la salida. A pesar de esto, este método es interesante por su simplicidad y es muy útil en ambientes de bajo ruido.

### 2.3.3.- Algoritmos de error de salida

Realimentan la salida del sistema adaptativo para estimar la señal deseada. Seguramente, de entre los métodos disponibles en la literatura, el más simple sea el método subóptimo de mínimos cuadrados (también se hace referencia a él como algoritmo extendido de mínimos cuadrados). Es conveniente utilizar notación vectorial. Así, sean:

$$H[n] = \left(c_1[n], c_2[n], \dots, c_{N-1}[n], b_{0,1}[n], \dots, b_{N-1,N-1}[n], a_0[n], \dots, a_{N-1}[n]\right)^T$$

$$X[n] = \left(\hat{d}_{n-1}[n-1], \hat{d}_{n-2}[n-2], \dots, \hat{d}_{n-N+1}[n-N+1], x[n]\hat{d}_{n-1}[n-1], \dots,\right.$$

$$\left. x[n-N+1]\hat{d}_{n-N+1}[n-N+1], x[n], \dots, x[n-N+1]\right)^T$$

el vector de coeficientes del filtro adaptativo y el vector de entrada al filtro, respectivamente. Aquí, $\hat{d}_k[l]$ denota la estimación de la señal deseada en el tiempo l haciendo uso de los coeficientes en el tiempo k.

Luego, los coeficientes del filtro en el tiempo n se obtienen como la solución que minimiza:

$$J[n] = \sum_{k=0}^{n} l^{n-k}\left(d[k] - H^T[n]X[k]\right)$$

La salida del filtro será:

$$\hat{d}_n[n] = H^T[n]X[k]$$

donde H[n] se estima como: $H[n] = C^{-1}[n]P[n]$

Los vectores C[n] y P[n] son como los que se usan para los problemas de filtrado de Volterra RLS, es decir:

$$C[n] = \sum_{k=0}^{n} l^{n-k} X[k] X^T[k]$$

$$P[n] = \sum_{k=0}^{n} l^{n-k} d[k] X[k]$$

Se puede hacer un poco menos complejo haciendo uso del lema de inversión de matrices para invertir C[n]. Este lema se explica luego cuando se analiza el algoritmo RLS.

Aunque la formulación es parecida a la que se usa en el filtrado RLS, estas ecuaciones no representan una solución exacta de mínimos cuadrados (como ocurre en RLS) tal y como vamos a ver ahora. El problema exacto requiere que la función de coste J[n] se defina usando un error de estimación:

$$e[n] = d[k] - H^T[n] X[k]$$

obtenido en el tiempo k haciendo uso de la solución H[n] actual. De esta forma, el problema se plantea como si estuviéramos buscando una solución completamente diferente en cada iteración (aunque es posible adaptar los coeficientes basándonos en la solución previa).

En el problema descrito aquí, las señales

$$\hat{d}_{k-1}[k-1], \hat{d}_{k-2}[k-2], \ldots, \hat{d}_{k-N+1}[k-N+1]$$

que aparecen en el vector de entrada se obtienen en las iteraciones k-1, k-2, ..., k-N+1, respectivamente. Por tanto, la solución de los coeficientes en el tiempo n depende de las soluciones previas.

A pesar de esto, resultados experimentales parecen indicar que este método funciona muy bien.

### 2.3.4.- Comparación ambos métodos

El algoritmo de error de salida es menos sensible a las componentes de ruido aditivo presente en la respuesta deseada que el de ecuación de error.

Sin embargo, la superficie de error tiene mínimos locales y el filtro adaptativo podría no converger al mínimo global, a menos que el sistema esté inicializado adecuadamente.

La función de coste del algoritmo de ecuación de error es cuadrática mientras que la del error de salida es no cuadrática.

También hay diferencias a la hora de interpretar estas funciones de coste. La del algoritmo de error de salida se minimiza aplicando el método de descenso del gradiente a un filtro IIR adaptativo. Sin embargo, la del algoritmo de ecuación de error se interpreta mejor como una configuración FIR adaptativa de dos canales.

Se está avanzando mucho en la investigación del algoritmo de error de salida.

### 2.3.5.- Inconvenientes de las ecuaciones en diferencias no lineales y recursivas

Las ecuaciones en diferencias plantean una serie de problemas que no se ven con los filtros de Volterra. El más importante es el tema de la estabilidad. Con los primeros hay que estar continuamente controlándolo.

Con los sistemas no lineales, podemos encontrar entradas acotadas que lleven al sistema a la inestabilidad. Esto no ocurre con los sistemas lineales. Por eso a menudo se habla de una estabilidad dependiente de la entrada.

Muchas de las técnicas disponibles no pueden manejar este problema sin la intervención humana.

## 2.4.- Filtros polinomiales adaptativos de lattice

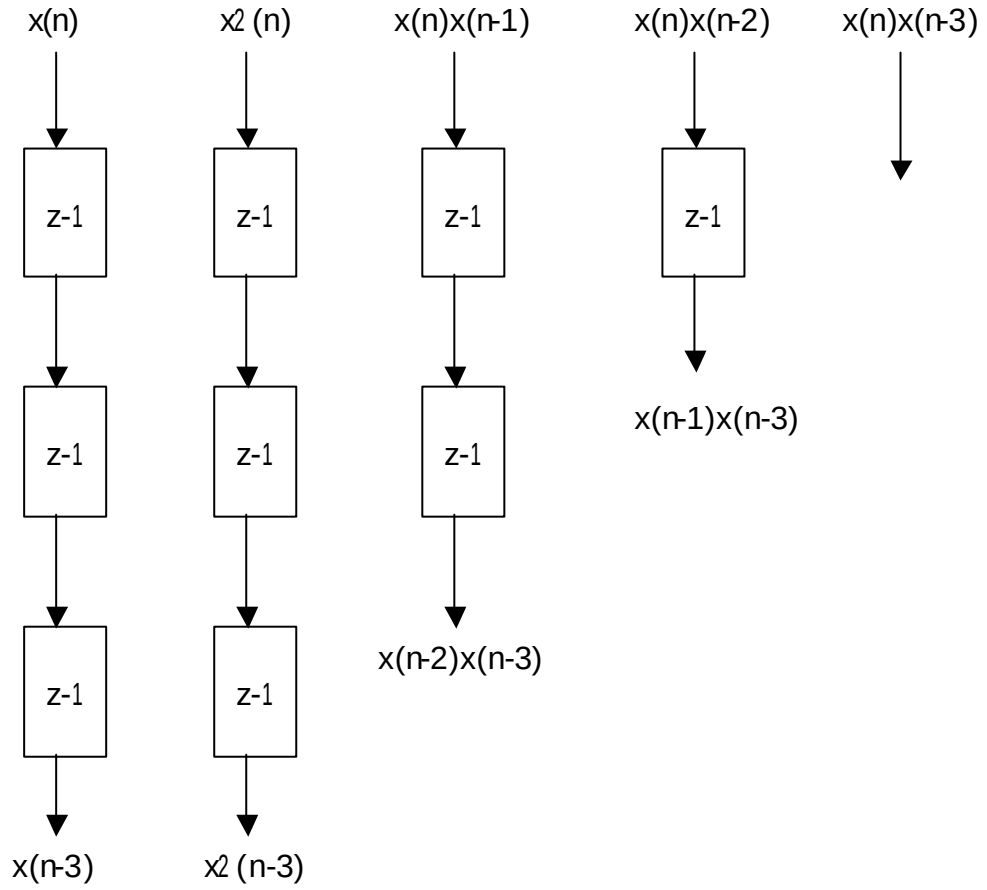
### 2.4.1.- Introducción

Los filtros adaptativos de lattice ortogonalizan las señales de entrada al filtro y entonces estiman la respuesta deseada como una combinación lineal de estas señales ortogonales. Las ventajas de estos filtros en aplicaciones de filtrado adaptativo son muchas. Los filtros lattice equipados con algoritmos de tipo LMS convergen más rápido y con una menor dependencia de la señal de entrada que los filtros de forma directa. También tienen mejores propiedades numéricas: la adaptación de los parámetros del filtro en cada etapa se puede hacer independientemente del resto de las etapas. Además, la estructura es bastante modular y, por eso, estos filtros son muy adecuados para una implementación VLSI. [1]

Esto es en teoría. Luego, en la práctica, surgen algunos problemas al adaptar los coeficientes del filtro en una etapa independientemente del resto.

### 2.4.2.- Estructura lattice

Aquí, vamos a desarrollar una estructura lattice para un sistema truncado de Volterra de segundo orden. Estas ideas se podrían igualmente aplicar a otros tipos de sistemas polinomiales. Para poder desarrollar esta parametrización de Lattice, es conveniente visualizar el problema de filtrado no lineal como un problema de filtrado lineal multicanal. En un diagrama de bloques, se podría ver:




En este caso, como el orden es p=2 y la memoria del sistema es N=3, podemos ver que tenemos 5 canales. Las señales se toman tanto desde los puntos de entrada como desde los de salida de los elementos de retraso y se combinan linealmente para estimar la respuesta deseada. Esta estructura multicanal es diferente de la tradicional puesto que el número de elementos de retraso (y coeficientes) en cada canal es diferente de los otros.

Sin embargo, este problema se puede superar y, por eso, esta estructura es la base para los filtros de Volterra con algoritmos rápidos RLS y para algunas realizaciones de lattice de los filtros adaptativos de Volterra. Algunas

realizaciones lattice usan coeficientes y elementos de retraso adicionales para igualar el número de coeficientes en cada canal. Otras funcionan sólo con entradas gausianas.

Nosotros nos vamos a centrar en una estructura basada en un filtro de lattice multicanal desarrollado por Ling y Proakis y en un predictor no lineal desarrollado por Zarzycki.

Por simplicidad, consideraremos el caso en que N=3 y p=2. Las señales que intervienen en la estimación se pueden agrupar en tres columnas:

$$\begin{pmatrix} x[n] & x[n-1] & x[n-2] \\ x^2[n] & x^2[n-1] & x^2[n-2] \\ & x[n]x[n-1] & x[n-1]x[n-2] \\ & & x[n]x[n-2] \end{pmatrix} = \begin{pmatrix} \underline{X}_0^b[n] & \underline{X}_1^b[n] & \underline{X}_2^b[n] \end{pmatrix} \qquad (2.11)$$

La idea básica que se emplea para el filtro lattice de Volterra es obtener una descomposición ortogonal de Gram-Schmidt de $\begin{pmatrix} \underline{X}_0^b[n] & \underline{X}_1^b[n] & \underline{X}_2^b[n] \end{pmatrix}$

Sean $\underline{b}_0[n]$, $\underline{b}_1[n]$ y $\underline{b}_2[n]$ una base ortogonal para el vector anterior. Así, cualquier combinación lineal de este último se puede escribir de una manera equivalente como otra combinación lineal de los primeros y viceversa. Esto quiere decir que en lugar de estimar la respuesta deseada d[n] como una combinación lineal de los elementos de $\underline{X}_i^b[n],\ i=0,1,2$ , se puede estimar como una combinación lineal de los elementos de $\underline{b}_i[n]$. Por tanto, sea:

$$\hat{d}[n] = \left(k_0^d\right)^T \underline{b}_0[n] + \left(k_1^d\right)^T \underline{b}_1[n] + \left(k_2^d\right)^T \underline{b}_2[n] \qquad (2.12)$$

la mejor estimación posible, donde $k_0^d, k_1^d, k_2^d$ son vectores de coeficientes apropiados en donde se ha eliminado la posible dependencia con el tiempo. Una de las grandes ventajas de la estructura lattice es que como los $\underline{b}_i[n]$ son ortogonales entre si, el vector $k_i^d$ se puede obtener únicamente a partir de los estadísticos de d[n] y $\underline{b}_i[n]$. Por ejemplo, la solución mínima cuadrática para $k_0^d$ sería:

$$k_0^d = \mathrm{E}\left\{\underline{b}_0[n]\underline{b}_0^T[n]\right\}^{-1} \mathrm{E}\{d[n]\underline{b}_0[n]\}$$

Es conocido que una forma de obtener $\underline{b}_i[n]$ es definirlo como el vector de error de predicción hacia atrás de orden i-ésimo de $\underline{X}_i^b[n]$. De esta manera, $\underline{b}_i[n]$ sería el error de estimación cuando $\underline{X}_i^b[n]$ se estima usando las columnas previas de (2.11). Así, por ejemplo, $\underline{b}_0[n]$ se define:

$$\underline{b}_0[n] = \underline{X}_0^b[n] = \begin{pmatrix} x[n] \\ x^2[n] \end{pmatrix}$$

Utilizando los errores de predicción hacia atrás anteriores e incorporando los errores de predicción hacia delante, se van estimando $\underline{b}_1[n]$ y $\underline{b}_2[n]$.

Una vez que se tienen estos vectores, se calculan los coeficientes $k_i^d$ tal y como se ha visto antes y ya se tiene una estimación de la respuesta deseada, que es lo que se va buscando.

### 2.4.3.- Filtro adaptativo basado en esta estructura

Una vez que la estructura lattice se ha desarrollado, obtener un filtro adaptativo basado en esta estructura no es muy difícil. Los algoritmos del gradiente se pueden extender fácilmente para el caso no lineal. La principal idea que se emplea en los filtros lattice de tipo LMS es que en cada etapa, los coeficientes se puedan optimizar independientemente de las otras etapas. Esto se debe a la ortogonalidad de las señales en las diferentes etapas cuando se usan coeficientes de lattice óptimos. Vamos a desarrollar un poco más esta idea.

Es lógico pensar que en la ecuación (2.12), cada coeficiente $k_i^d$ sea el óptimo en la estimación de la respuesta deseada como una combinación lineal de $\underline{b}_i[n]$. Así:

$$e_i[n] = d[n] - \sum_{j=0}^{i-1} \left(k_j^d\right)^T \underline{b}_j[n]$$

Como $\underline{b}_0[n],\ldots,\underline{b}_{i-1}[n]$ son ortogonales a $\underline{b}_i[n]$, entonces $e_i[n]$ contiene todos los componentes de d[n] que se pueden estimar usando $\underline{b}_i[n]$. Por tanto, si estimamos $e_i[n]$ usando $\underline{b}_i[n]$, obtendremos el mismo resultado que si hubiéramos estimado d[n] usando $\underline{b}_i[n]$. Así, la adaptación de $k_i^d$ se puede considerar como un problema de filtrado adaptativo por separado donde $e_i[n]$ es la respuesta deseada, $\underline{b}_i[n]$ es la entrada al filtro y $e_{i+1}[n]$ es el error. Las ecuaciones para este filtro adaptativo de tipo LMS serían:

$$e_{i+1}[n] = e_i[n] - \left(k_i^d[n]\right)^T \underline{b}_i[n]$$

$$k_i^d[n+1] = k_i^d[n] + me_{i+1}[n]\underline{b}_i[n]$$

Se pueden usar diferentes valores de $\mu$ para las distintas etapas.

### 2.4.4.- Complejidad computacional

Una de las desventajas de la estructura lattice cuando se compara con la forma directa es que la primera requiere $Pr(N^3)$ coeficientes para describir totalmente un sistema de Volterra de segundo orden con N retrasos mientras que la segunda necesita sólo $Pr(N^2)$ coeficientes. Por tanto, la complejidad computacional de los algoritmos adaptativos del gradiente basados en la estructura lattice también será proporcional a $N^3$ operaciones por iteración.

Esta complejidad es un poco menor que para la mayoría de filtros adaptativos de Volterra RLS. Por tanto, la ventaja computacional de los filtros adaptativos lattice de Volterra por el gradiente sobre los filtros adaptativos de Volterra RLS no es tan significativa como en el caso de implementaciones en la forma directa.

Korenberg ha desarrollado algoritmos usando la ortogonalización de Gram-Schmidt de la señal de entrada y se puede aplicar, en general, a modelos de sistemas polinomiales. Sin embargo, para modelos de sistemas bilineales y de Volterra, la estructura lattice vista aquí es un poco más eficiente que la técnica de Korenberg.

# CAPÍTULO 3.

# ALGORITMOS PARA LOS FILTROS ADAPTATIVOS DE VOLTERRA

## 3.1.- Algoritmo LMS

Se adaptan los coeficientes en cada iteración usando el algoritmo de descenso paso a paso, que intenta minimizar $e^2[n]$. Estas ecuaciones de adaptación son: [1]

$$h_1[m_1;n+1] = h_1[m_1;n] - \frac{m_1}{2}\frac{\partial e^2[n]}{\partial h_1[m_1;n]}$$

$$h_2[m_1,m_2;n+1] = h_2[m_1,m_2;n] - \frac{m_2}{2}\frac{\partial e^2[n]}{\partial h_2[m_1,m_2;n]}$$

siendo

$$e^2[n] = \left(d[n]-\hat{d}[n]\right)^2 = \left(d[n]-\left(h_0 + H^T[n]X[n]\right)\right)^2 =$$

$$= d^2[n] + h_0^2 + \left(H^T[n]X[n]\right)^2 + 2h_0 H^T[n]X[n] - 2d[n]\left(h_0 + H^T[n]X[n]\right)$$

Por tanto, las derivadas valdrán:

$$\frac{\partial e^2[n]}{\partial h_1[m_1]} = 2\left(H^T[n]X[n]\right)x[n-m_1] + 2h_0\,x[n-m_1] - 2d[n]x[n-m_1] =$$

$$= 2x[n-m_1]\left(H^T[n]X[n] + h_0 - d[n]\right) = -2x[n-m_1]e[n]$$

$$\frac{\partial e^2[n]}{\partial h_2[m_1,m_2]} = 2\left(H^T[n]X[n]\right)x[n-m_1]x[n-m_2] + 2h_0\,x[n-m_1]x[n-m_2] -$$

$$- 2d[n]x[n-m_1]x[n-m_2] = 2x[n-m_1]x[n-m_2]\left(H^T[n]X[n] + h_0 - d[n]\right) =$$

$$= -2x[n-m_1]x[n-m_2]e[n]$$

Sustituyendo en las ecuaciones de arriba, se llega a:

$$h_1[m_1;n+1] = h_1[m_1;n] + \mu_1 e[n]x[n-m_1]$$

$$h_2[m_1,m_2;n+1] = h_2[m_1,m_2;n] + \mu_2 e[n]x[n-m_1]x[n-m_2]$$

donde $\mu_1$ y $\mu_2$ son constantes pequeñas y positivas que controlan la velocidad de convergencia y las características de seguimiento y en régimen permanente del filtro.

Para situaciones más generales, se pueden obtener unas ecuaciones similares. También se usan variaciones de este algoritmo LMS.

Se suele utilizar una notación vectorial para una mayor simplicidad y facilidad en el análisis de este algoritmo. Con esta notación, las ecuaciones del algoritmo quedan:

*Vector de coeficientes:*

$$H[n] = [h_1[0;n], \ldots, h_1[N-1;n], h_2[0,0;n], \ldots, h_2[0,N-1;n],$$

$$h_2[1,1;n], \ldots, h_2[N-1,N-1;n]]^T$$

*Vector de entrada:*

$$X[n] = [x[n], \ldots, x[n-N+1], x^2[n], x[n]x[n-1], \ldots, x[n]x[n-N+1],$$

$$x^2[n-1], \ldots, x^2[n-N+1]]^T$$

*Inicialización:*

H[0] se elige arbitrariamente

*Algoritmo:*

$$e[n] = d[n] - H^T[n]X[n]$$

$$H[n+1] = H[n] + mX[n]e[n]$$

Nota: $\mu$ es una matriz diagonal con $\mu_1$ apareciendo en las primeras N filas y $\mu_2$ en el resto.

Esta estructura sólo difiere del caso lineal en la manera en que se definen los vectores por lo que comprobamos que, como se ha dicho antes, es una extensión directa del caso lineal.

Bajo ciertas simplificaciones, es inmediato demostrar que los valores medios de los coeficientes convergen a sus valores óptimos (para un ambiente estacionario) si las constantes de convergencia se eligen de tal manera que cumplan:

$$0 < m_1, m_2 < \frac{2}{l_{max}}$$

donde $\lambda_{max}$ es el mayor autovalor de la matriz de autocorrelación del vector de entrada X[n].

Al igual que en el caso lineal, el problema radica en que estos autovalores controlan la velocidad de convergencia del algoritmo. En general, cuanto mayor sea el número de condición (cociente $\lambda_{max}$ / $\lambda_{min}$ ) de la matriz de correlación de las entradas, menor es la velocidad de convergencia. Esto es un gran inconveniente para el caso no lineal puesto que,

en general, este cociente es muy grande. Incluso cuando la señal de entrada es blanca, la presencia de no linealidades en el vector de entrada hará que este cociente sea mayor que uno.

Por tanto, es importante buscar estructuras y algoritmos alternativos que converjan con menos dependencia de los estadísticos de la señal de entrada. Una opción es usar el algoritmo RLS (que se verá más adelante); y otra alternativa es utilizar este algoritmo LMS pero con estructuras lattice (u otras ortogonalizadas) para implementar los filtros no lineales.

## 3.2.- Algoritmo RLS

El filtro adaptativo LMS se puede considerar como una solución aproximada para el problema de optimización estadística que intenta minimizar el valor cuadrático medio del error estimado en cada iteración. Por otro lado, los filtros adaptativos RLS dan la solución exacta a un problema de optimización formulado de una manera determinista. Es decir, los algoritmos LMS utilizan la esperanza matemática para la estimación de la función de coste mientras que los RLS no. Por eso, si el problema está formulado de una manera determinista, la solución RLS es exacta. Esta formulación da origen al filtro adaptado RLS con factor de olvido exponencial. El factor de olvido se utiliza para que los datos pasados sean olvidados y así permitir el seguimiento de las variaciones estadísticas de los datos cuando el filtro opera en un ambiente no estacionario. Utilizaremos un factor de olvido exponencial.

En el caso de filtro de Volterra de segundo orden, se trata de minimizar la siguiente función de coste: [1]

$$J[n] = \sum_{k=0}^{n} \lambda^{n-k} \left(d[k] - H^T[n]X[k]\right)^2$$

donde $\lambda$ ($0 << \lambda < 1$) es el factor que, como se ha dicho antes, controla la duración de la memoria del filtro adaptativo.

La solución a este problema para cada iteración se encuentra fácilmente derivando J[n] respecto de H[n], igualándolo a cero y resolviendo para H[n]. La solución óptima para la iteración n-ésima viene dada por:

$$H[n] = C^{-1}[n-1]P[n]$$

con

$$C[n] = \sum_{k=0}^{n} \lambda^{n-k} X[k]X^T[k]$$

$$P[n] = \sum_{k=0}^{n} \lambda^{n-k} d[k]X[k]$$

Aquí, C[n] es la matriz de autocorrelación LS del vector de entrada, X, y P[n] es el vector de correlación cruzada LS entre el vector de entrada, X, y la respuesta deseada, d[n].

H[n] se puede adaptar recursivamente haciendo:

$$C[n] = \lambda C[n-1] + X[n]X^T[n]$$

$$P[n] = \lambda P[n-1] + d[n]X[n]$$

Se puede hacer un poco menos complejo haciendo uso del lema de inversión de matrices para invertir C[n].

Este lema es un resultado muy conocido en el álgebra matricial. Se puede expresar de distintas maneras. Sean A y B dos matrices definidas positivas de MxM relacionadas por:

$$A = B^{-1} + C D^{-1} C^H$$

donde D es otra matriz definida positiva de NxM y C es una matriz de MxN. De acuerdo con el lema de inversión de matrices, se puede expresar la inversa de la matriz A como sigue:

$$A^{-1} = B - BC\left(D + C^H BC\right)^{-1} C^H B$$

Otra forma de expresarlo sería:

$$A = B + C D C^H$$

y su inversa:

$$A^{-1} = B^{-1} - B^{-1} C\left(C^H B^{-1} C + D^{-1}\right)^{-1} C^H B^{-1}$$

El algoritmo RLS se puede resumir en el siguiente cuadro:

*Vector de coeficientes*

$$H[n] = [h_1[0;n], \ldots, h_1[N-1;n], h_2[0,0;n], \ldots, h_2[0,N-1;n],$$

$$h_2[1,1;n], \ldots, h_2[N-1,N-1;n]]^T$$

*Vector de entrada*

$$X[n] = [x[n], \ldots, x[n-N+1], x^2[n], x[n]x[n-1], \ldots, x[n]x[n-N+1],$$

$$x^2[n-1], \ldots, x^2[n-N+1]]^T$$

*Inicialización*

$$H[0] = [0, 0, \ldots, 0]^T$$

$$C^{-1}[0] = \delta^{-1} I$$

δ=constante pequeña positiva

*Algoritmo*

$$k[n] = \frac{l^{-1} C^{-1}[n-1] X[n]}{1 + l^{-1} X^T[n] C^{-1}[n-1] X[n]}$$

$$e[n] = d[n] - H^T[n-1] X[n]$$

$$H[n] = H[n-1] + mk[n] e[n]$$

$$C^{-1}[n] = l^{-1} C^{-1}[n-1] - l^{-1} k[n] X^T[n] C^{-1}[n-1]$$

$$e[n] = d[n] - H^T[n] X[n]$$

## 3.3.- Complejidad computacional. Soluciones

El algoritmo LMS tiene $Pr(N^2)$ multiplicaciones (el número de multiplicaciones es proporcional a $N^2$) por iteración mientras que el RLS tiene $Pr(N^4)$. El precio que se paga en términos de complejidad computacional por un mejor comportamiento es excesivo para muchas aplicaciones.

Existen algoritmos rápidos que simplifican considerablemente esta complejidad computacional haciendo uso del hecho de que la mayoría de los elementos de los vectores X[n] y X[n-1] son iguales. Hay algoritmos con $Pr(N^3)$ multiplicaciones por iteración. Estos algoritmos proporcionan una mejor convergencia que los LMS y también parecen ser más robustos a las variaciones estadísticas de las señales de entrada. Sin embargo, la complejidad computacional($Pr(N^3)$) sigue siendo mucho mayor que la del LMS($Pr(N^2)$).

Existen aproximaciones, computacionalmente más simples, a la solución adaptativa RLS. Éstas asumen que la señal de entrada es gausiana y que el sistema falla cuando esto no ocurre.

Un gran problema de los algoritmos rápidos RLS es que tienen unas propiedades numéricas muy pobres.

# CAPÍTULO 4.

# APLICACIONES FILTROS DE VOLTERRA DE SEGUNDO ORDEN

## 4.1.- Modelado de sistemas no lineales

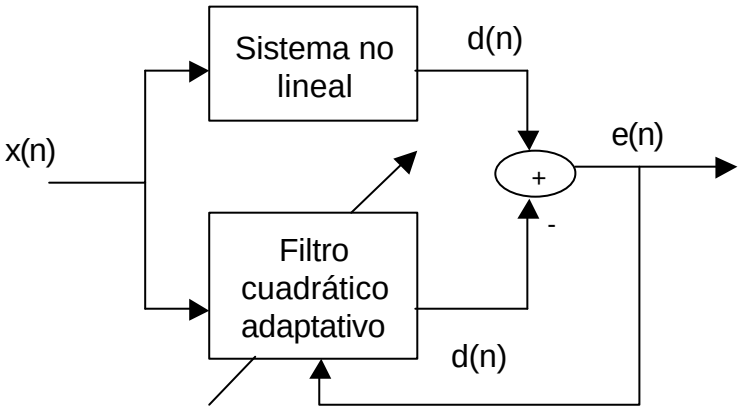
El modelado de sistemas no lineales es una de las primeras aplicaciones prácticas de la serie discreta de Volterra. En algunos casos, dependiendo de la naturaleza de las no-linealidades, se pueden necesitar núcleos de orden alto. Esto ocurre, por ejemplo, en el campo de la comunicación cuando se modelan canales de referencia fuertemente distorsionados, amplificadores de transmisión no lineales, y canales paso banda no lineales en sistemas de transmisión digital. [2]

Sin embargo, hay muchas aplicaciones donde es necesario usar núcleos cuadráticos. Por ejemplo, se ha demostrado que una función de transferencia cuadrática es apropiada para modelar interacciones no lineales de tres ondas asociadas con turbulencias en fluidos y plasmas. Otras aplicaciones serían dispersión electromagnética desde objetivos no lineales, y modelado del comportamiento dinámico no lineal de los elementos cercanos a la costa sujetos a la excitación aleatoria de las olas del mar.

La última es un ejemplo típico de una situación en la cual el modelo cuadrático es apropiado, mientras que el modelo lineal no lo es. De hecho, la relación entre la excitación de las olas y la correspondiente respuesta de un barco amarrado se caracteriza por la presencia de las llamadas oscilaciones flotantes de baja frecuencia (LFDO's). La LFDO en respuesta a las fluctuaciones del mar se pueden explicar con la presencia de una fuerza de segundo orden proporcional al cuadrado de las alturas instantáneas de las olas. De acuerdo con esta relación cuadrática, las

componentes espectrales de la ola incidente generarán sumas y diferencias de frecuencias. Las frecuencias bajas correspondientes al ancho de banda de resonancia del barco amarrado, las cuales están por debajo de las frecuencias dominantes asociadas a la ola de entrada, se amplifican. Así la fuerza de segundo orden producirá la parte predominante en la respuesta del barco amarrado. Por tanto, los operadores lineales y cuadráticos del filtro, obtenidos de (2.9) y (2.10) pueden describir bien la LFDO mientras que un modelo más simplificado vendría dado por un dispositivo cuadrático seguido por un filtro lineal (el modelo de Hammerstein).

En contra a estas soluciones cerradas, las técnicas adaptativas también se pueden usar para modelar no linealidades de acuerdo con el siguiente esquema, en donde un filtro cuadrático adaptativo se usa para identificar un sistema no lineal.



## 4.2.- Otras aplicaciones

- *Cancelación de eco no lineal:*

En los módems digitales modernos, las técnicas de cancelación de eco aseguran una transmisión full-dúplex con una separación adecuada de canal. El propósito del cancelador es eliminar el cross talk, o señal de "eco", que interfiere con la señal procedente del transmisor lejano.

Para ello a menudo es necesario tener en cuenta distorsiones no lineales que se producen en la práctica como la transmisión de pulsos asimétricos, efectos de saturación en los transformadores, y no linealidades en los convertidores.

Existen métodos para expandir una función no lineal arbitraria de bits en una serie similar a la expansión de Volterra. Esta expansión utiliza sólo un número finito de términos incluyendo productos de pares de bits que corresponden a la parte cuadrática.

- *Cancelación adaptativa de ruido:*

En los sistemas de comunicación digital, por ejemplo, las no linealidades en el canal a menudo se pueden modelar con menos memoria. Como estos efectos tienen lugar en una red donde se usan operaciones de filtrado lineal, el efecto total de la señal de entrada sobre el canal es un mapeo no lineal con memoria el cual se puede describir por una serie discreta de Volterra.

De hecho, la técnica de la serie de Volterra se ha aplicado para ecualización adaptativa de no linealidades en el canal. También se han hecho cosas en ecualización de canales en transmisiones por satélite y en cancelación no lineal intersímbolo. Esto último se basa en una versión ortogonalizada de la serie de Volterra.

- *Detección y estimación:*

El problema de la detección aplicado a señales consiste en distinguir entre estas dos situaciones: i)el vector x contiene la parte útil (la "señal") y también la parte no deseada (el "ruido"); y ii)el vector x contiene sólo ruido.

Se han aplicado varios criterios para el diseño de filtros óptimos para detección. Entre éstos, la relación señal a ruido (SNR) y sus generalizaciones, conocidas como criterio de desviación simple y generalizado (SD y GD) han sido los más usados. En este contexto, los filtros cuadráticos se han investigado para usarlos en problemas relacionados con la detección ruido en ruido para aplicaciones sónar.

- *Extracción de bordes y eliminación de ruido:*

La detección de los bordes en una imagen es una herramienta básica en el procesado de señal en dos dimensiones, teniendo muchas aplicaciones en visión robótica, inspección automática, codificación de imagen,…

Existen muchos algoritmos para esta aplicación; entre ellos, el más usado es el operador Sobel, que es precisamente un filtro cuadrático.

El problema contrario también es una importante aplicación: eliminar ruido en una imagen sin dañar los bordes. Es uno de los típicos problemas que pueden ser tratados con técnicas no lineales.

En particular, se utiliza mucho con fotografías tomadas en malas condiciones.

- *Predicción de imágenes:*

En transmisión digital de imágenes se utilizan métodos para reducir la tasa de bit. Estos métodos requieren la presencia de un predictor. Pues bien, un predictor no lineal, basado en un filtro cuadrático, ofrece resultados interesantes sobre todo cuando hay cierta cantidad de movimiento presente en la imagen. En este caso, la parte cuadrática del operador compensa los efectos de alisado producidos por un predictor lineal convencional.

# CAPÍTULO 5.

# SIMULACIONES

Se van a realizar dos ejemplos para comparar los algoritmos LMS y RLS para filtros de Volterra. También se verá el algoritmo LMS normalizado. Es análogo al LMS con la diferencia de que las ecuaciones de adaptación se normalizan respecto de la potencia de la señal de entrada.

Se realizará la comparación tanto en la velocidad de convergencia como en el número de operaciones requeridas. Además se jugará con el parámetro $\mu$ (paso de adaptación) para encontrar el óptimo en ambos algoritmos. En la mayoría de algoritmos RLS, el paso de adaptación es igual a la unidad. Aquí se ha trabajado con un algoritmo que utiliza este parámetro y se tratará de encontrar el óptimo.

Se trabaja con procesos aleatorios distribuidos uniformemente como entradas. Se cambiará la media de este proceso para estudiar más casos.

En el primer ejemplo, el sistema es bastante más simple que en el segundo.

## 5.1.- Ejemplo 1

Se trata de identificar el siguiente sistema utilizando el LMS, el LMS normalizado y el RLS Volterra:

$$y(n) = 1 + x(n) + x^2(n)$$

donde x(n) es un proceso aleatorio distribuido uniformemente con varianza 1, $s_x^2 = 1$, y media 0, 1 y 10 para estudiar tres casos distintos y ver como influye la entrada en la estimación de los coeficientes.

La señal deseada se obtiene sumando a la salida del sistema un ruido gausiano de media cero y varianza $s_w^2$:

$$d(n) = y(n) + w(n)$$

La varianza del ruido, $s_w^2$, toma tres valores distintos y estudiaremos los tres casos. Estos tres valores son: 0,0.05 y 1.

Se van a considerar varios pasos de adaptación para irlos comparando.

Además de esto, se planteaba otro problema: la existencia del 1 en la ecuación del sistema. Esta constante equivale a la $h_0$ en la expansión en serie de Volterra. En el desarrollo teórico, se tomó igual a cero y se quedó olvidada. Ahora hay que deducir una ecuación para ella de igual manera que se hizo para el resto de coeficientes. Así, se tiene:

$$h_0[n+1] = h_0[n] - \frac{m}{2}\frac{\partial e^2[n]}{\partial h_0[n]}$$

siendo

$$e^2[n] = \left(d[n] - \hat{d}[n]\right)^2 = \left(d[n] - \left(h_0 + H^T[n]X[n]\right)\right)^2 =$$

$$= d^2[n] + h_0^2 + \left(H^T[n]X[n]\right)^2 + 2h_0 H^T[n]X[n] - 2d[n]\left(h_0 + H^T[n]X[n]\right)$$

Por tanto, la derivada valdrá:

$$\frac{\partial e^2[n]}{\partial h_0} = 2h_0 + 2H^T[n]X[n] - 2d[n] = -2e[n]$$

Y sustituyendo en la ecuación de arriba, se llega a:

$$h_0[n+1] = h_0[n] + me[n]$$

Las ecuaciones para los demás coeficientes ya se dedujeron antes pero se reproducen aquí de nuevo para tenerlas todas juntas:

$$h_1[m_1;n+1] = h_1[m_1;n] + m\ e[n]x[n-m_1]$$

$$h_2[m_1,m_2;n+1] = h_2[m_1,m_2;n] + m\ e[n]x[n-m_1]x[n-m_2]$$

En este ejemplo se usa un mismo valor del paso de adaptación μ para todos los coeficientes.

Vamos a ver los distintos resultados para el LMS, el LMS normalizado y el RLS. Luego, se compararán los óptimos en cada caso.

- **LMS:**

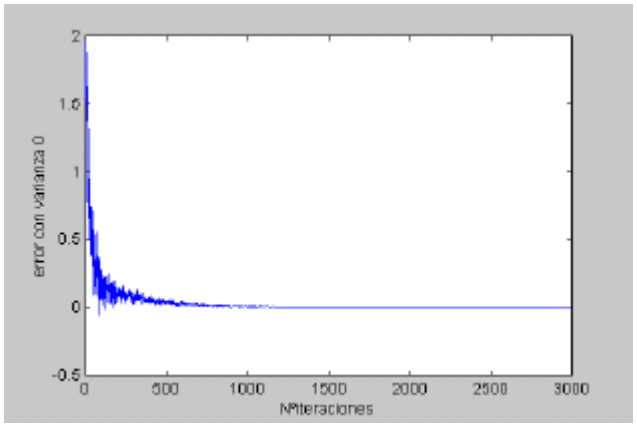
Vamos a empezar por el caso en el que no hay ruido. Se supone que la adaptación aquí será bastante buena.

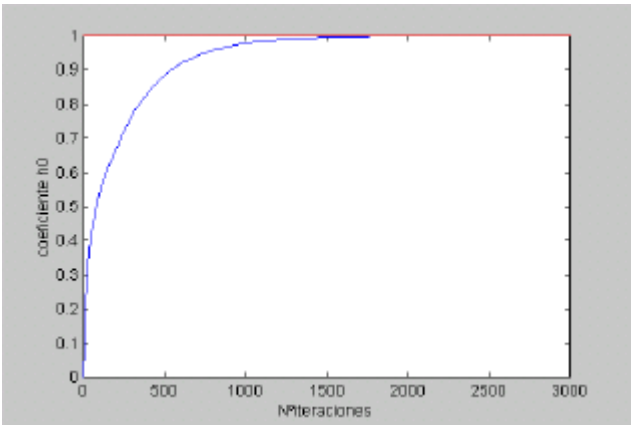
- $s_w^2 = 0$:

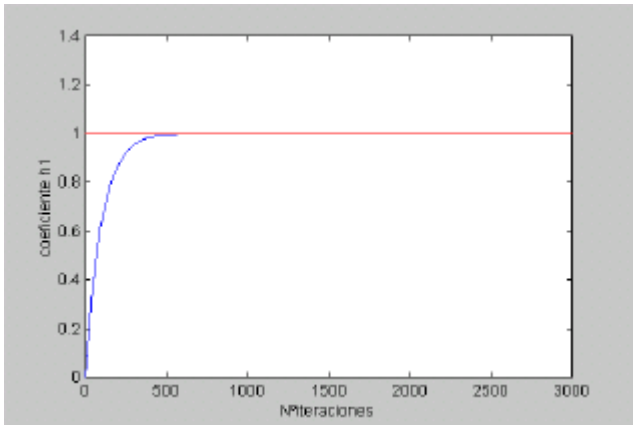
Se va a empezar con un ***proceso de entrada de media 0*** para irla subiendo más adelante.

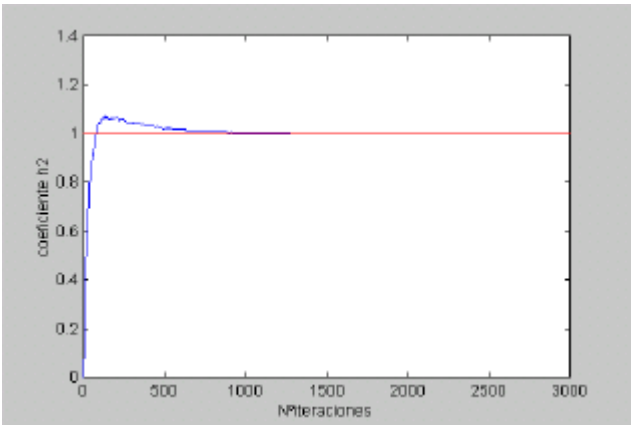
Se va a representar una gráfica con el error (diferencia entre la respuesta deseada y la salida del filtro) y otras tres gráficas más mostrando la convergencia de cada uno de los tres coeficientes. La línea roja es el valor final del coeficiente mientras que la azul es la evolución del coeficiente con cada iteración.

Así, para **m=0.01**, se obtiene:



Se observa que los tres coeficientes convergen perfectamente a su valor final, que es 1 en los tres casos. Los coeficientes $h_1$ y $h_2$ convergen antes que $h_0$. Esto se debe a que la estimación de $h_0$ se realiza sin tener en cuenta el proceso de entrada directamente.

El número de operaciones flotantes que emplea Matlab es de 40230950. Este número será siempre el mismo mientras se trabaje con el algoritmo LMS.

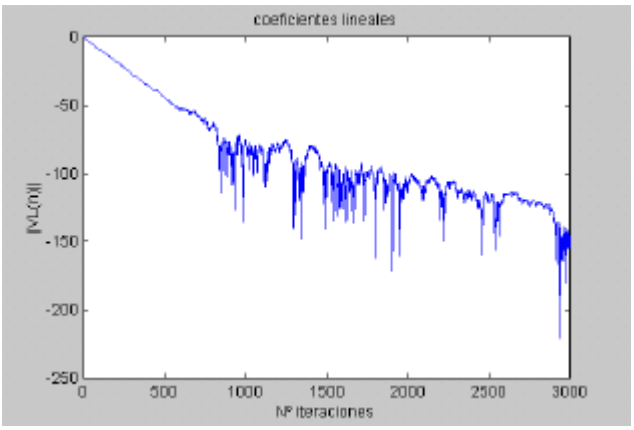
Si utilizamos como criterio de convergencia que la señal de error se mantenga menor que $\pm 0.03$, el algoritmo converge en la iteración 689.

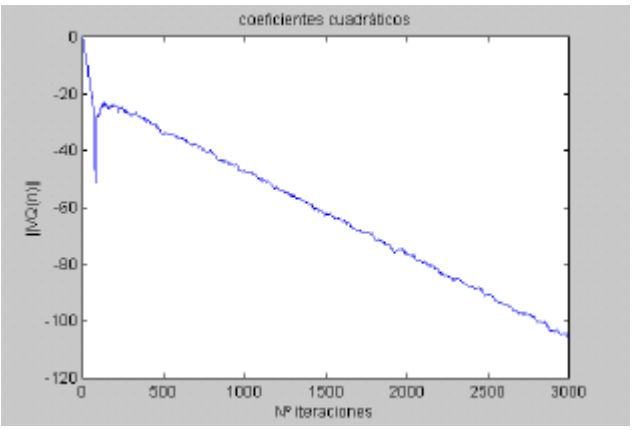
Se podría haber optado por representar los coeficientes de una manera más complicada de acuerdo con las siguientes expresiones:

$$\|V_L(n)\| = 10\log \frac{\sum_{m_1=0}^{N-1} \left(\hat{h}_1[m_1;n] - h_1[m_1]\right)^2}{\sum_{m_1=0}^{N-1} \left(h_1[m_1]\right)^2}$$

$$\|V_Q(n)\| = 10\log \frac{\sum_{m_1=0}^{N-1}\sum_{m_2=0}^{N-1} \left(\hat{h}_2[m_1,m_2;n] - h_2[m_1,m_2]\right)^2}{\sum_{m_1=0}^{N-1}\sum_{m_2=0}^{N-1} \left(h_2[m_1,m_2]\right)^2}$$

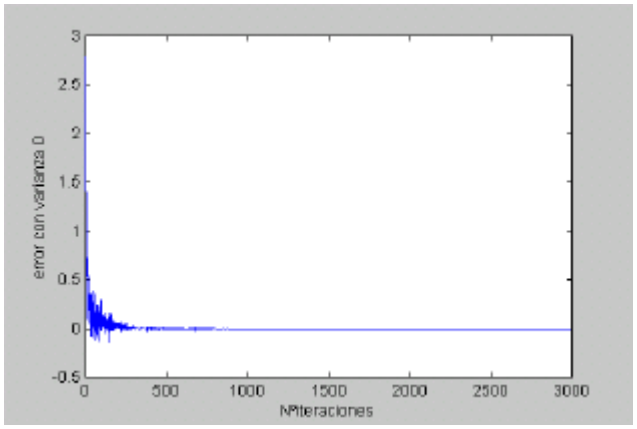
Es una comparación cuadrática-logarítmica entre los coeficientes estimados y los que realmente se tienen. El resultado en gráficas sería:

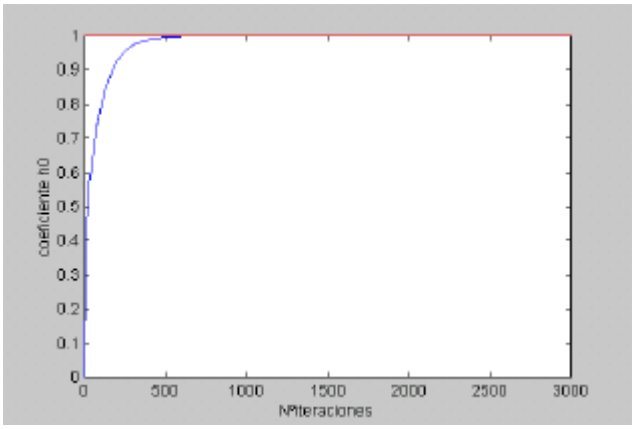


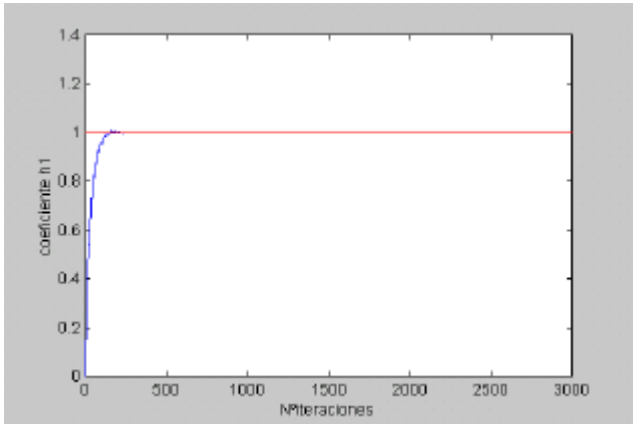



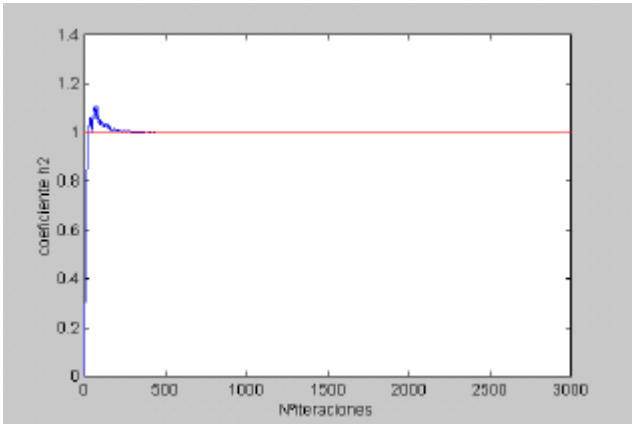
Para el resto de simulaciones, se representará cada coeficiente por separado porque se aprecia mejor la convergencia.

Se simula ahora con un **paso de adaptación** igual a **0.03**. Los resultados son:



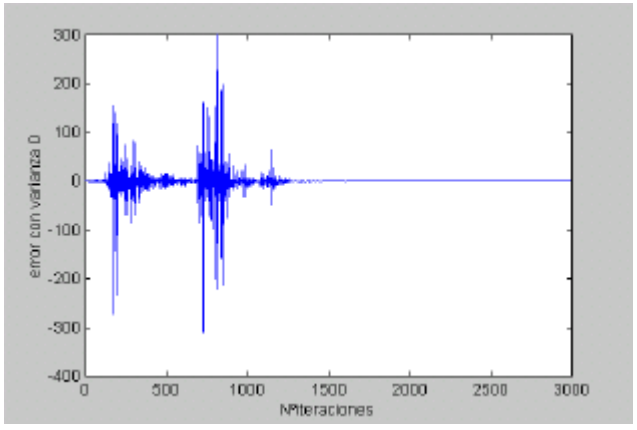



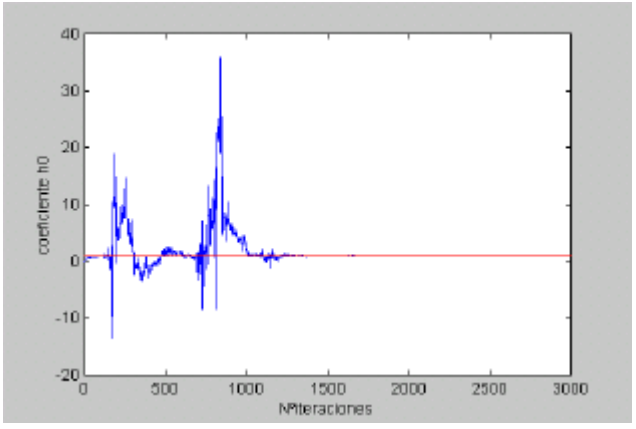



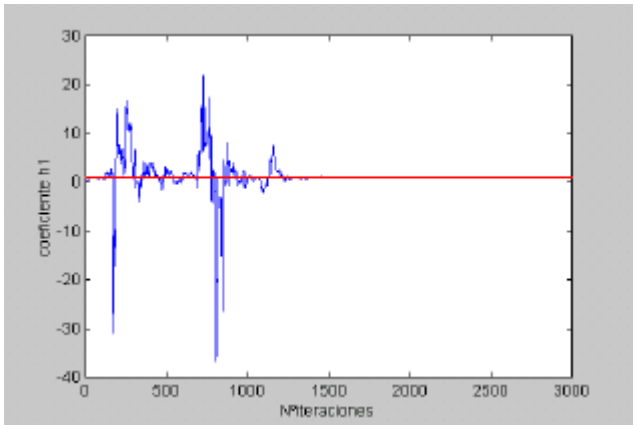



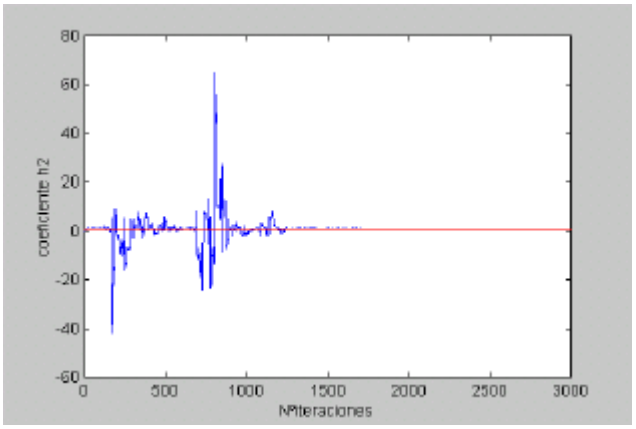

La convergencia se alcanza en la iteración 259 de acuerdo con el criterio del 3%.

Continuando con el ejemplo, para **m=0.05**, las gráficas serían:





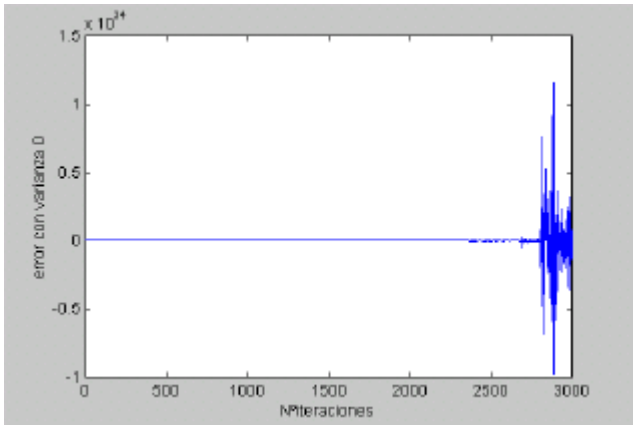



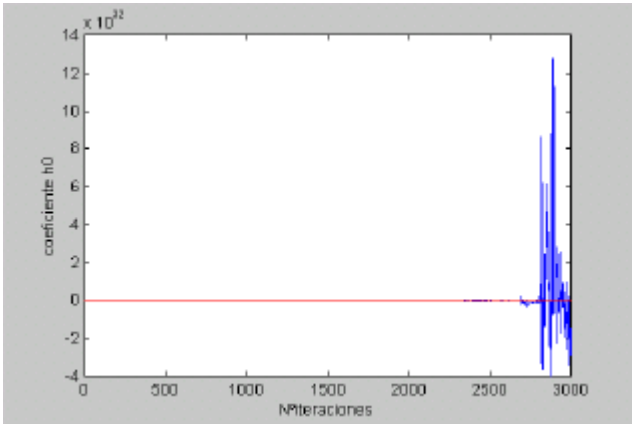

Para este valor del paso de adaptación, se aprecia que los coeficientes también convergen a su valor final. Ya no se aprecia apenas diferencia entre la velocidad de convergencia de cada coeficiente.

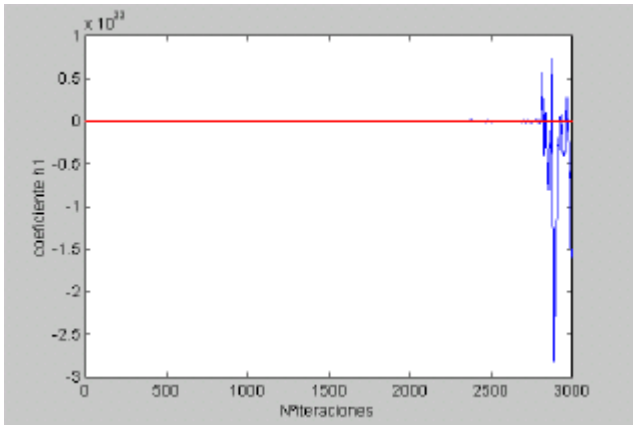
Siguiendo con el mismo criterio de convergencia anterior (señal de error menor que $\pm 0.03$), ésta se alcanza en la iteración 2069. Es decir, que al aumentar el paso de adaptación, la convergencia se hace más lenta.

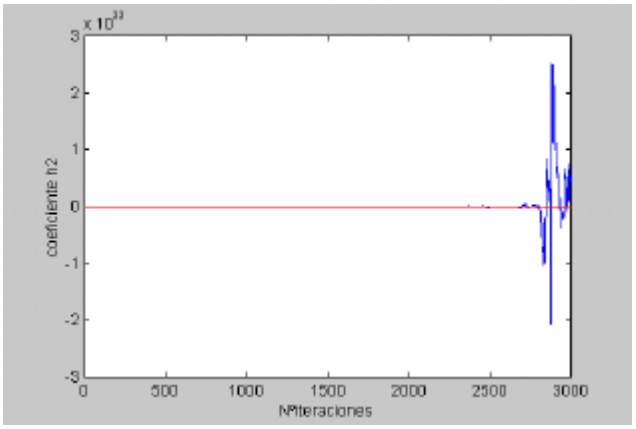
Es de esperar que conforme se vaya aumentando el paso de adaptación, la convergencia sea aún más lenta. Harán falta un número superior de iteraciones e incluso dejará de converger.

Para comprobar esto último, se ha simulado para **m=0.08** y éste ha sido el resultado:

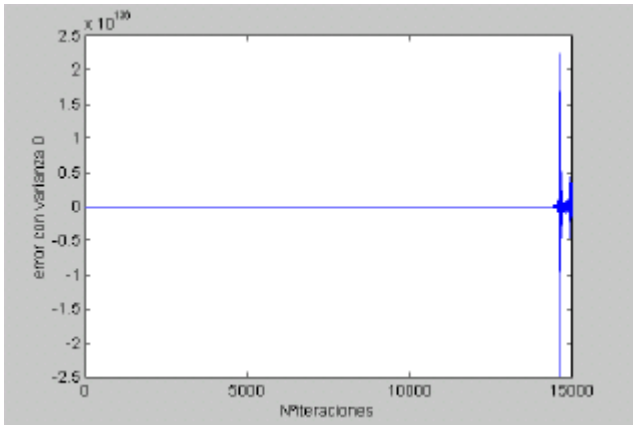


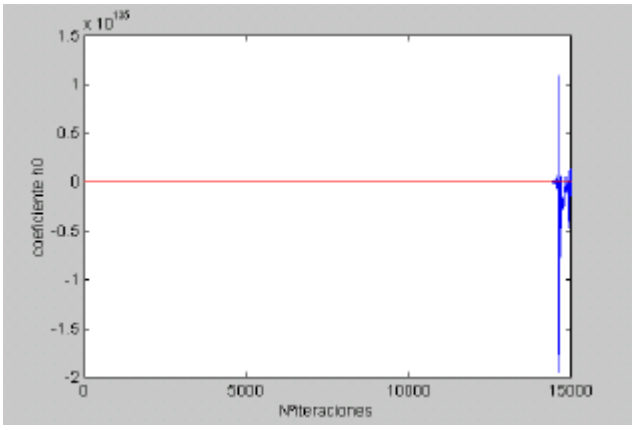



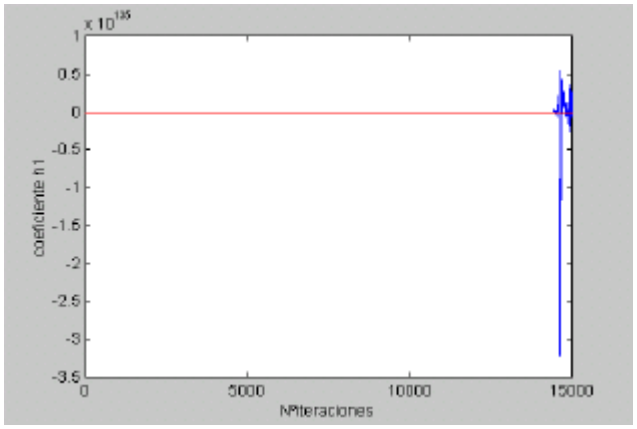


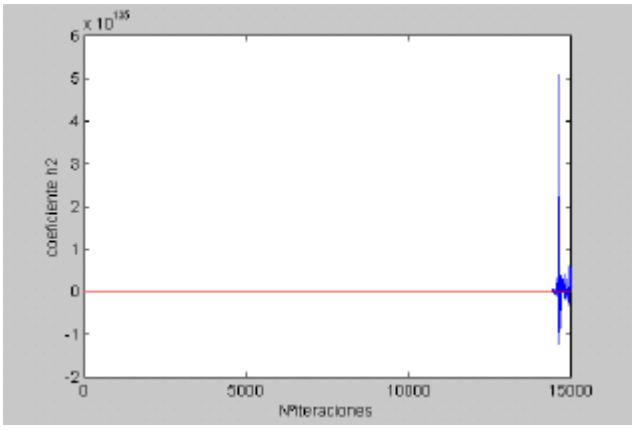




Se observa claramente que con 3000 iteraciones el algoritmo no llega a converger. Vamos a volverlo a simular con estos mismos valores pero con 15000 iteraciones. El resultado ha sido:



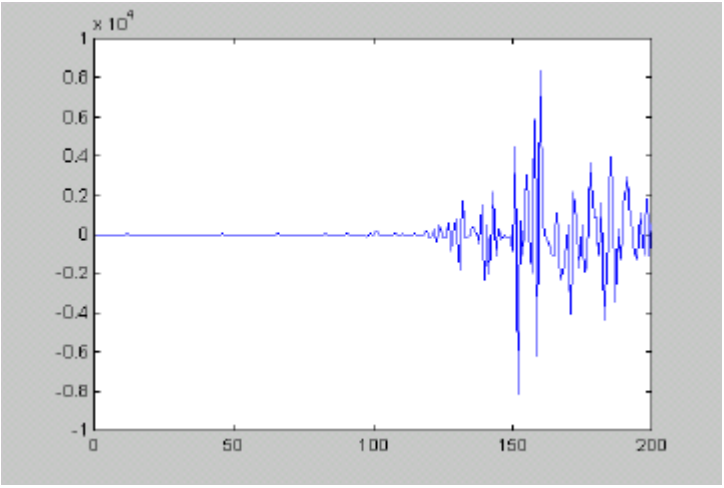




Se observa que el algoritmo tampoco converge con 15000 iteraciones. Al principio, parece que las gráficas valen cero

pero lo que ocurre es que los valores finales son muy grandes y queda todo un poco desescalado. Si analizamos las 200 primeras iteraciones de la señal de error, se observará que la señal de error va creciendo:




Luego, para este paso de adaptación, el algoritmo diverge.

En el análisis teórico del algoritmo LMS, se vio que el paso de adaptación se debe mantener menor que un cierto valor para poder alcanzar la convergencia. Este valor era $2/\lambda_{max}$, siendo $\lambda_{max}$ el mayor autovalor de la matriz de autocorrelación del proceso de entrada X[n].

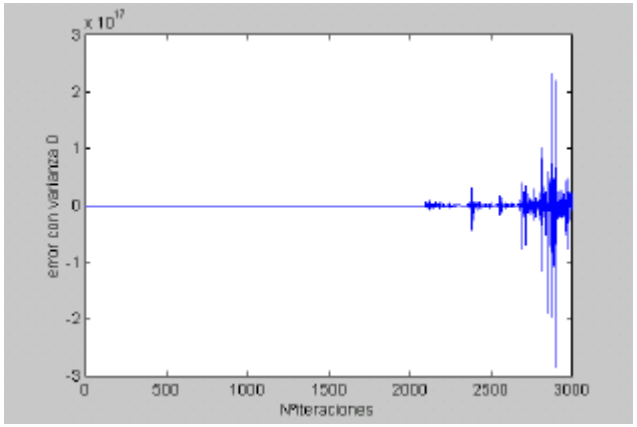
En este caso se tiene que:

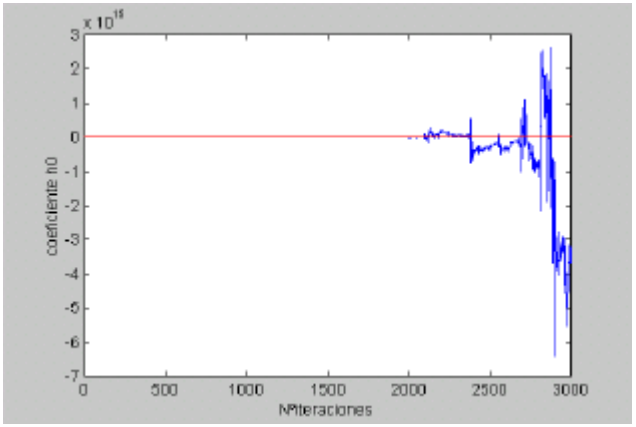
$$\lambda_{max} = 16.1614 \rightarrow \mu_{max} = 0.1238$$

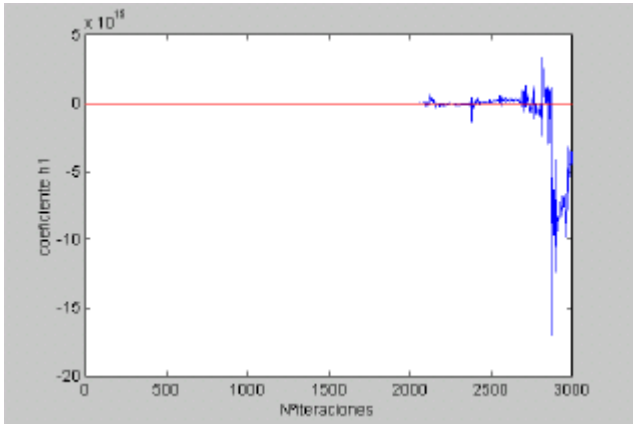
Se ha obtenido que, en la práctica, este valor del paso de adaptación debe ser menor para alcanzar la convergencia. Además, ésta no es más rápida cuanto mayor sea el paso de adaptación hasta llegar al valor crítico. Antes de divergir, el algoritmo converge más lento al ir aumentando el paso de adaptación y acercarlo más a dicho valor crítico.

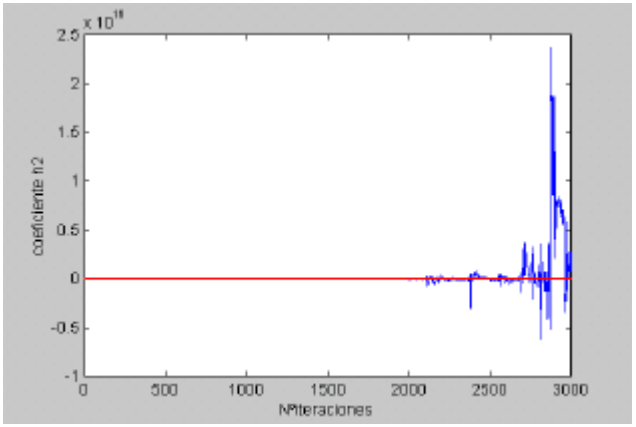
Vamos a analizar ahora el caso en el que ***la media del proceso de entrada es 1***. Es decir, se va a comprobar como afecta la entrada en el proceso de convergencia.

Se empezará con un **paso de adaptación** igual a **0.02**. La señal de error y la convergencia de los coeficientes sale:

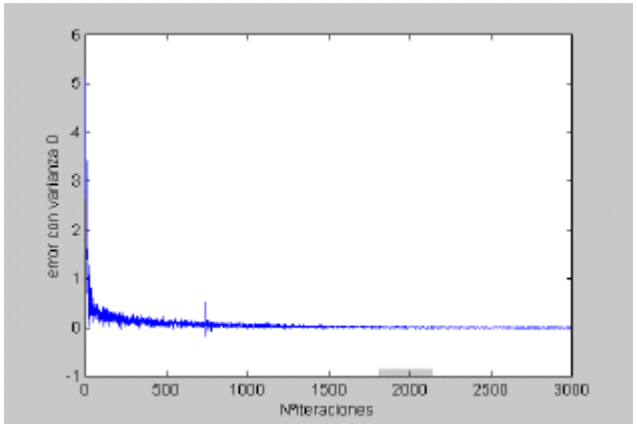


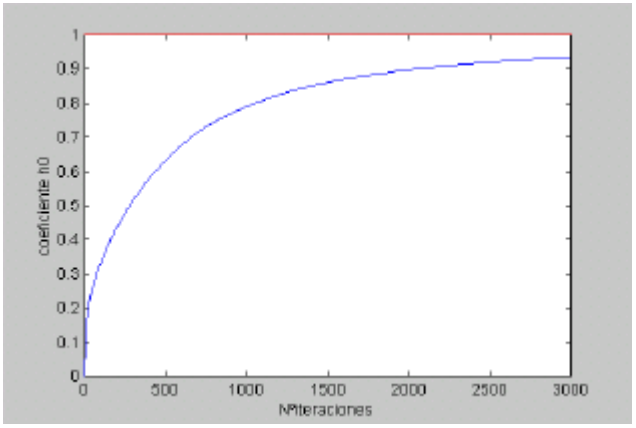



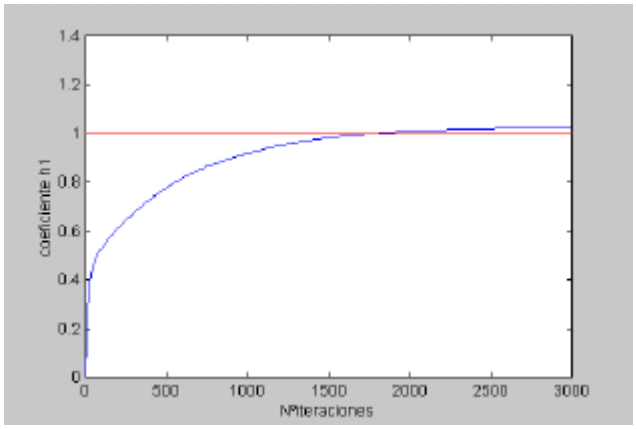



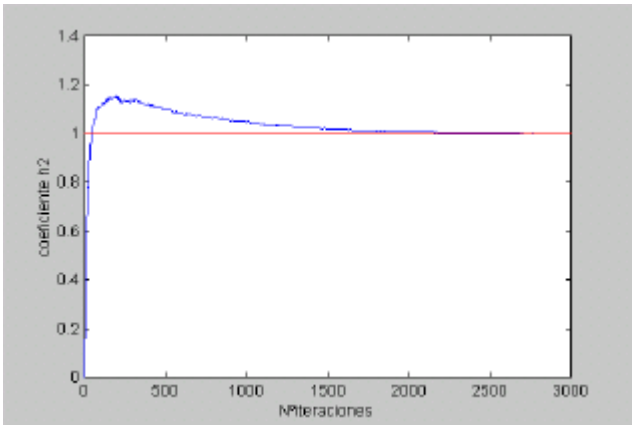




Se aprecia en las gráficas que el algoritmo no converge puesto que la señal de error va creciendo conforme pasan las iteraciones. Esto quiere decir que la media de la señal de entrada influye bastante en el algoritmo. Vamos a bajar el **paso de adaptación** a **0.005** para ver si se consigue la convergencia. Los resultados son:



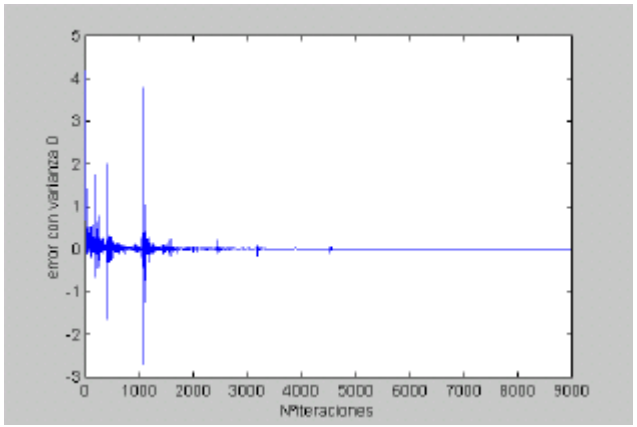



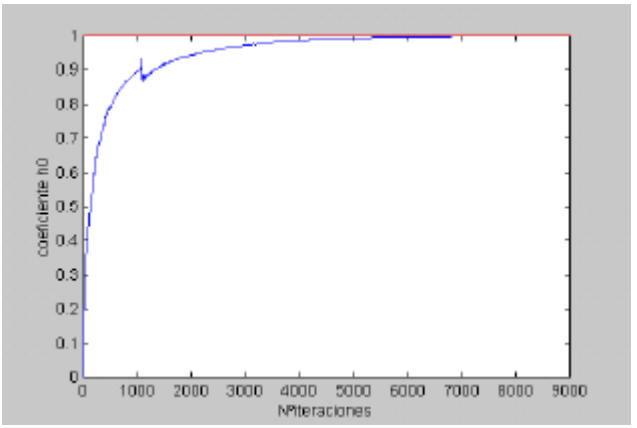


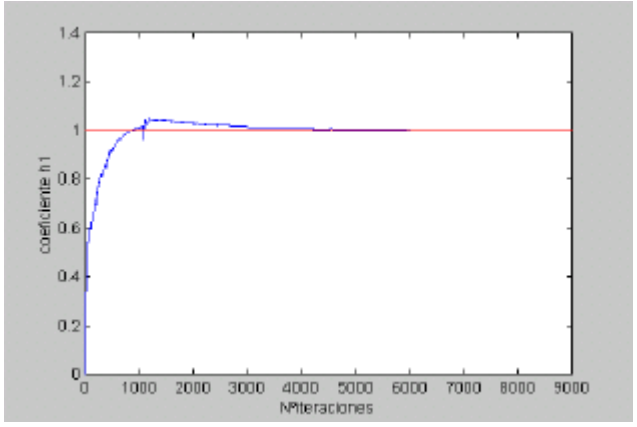



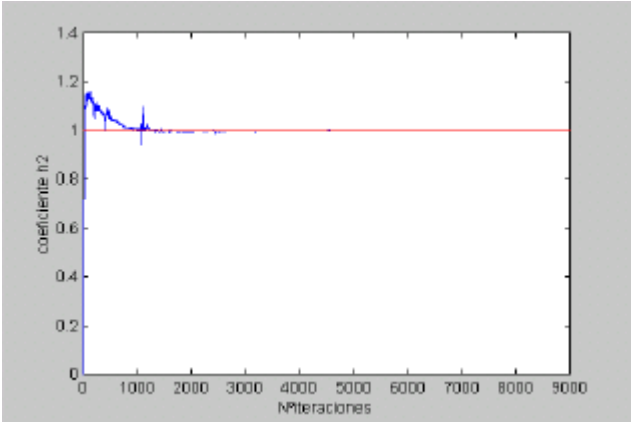

Ahora se ve que el algoritmo converge. Siguiendo con el mismo criterio anterior (señal de error menor que $\pm 0.03$), la convergencia se obtiene en la iteración 1976.

Vamos a volverlo a simular para un **paso de adaptación** de **0.01** pero con 9000 iteraciones para analizar mejor la convergencia. Los resultados son:



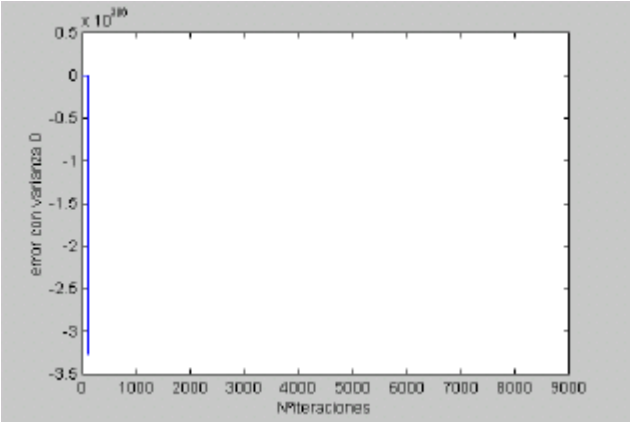



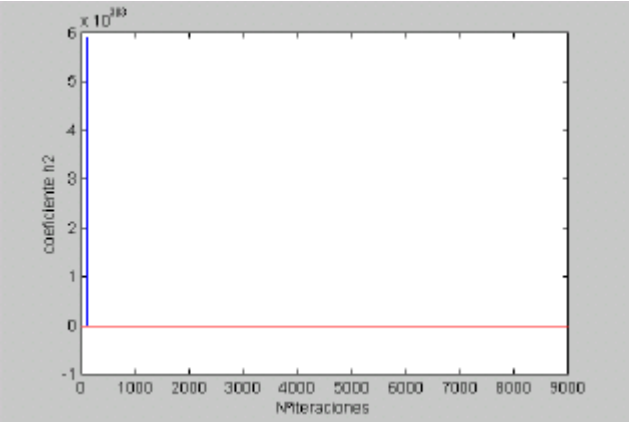



El algoritmo converge en la iteración 4549 (siguiendo con el mismo criterio de convergencia). Luego se ha obtenido que, al aumentar la media del proceso de entrada, la convergencia se hace más lenta para un mismo paso de adaptación. Esto es así porque las ecuaciones del algoritmo se deducen para un proceso de entrada de media cero. Además, para que converja, se necesita un paso de adaptación menor si se va subiendo la media.

Por último, vamos a simular ahora con ***un proceso de entrada de media 10***, que es un valor bastante mayor que los anteriores. Es de esperar que la convergencia sea bastante más lenta y que el paso de adaptación tenga que ser muy pequeño.

Se empieza con **m=0.01**. Vamos a trabajar con 9000 iteraciones para analizar mejor la convergencia. Se ha obtenido:

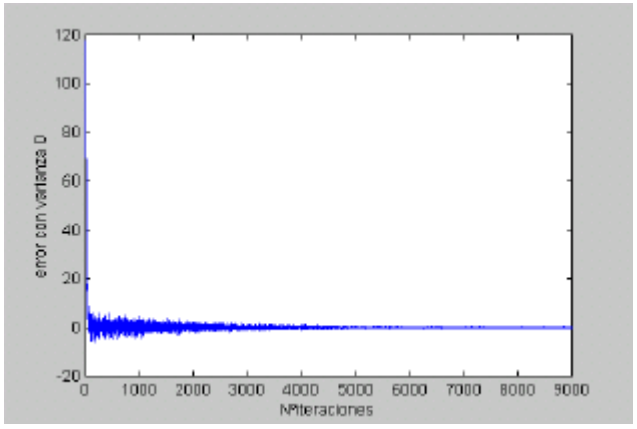


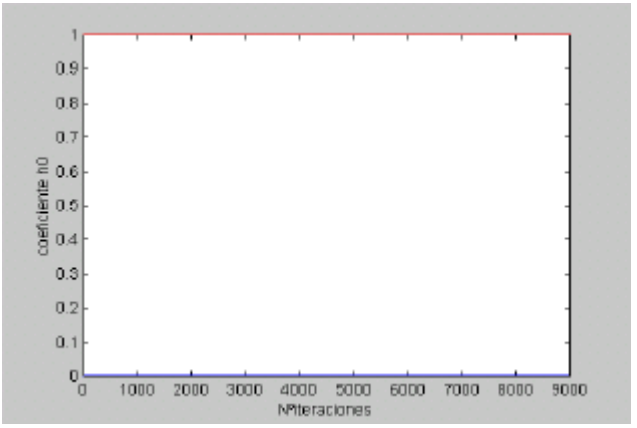



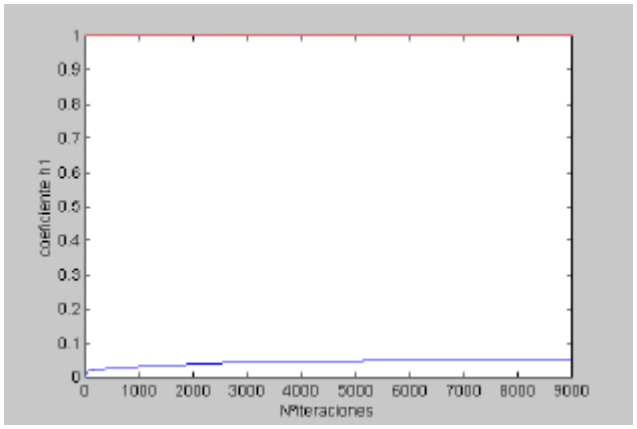

Sólo se ha representado el error y uno de los coeficientes ($h_2$) porque con la señal de error se ve claramente que el algoritmo diverge. La señal de error toma un valor demasiado grande para poderla representar.

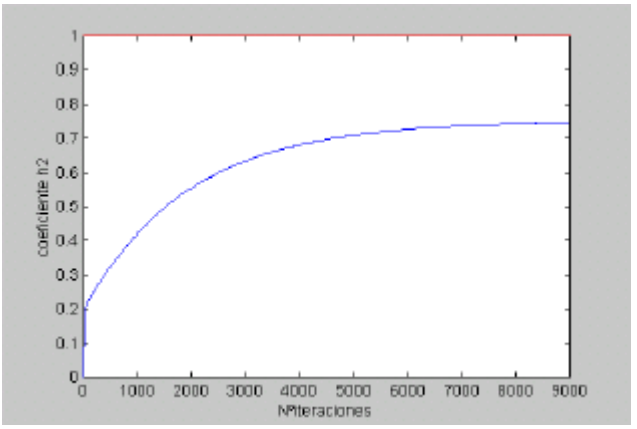
El resultado es relativamente lógico si se tiene en cuenta que la media del proceso de entrada es elevada y, en consecuencia, para que el algoritmo converja, el paso de adaptación debe ser muy pequeño.

Se ha vuelto a realizar la simulación pero con **m=0.00001**, es decir, tres órdenes de magnitud menor que antes. Se sigue trabajando con 9000 iteraciones para poder conseguir la convergencia. Ahora los resultados son:



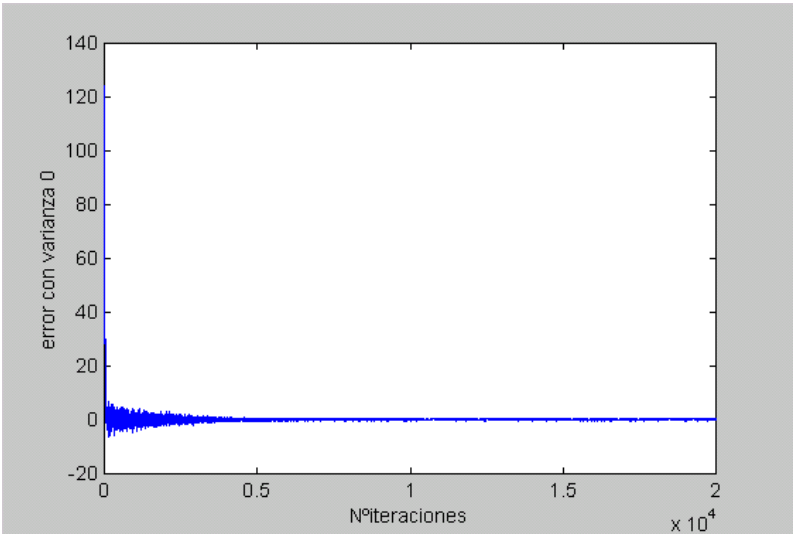



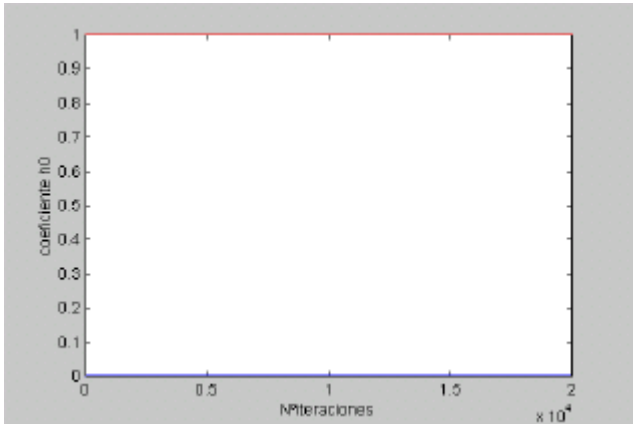



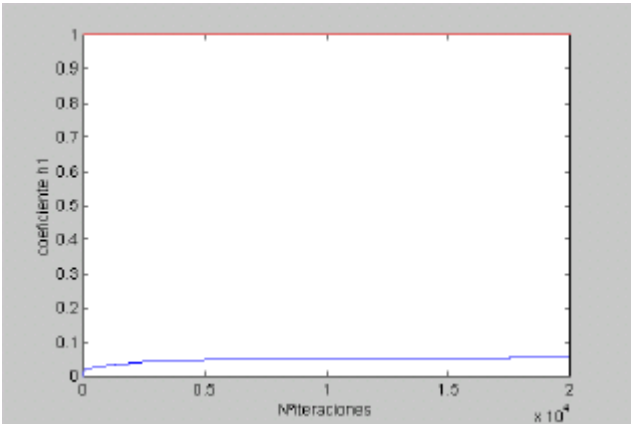

El algoritmo parece que termina convergiendo pero hacen falta más iteraciones. El coeficiente $h_2$ es el más influyente, lo cual es lógico porque es el que multiplica a la señal al cuadrado. Los coeficientes $h_0$ y $h_1$ están lejos de la convergencia mientras que $h_2$ se aproxima bastante.

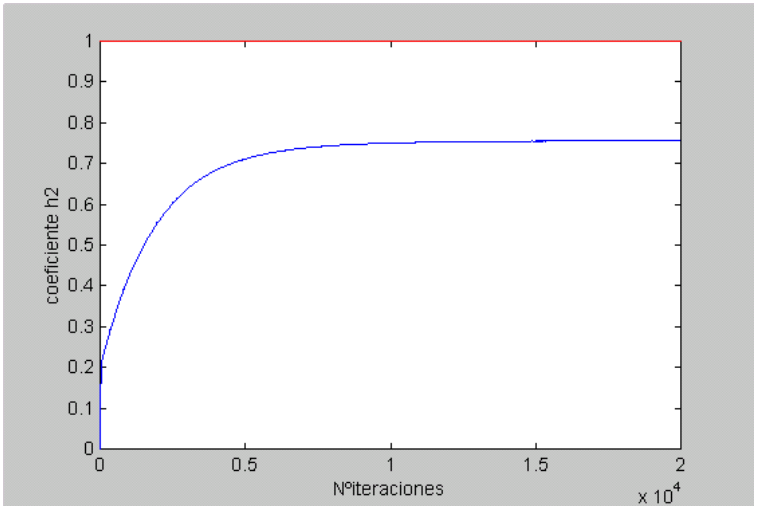
Con el mismo criterio de convergencia, no se puede decir que el algoritmo converja puesto que la señal de error nunca llega a ser menor que $\pm 0.03$.

Vamos a volverlo a simular pero con 20000 iteraciones para asegurarnos que el algoritmo converge y comprobar así cuantas iteraciones hacen falta exactamente. Los resultados ahora son:



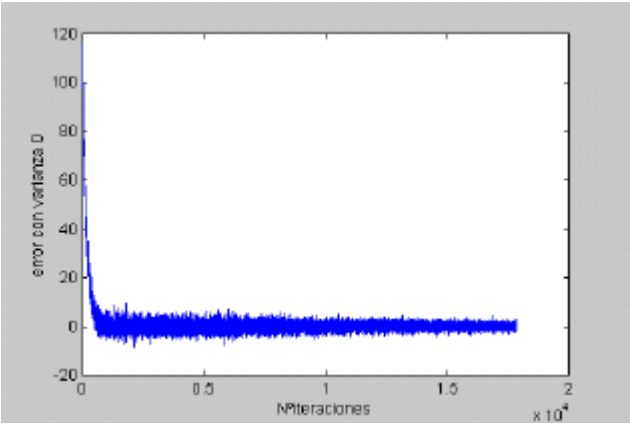



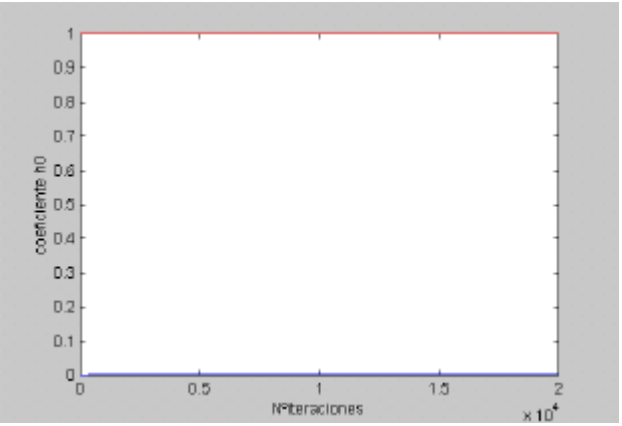



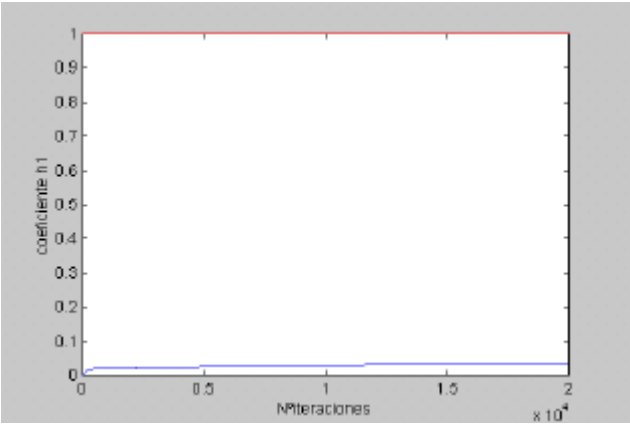

La convergencia es tan lenta que ni con 20000 iteraciones se consigue totalmente. Gracias al coeficiente cuadrático, la señal de error es bastante pequeña. Apenas hay

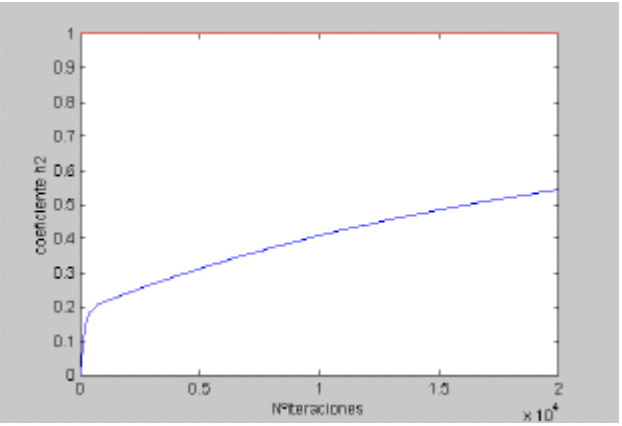
diferencia entre 9000 y 20000 iteraciones lo que quiere decir que harán falta muchísimas iteraciones más para conseguir la convergencia total.

Vamos a probar con un paso de adaptación de un orden menor (**$m=0.1*10^{-6}$**) para ver si así se consigue que converja el algoritmo. Los resultados son:




Se obtienen prácticamente los mismos resultados que antes, es decir, una convergencia muy lenta y que ni con 20000 iteraciones se llega al final.

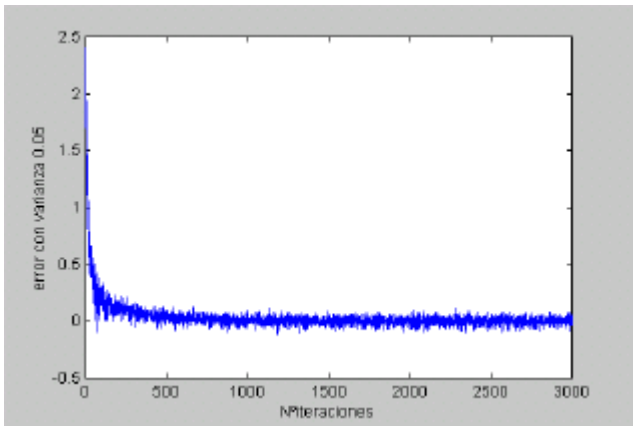
Por tanto, se puede concluir que el algoritmo está pensado para trabajar con procesos de entrada de media cero. Cuando la media va creciendo, la convergencia se hace más lenta y el paso de adaptación debe ser menor para conseguirla. El número de iteraciones necesarias para conseguir la convergencia del algoritmo es mayor conforme la media del proceso de entrada va creciendo. No es una relación proporcional, es decir, si la media del proceso de entrada

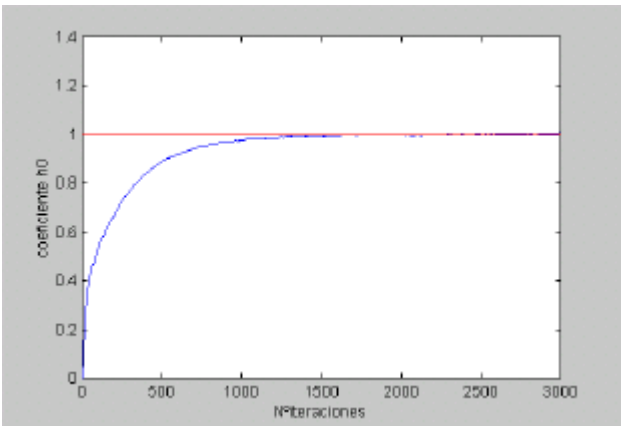
aumenta un poco, el número de iteraciones debe aumentar muchísimo para que el algoritmo converja.

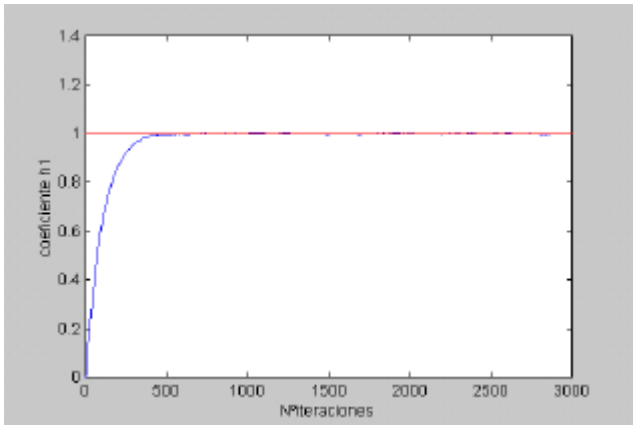
Vamos a analizar a continuación la situación cuando interviene el ruido a la hora de estimar la respuesta deseada. Se empezará primero con un ruido de varianza 0.05 (ruido pequeño) para tratar por último el caso en el que la varianza del ruido vale 1 (ruido grande). Es lógico suponer que la estimación empeorará a medida que aumente el ruido.

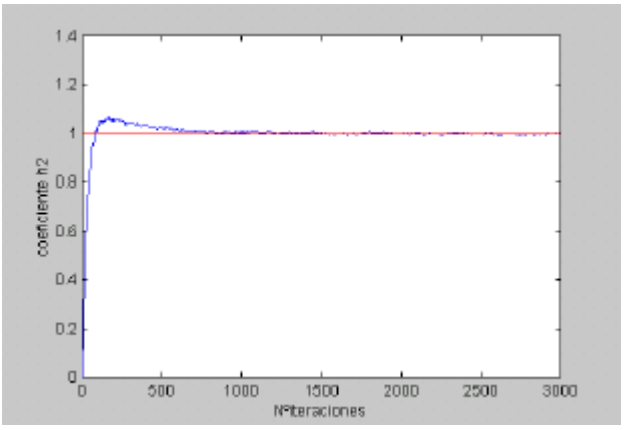
- $s_w^2 = 0.05$:

Se empieza con ***un proceso de entrada de media 0***. Con un **paso de adaptación** igual a **0.01**, los resultados son:




Los coeficientes convergen bastante bien. En la señal de error se aprecia la presencia del ruido en las pequeñas oscilaciones.

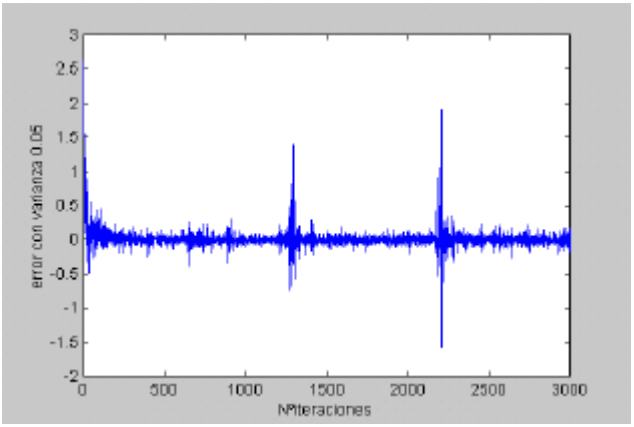
El número de operaciones que emplea Matlab es de 40230950. Mientras se mantenga este ruido y se trabaje con

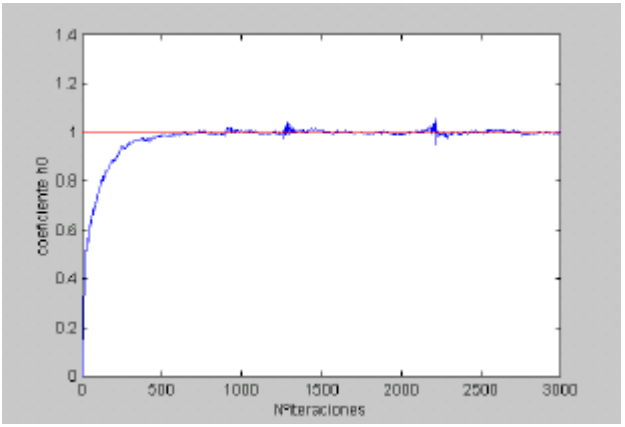
3000 iteraciones, este valor es el mismo. Evidentemente, al aumentar el número de iteraciones, crece el número de operaciones necesarias.

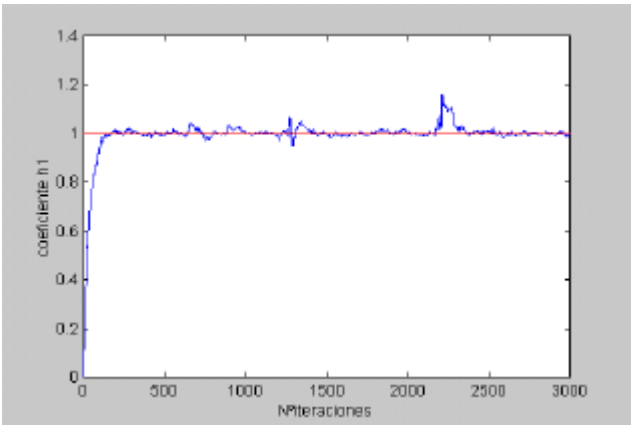
Debido a la presencia del ruido, es conveniente cambiar el criterio de convergencia relajando la condición que teníamos antes. Se han tomado dos criterios: el primero, que la señal de error sea menor que $\pm 0.1$ (criterio del 10%); y, el segundo, que sea menor que $\pm 0.15$ (criterio del 15%).

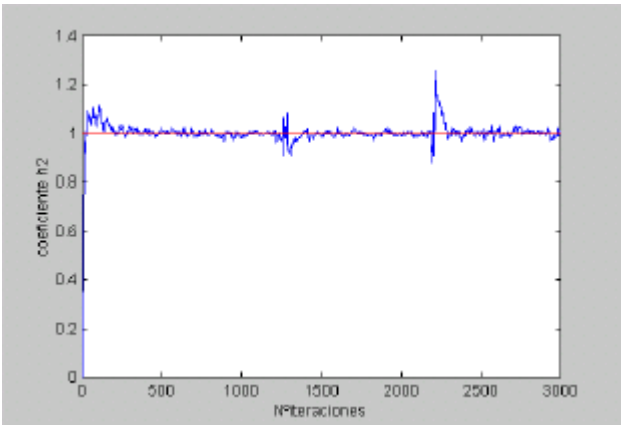
Para este primer valor del paso de adaptación, la convergencia se produce en la iteración 2347 según el criterio del 10% y en la 310 según el del 15%.

Aumentamos ahora el **paso de adaptación** a **0.03**. Los resultados son:

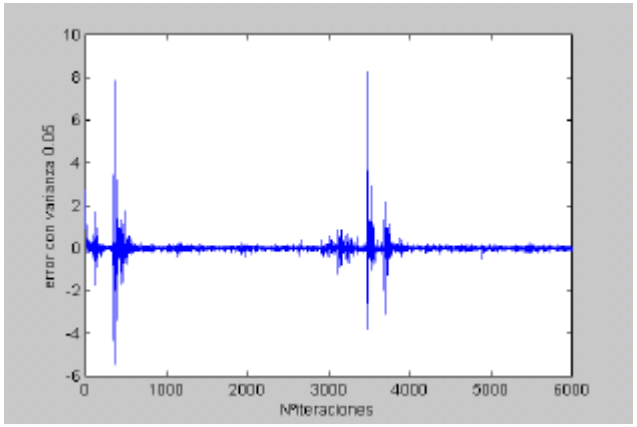


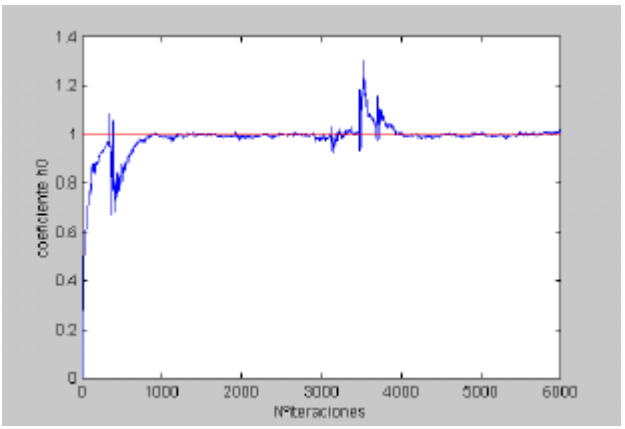



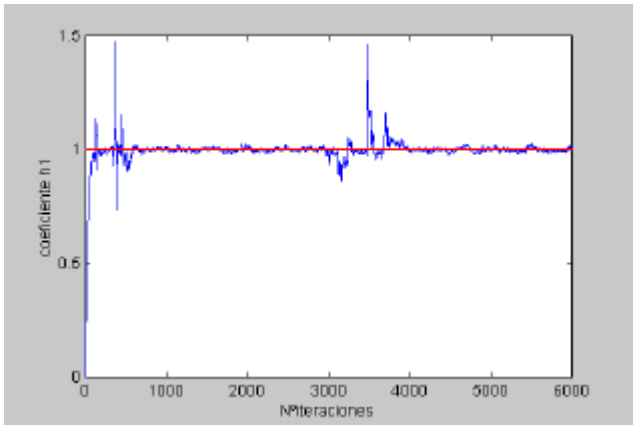



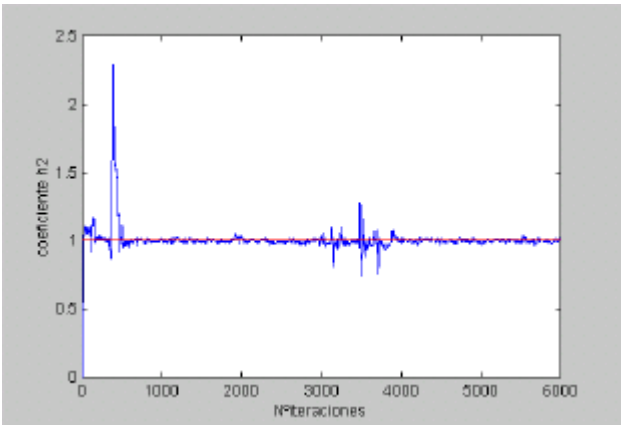




En este caso, la convergencia es peor aunque los coeficientes alcanzan su valor final. Sin embargo, con ninguno de los dos criterios que hemos establecido se alcanza la convergencia. Por ello se ha vuelto a simular considerando 6000 iteraciones. Los resultados son:



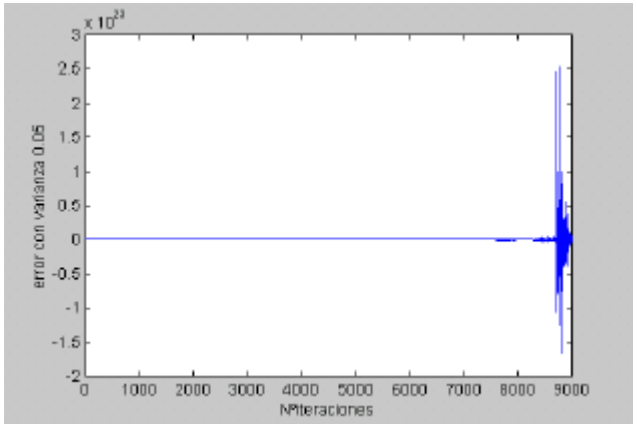



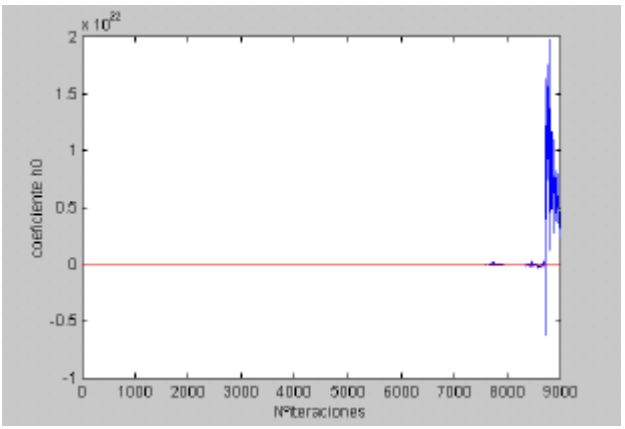



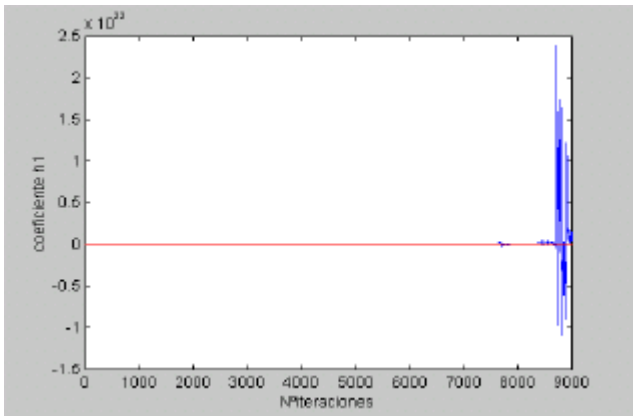



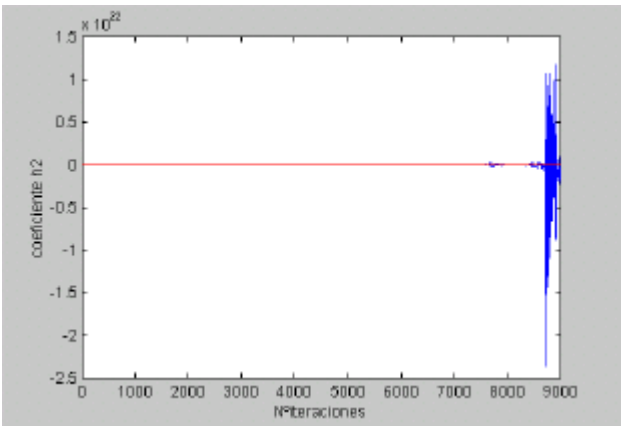

Ahora sí se consigue la convergencia con ambos criterios. De acuerdo con el del 10%, la convergencia se consigue en la iteración 5932; mientras que de acuerdo con el del 15%, ésta se obtiene en la 5834. Lógicamente, con el segundo criterio la convergencia se consigue con menos iteraciones puesto que la condición está más relajada. El número de operaciones ha crecido a 80463950 al aumentar a 6000 iteraciones.

Se ha realizado una última simulación para este caso con **m=0.06** y realizando 9000 iteraciones para estudiar mejor la convergencia. Se han obtenido estos resultados:

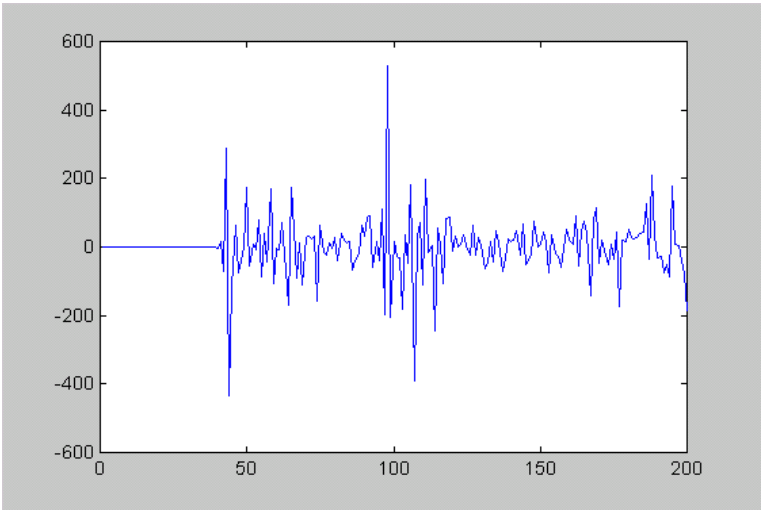


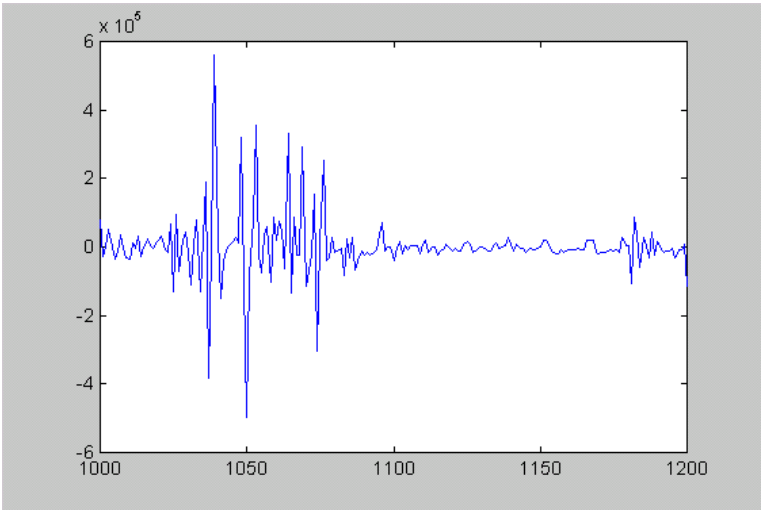




Este valor del paso de adaptación es demasiado grande y el algoritmo ya no converge. La señal de error va creciendo conforme aumentan las iteraciones aunque en la gráfica parezca que sólo crece al final. Lo que ocurre, como ya se comentó anteriormente, es que los valores finales son demasiado grandes y se pierde un poco la escala. Para comprobarlo, vamos a ver las primeras 200 muestras de la señal de error:

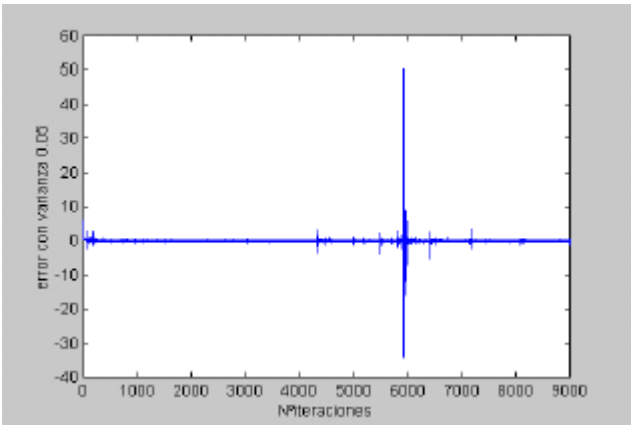


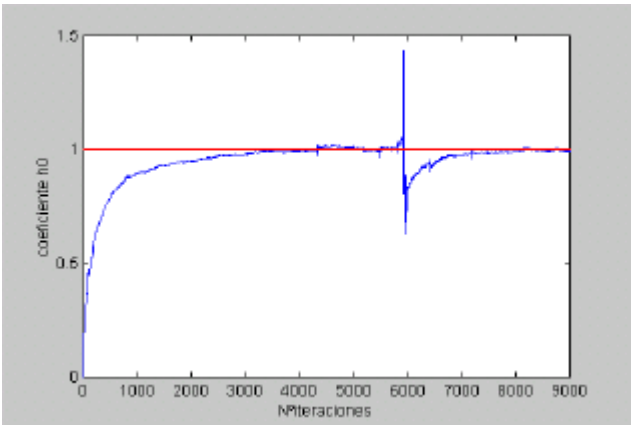
Para demostrar este crecimiento de la señal de error, la representamos también entre las muestras 1000 y 1200. Se observa que el valor del error es mayor que en las primeras 200 muestras anteriores:

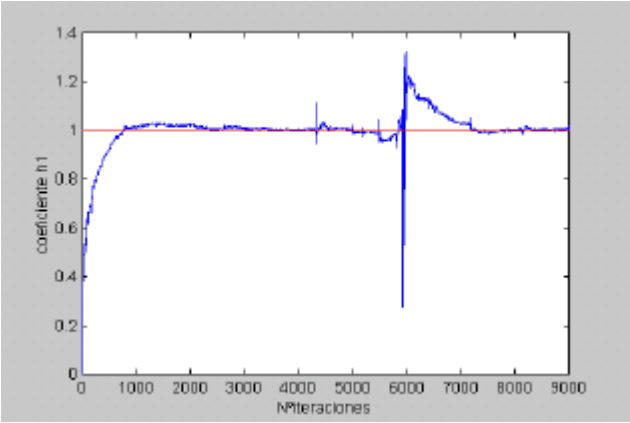


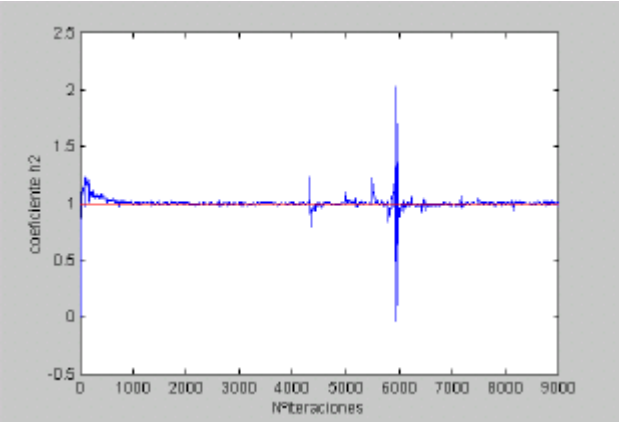

A continuación ***aumentamos la media del proceso de entrada de 0 a 1***. Se sigue con la misma varianza de ruido de 0.05. Tal y como se ha visto antes, la convergencia será más lenta y hará falta un paso de adaptación más chico.

Empezamos simulando con **m=0.01** y realizando 9000 iteraciones para poder estudiar mejor la convergencia. Los resultados han sido:

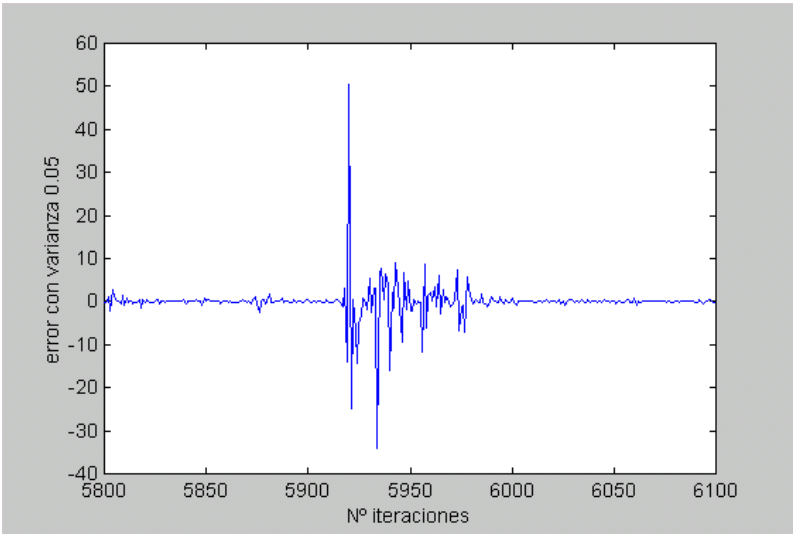




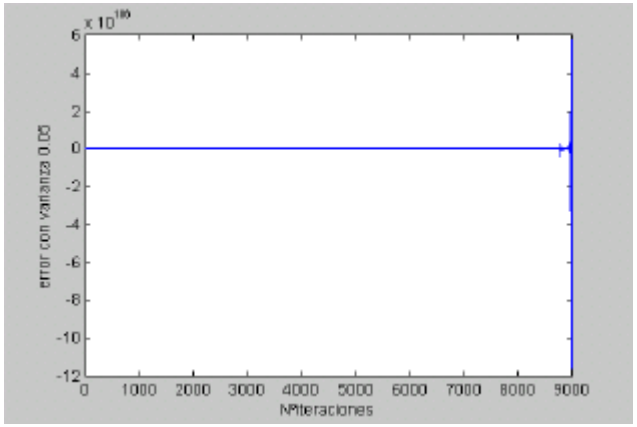



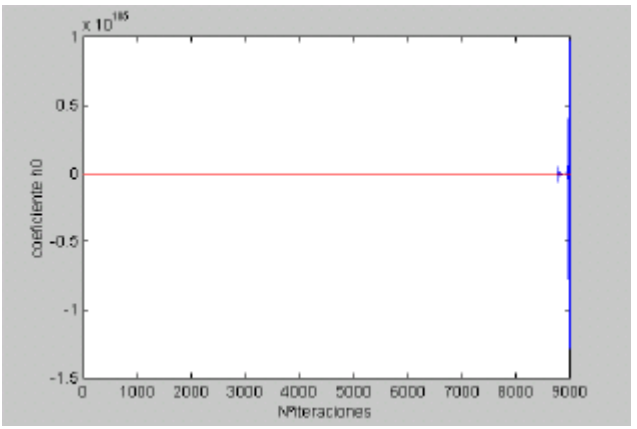

Más o menos el algoritmo converge aunque de acuerdo con nuestros criterios, esto se produce en la última iteración. Se aprecia un pico en la convergencia en torno a la iteración 6000. Vamos a analizarlo con más detalle mostrando las muestras alrededor de esta iteración:

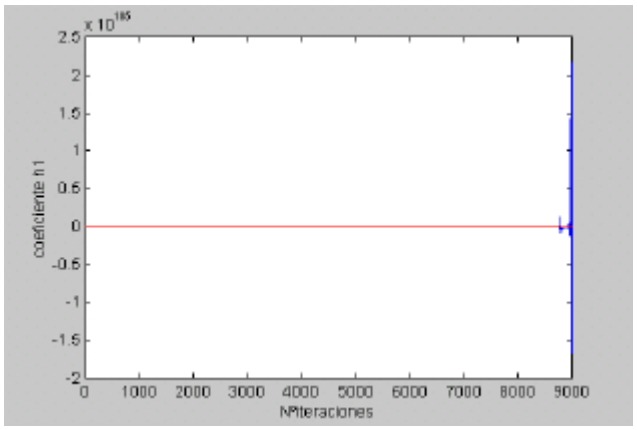


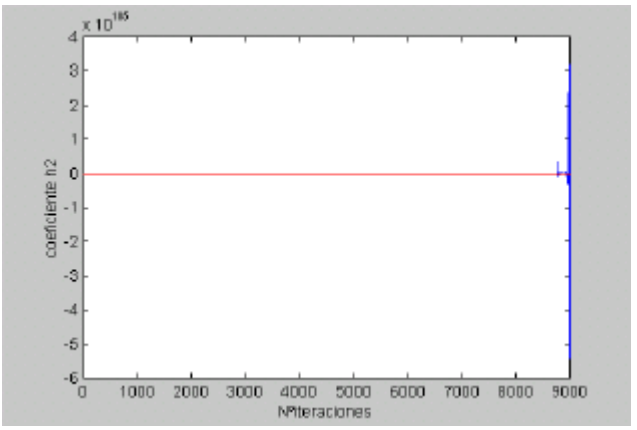

Esto se debe a un pico en la señal de entrada, que es un proceso aleatorio y como tal tiene variaciones. El algoritmo vuelve rápidamente a la convergencia.

A continuación, se simula nuevamente con un **paso de adaptación** de **0.03** manteniendo la varianza del ruido (0.05) y la media del proceso de entrada (1). Los resultados fueron:



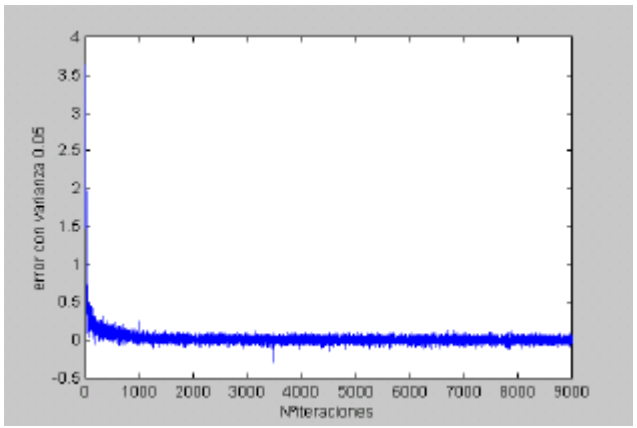



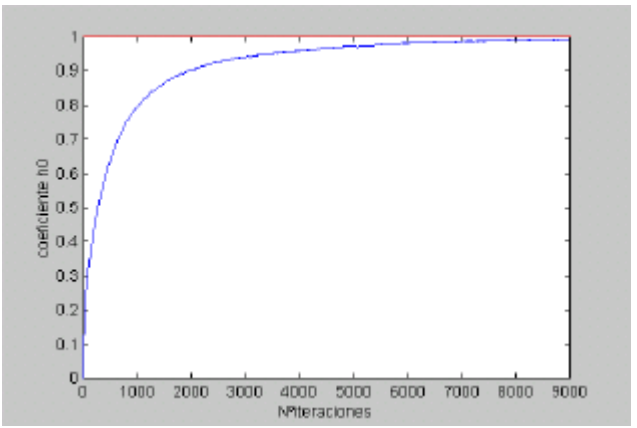



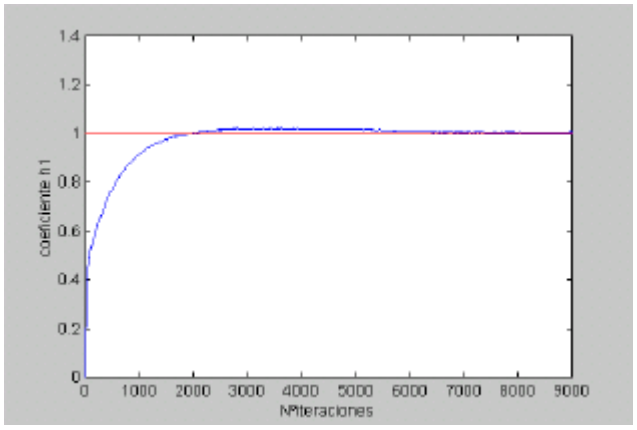



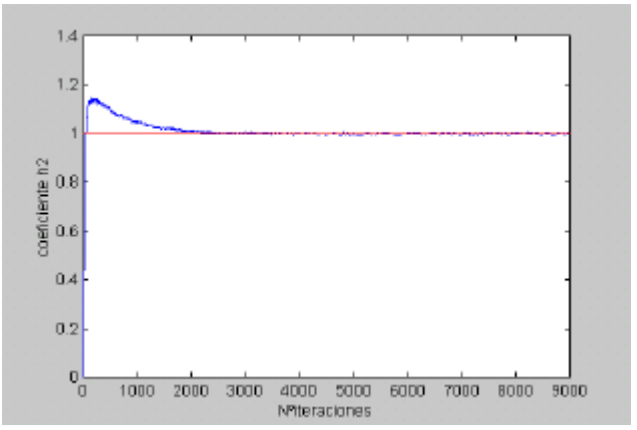

Como era de esperar, el algoritmo diverge puesto que al haber subido la media, el paso de adaptación debe ser pequeño. Como en ocasiones anteriores, la señal de error no crece sólo al final sino que es un crecimiento progresivo. Los últimos valores son tan grandes que desequilibran la gráfica.

Hacemos una última simulación en las mismas condiciones de ruido y con la misma entrada pero con un **paso de adaptación** de **0.005**. Se ha obtenido:



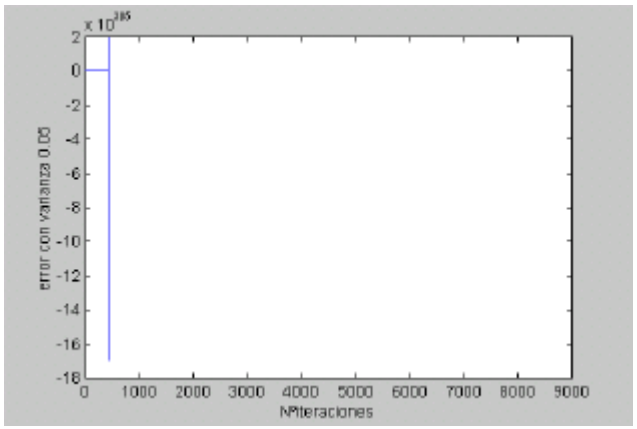


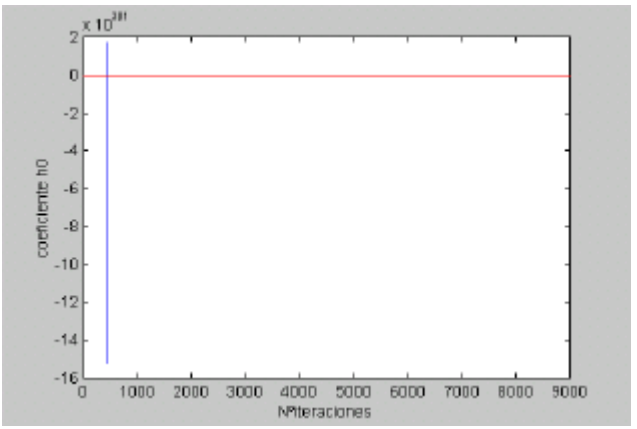



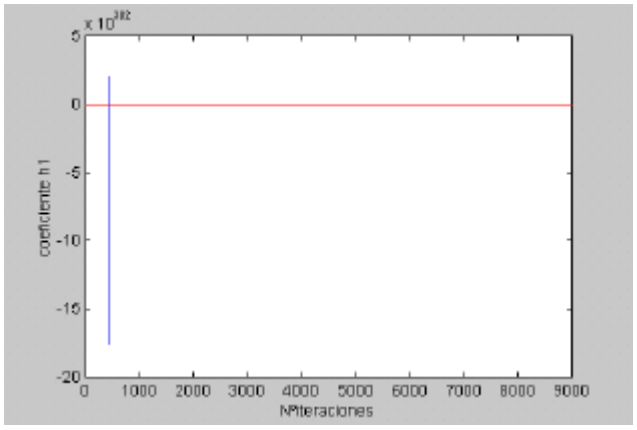

Ahora el algoritmo converge perfectamente. De acuerdo con el criterio del 10% converge en la iteración 8068 y según el del 15% en la 3463. Nuevamente se ve que al subir la media del proceso de entrada, el paso de adaptación debe bajar para que el algoritmo converja. La convergencia es más lenta que cuando la media del proceso de entrada es cero tal y como se ve al analizar la iteración en la que el algoritmo converge para cada caso.

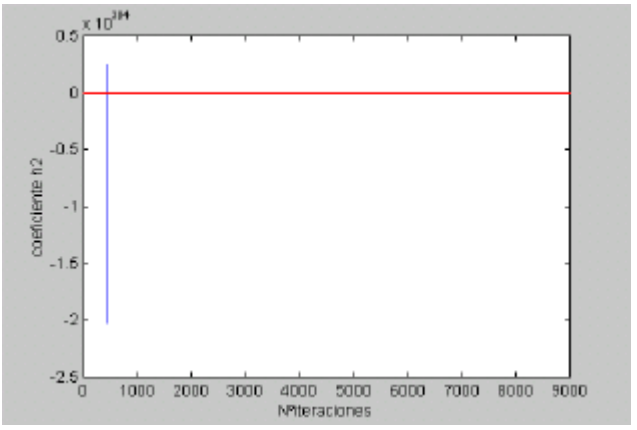
Finalmente, manteniendo la misma varianza de ruido (0.05), analizamos el caso en el que ***la media del proceso de entrada es 10***. Tal y como se ha visto antes, en este caso, el algoritmo converge muy lentamente y son necesarias demasiadas iteraciones. Además, el paso de adaptación debe ser muy pequeño.

Se empieza con **m=0.0001** y con 9000 iteraciones. Se ha obtenido:

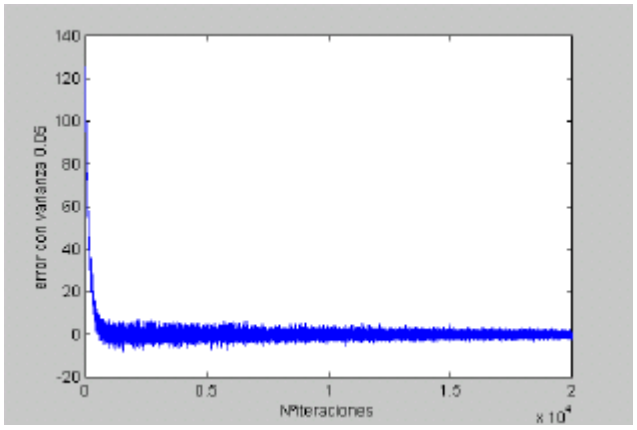


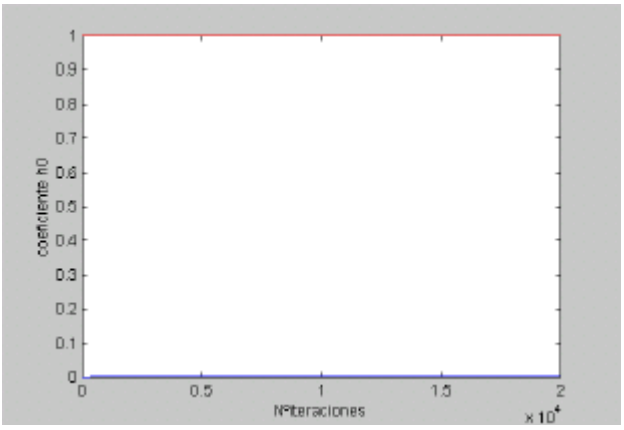



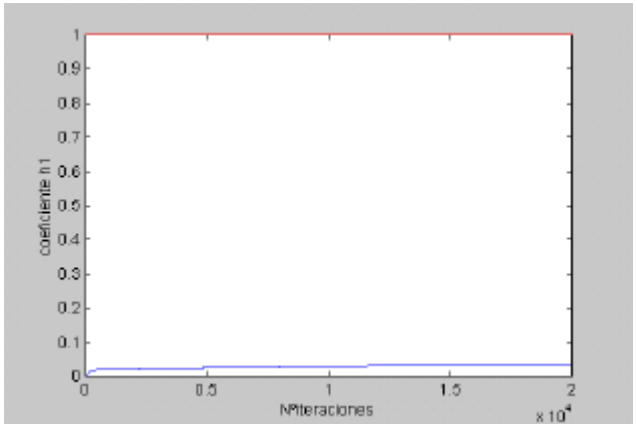


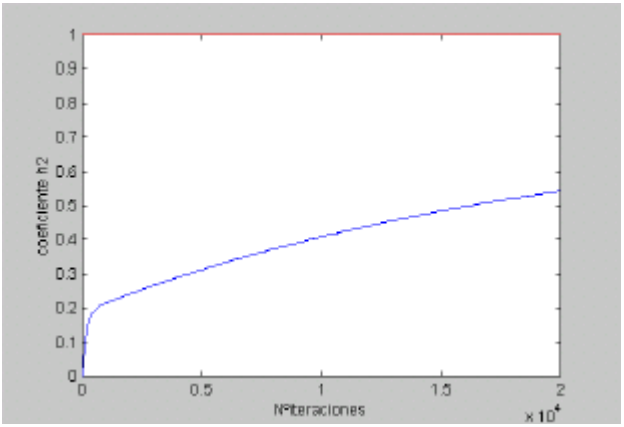




Como era de esperar, el algoritmo no converge con ese paso de adaptación. Vamos a probar con uno tres órdenes de magnitud más pequeño (**m=0.1*10$^{-6}$**) y con 20000 iteraciones para dar más tiempo al algoritmo a converger. Se obtiene:



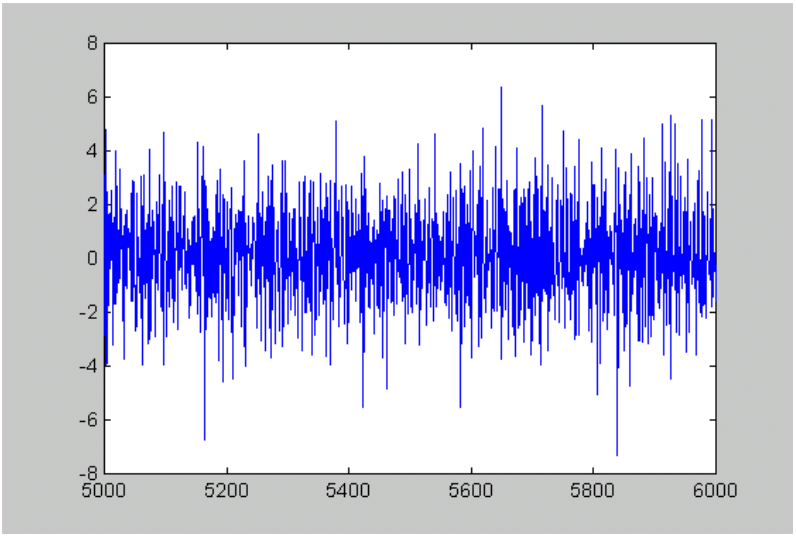



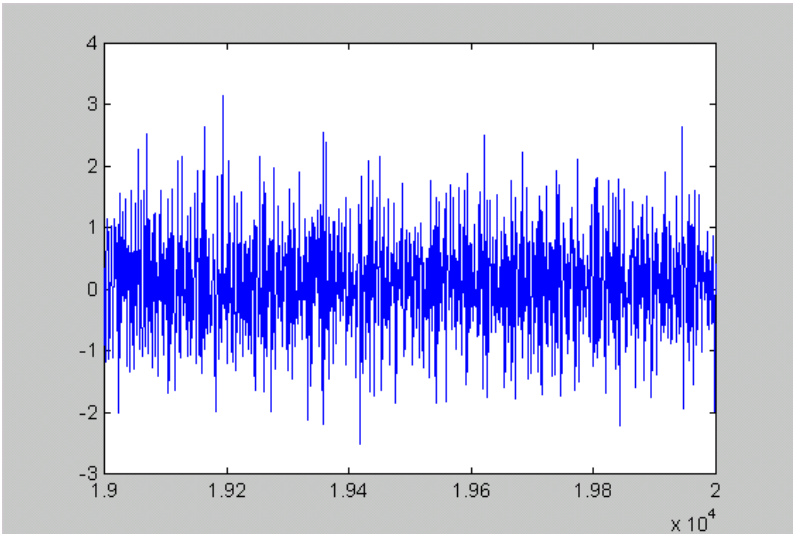



Con este paso de adaptación, el algoritmo no diverge pero con 20000 iteraciones no se consigue la convergencia completa.

Al representar todas las iteraciones no se aprecia bien pero la señal de error va bajando de valor conforme avanzan las iteraciones. Esta bajada es muy lenta. Para verlo mejor, se han representado por separado dos intervalos de dicha señal: uno, comprendido entre las iteraciones 5000 y 6000; y, otro con las 1000 últimas iteraciones:

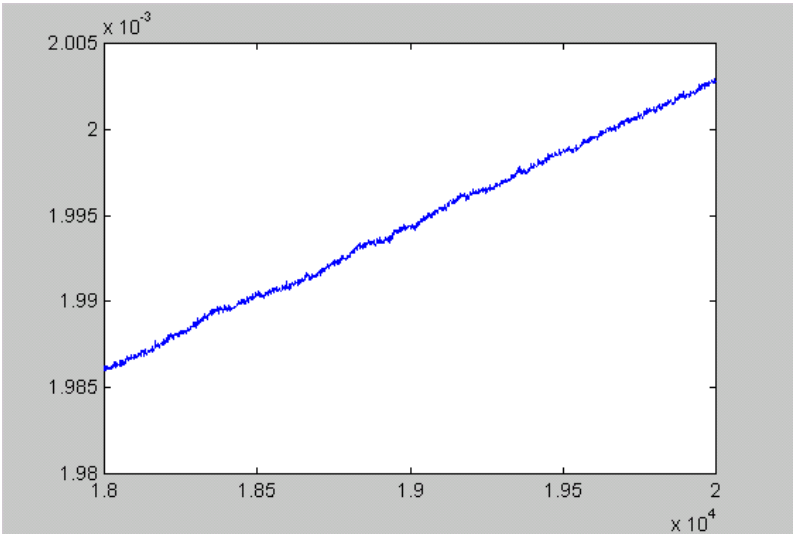



Se aprecia como los valores van bajando muy lentamente lo que demuestra la lentitud de la convergencia al considerar un proceso de entrada de media distinta de cero.

Por otro lado, el coeficiente $h_0$ parece que se mantiene siempre constante a cero. Sin embargo, esto no es cierto tal

y como se va a ver a continuación. Si se representan solamente las 2000 últimas muestras de dicho coeficiente, se tiene:

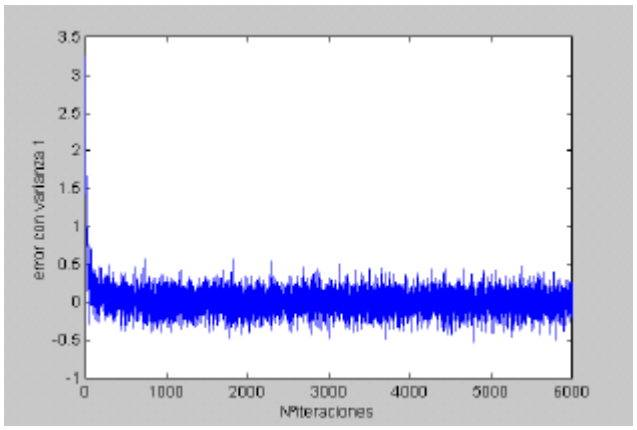


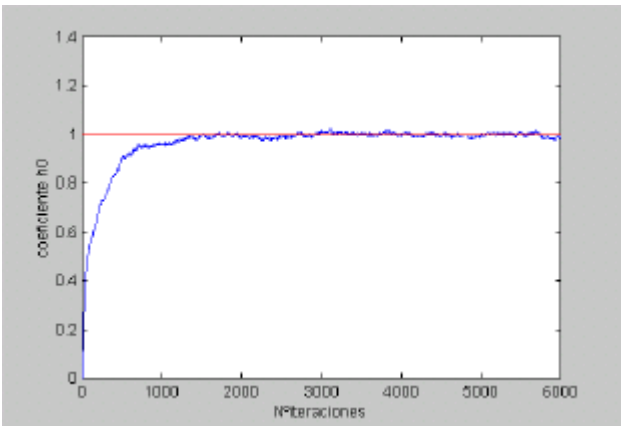

que demuestra claramente como el coeficiente va creciendo. Lo que sí queda también muy claro es que el crecimiento es extremadamente lento. Igual ocurre con los otros dos coeficientes.

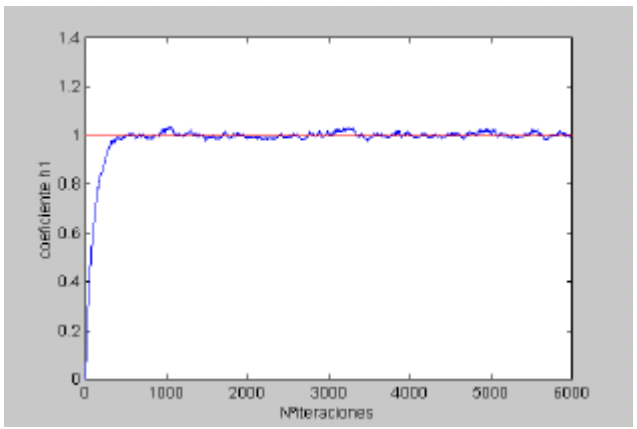
- $\underline{s_w^2 = 1}$:

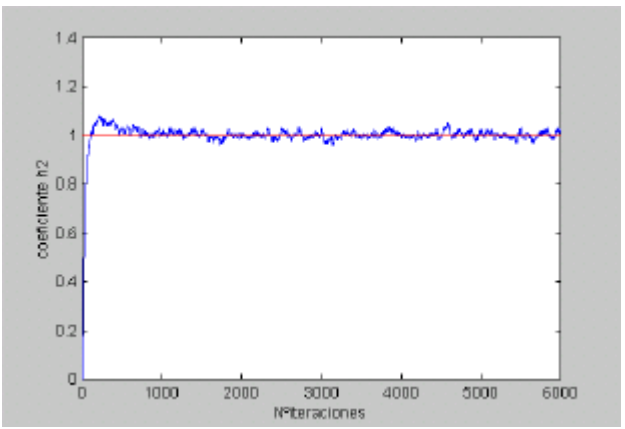
Ahora vamos a aumentar la varianza del ruido de 0.05 a 1. Debido a la presencia de este ruido, la adaptación será peor. Hay que cambiar nuevamente el criterio de convergencia. Se han vuelto a adoptar dos criterios: en el primero, se considera que el algoritmo converge cuando la señal de error se mantenga menor que $\pm 0.5$ (criterio del 50%); y en el segundo, cuando se mantenga menor que $\pm 0.55$ (criterio del 55%).

Comenzaremos con ***un proceso de entrada de media cero***. El primer valor del **paso de adaptación** será **0.01**, que es con el que se ha estado trabajando siempre. Se empieza considerando 6000 iteraciones. Los resultados son:



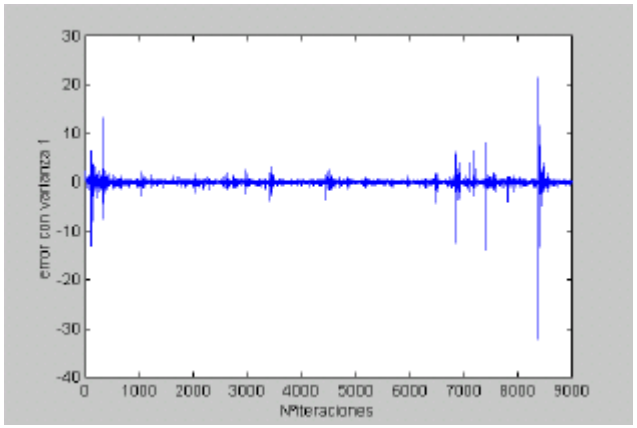



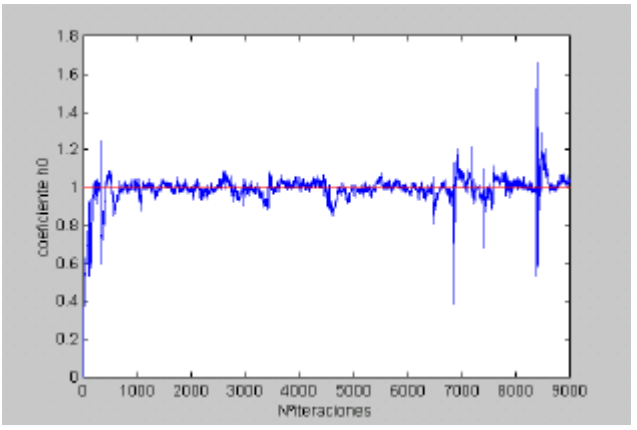

Se ve la presencia del ruido en las oscilaciones de la señal de error y en las pequeñas variaciones en torno al valor final en los tres coeficientes. El ruido siempre estará presente y por eso la convergencia nunca será totalmente perfecta. En este caso, es bastante buena. Según el criterio

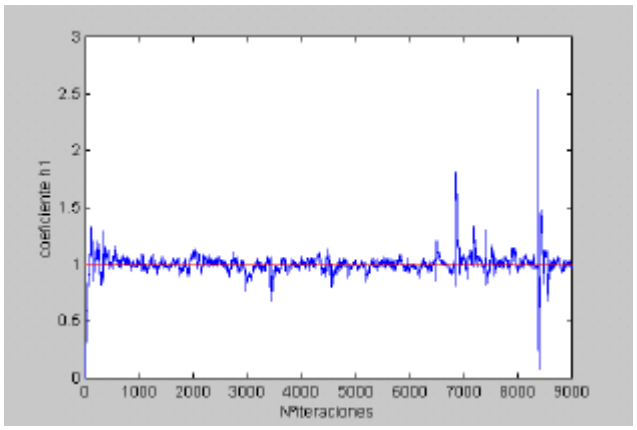
del 50%, ésta se produce en la iteración 5796, mientras que según el del 55%, en la iteración 1836.

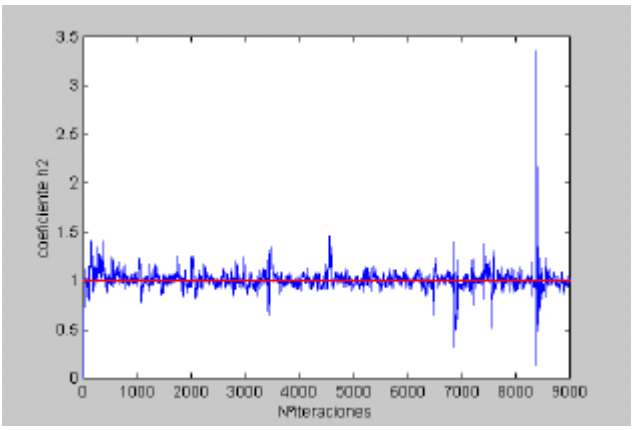
El número de operaciones necesarias es de 80463950, como siempre que se trabaje con este algoritmo y considerando 6000 iteraciones.

Vamos a aumentar el **paso de adaptación** a **0.03** para poder comparar con la situación de ruido pequeño con el mismo valor en los parámetros. Ahora se realizarán 9000 iteraciones puesto que tal y como se está viendo a lo largo de todas las simulaciones, la convergencia empeora al aumentar el paso de adaptación. Este deterioro en la convergencia se ve todavía más afectado debido a la presencia de ruido y/o de un proceso de entrada con media distinta de cero. En este caso, sólo tenemos la primera situación (ruido). Los resultados son:



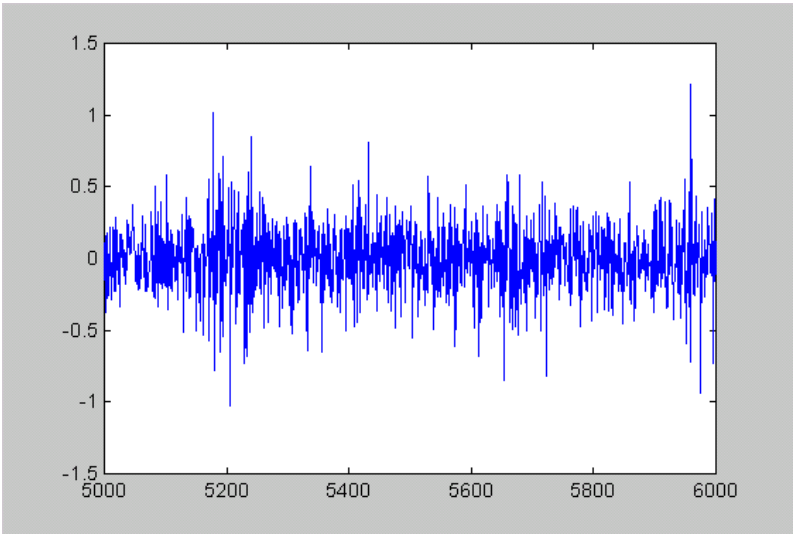




La convergencia se produce en la iteración 8981 según el criterio del 50% y en la 8974 según el del 55%.

Aunque mirando las gráficas parezca que no existen oscilaciones como en las situaciones anteriores, esto no es cierto. Lo que ocurre es que ahora los valores son mucho mayores que antes y entonces las oscilaciones parecen más

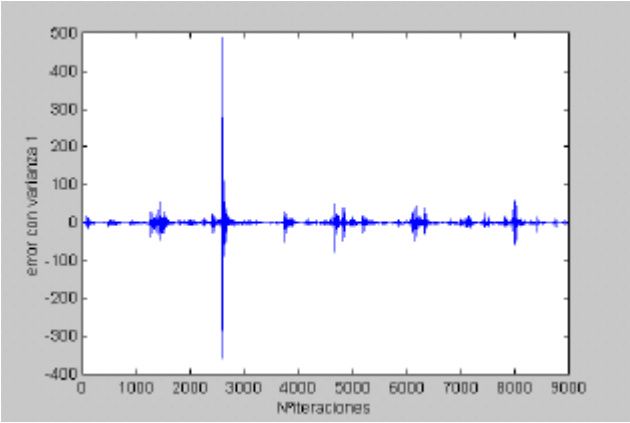
pequeñas. Para comprobarlo, se va a mostrar la señal de error en un pequeño intervalo (entre las muestras 5000 y 6000):

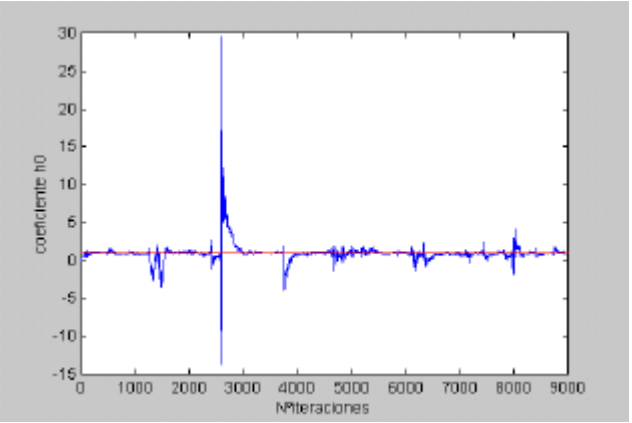


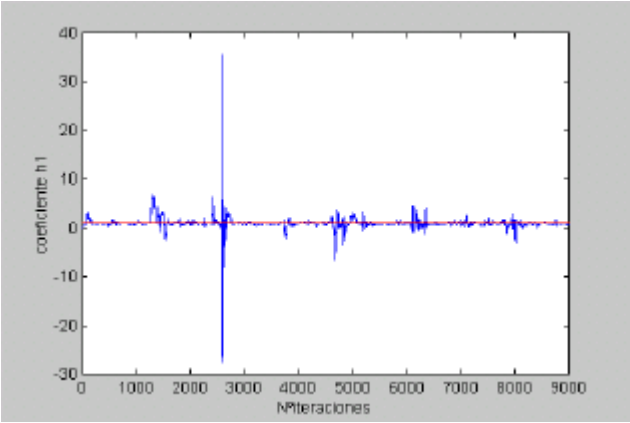

Aquí se aprecian claramente las oscilaciones debidas a la presencia del ruido.

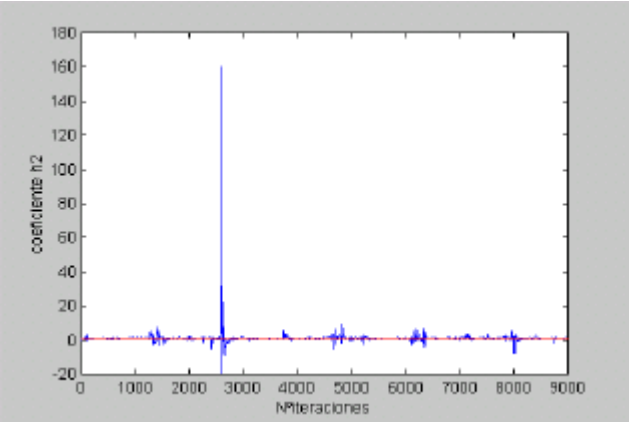
Al haber aumentado el paso de adaptación, la convergencia es más lenta.

Por último, se considera el caso en el que **m=0.04**. Las gráficas que se obtuvieron fueron:



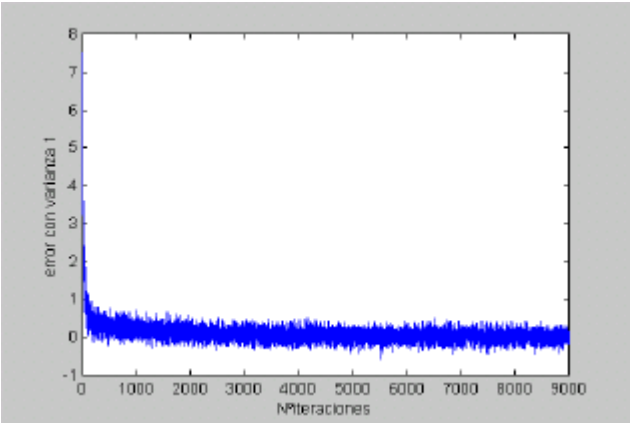


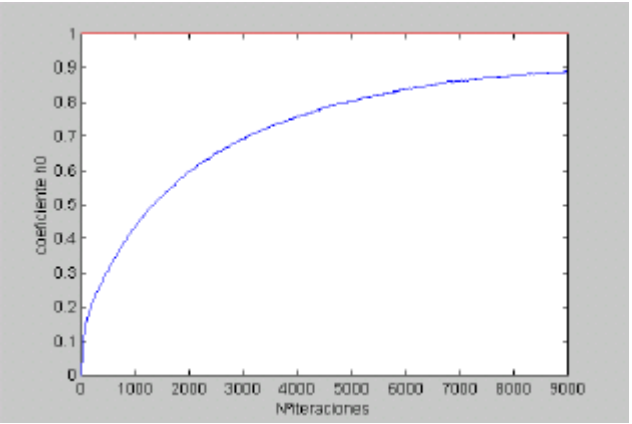



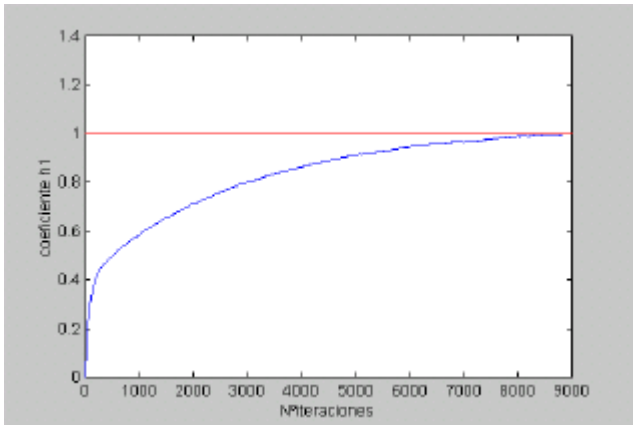

La convergencia se produce en la iteración 8992 según ambos criterios. Al igual que antes, al haber aumentado los valores, las oscilaciones parecen desaparecer pero como se acaba de ver no es cierto.

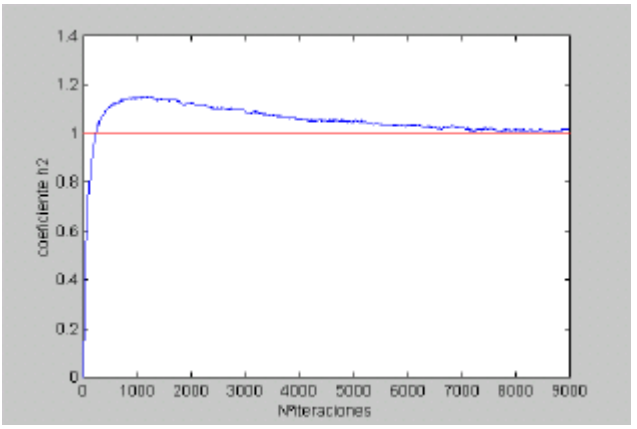
El valor tan elevado que se produce en torno a la iteración 2600 se debe a que el paso de adaptación empieza a ser demasiado grande y el algoritmo empieza a tener problemas de convergencia que consigue subsanar en las siguientes iteraciones.

Vamos a analizar ahora que ocurre cuando ***la media del proceso de entrada es 1***. Como ya se ha visto, en este caso la convergencia se ralentiza y hace falta un **paso de adaptación** más pequeño. Por eso, se ha empezado con **m=0.001**. Los resultados son:



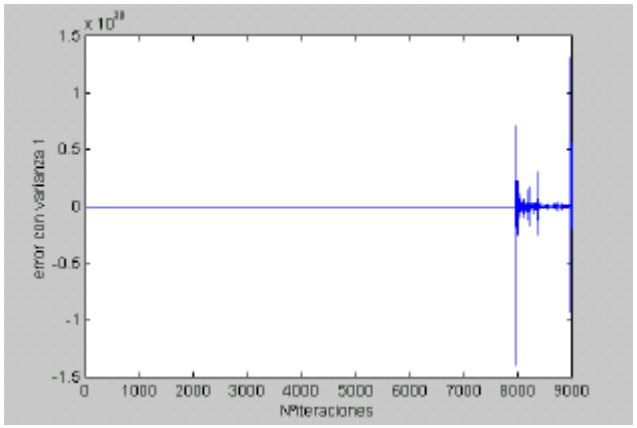



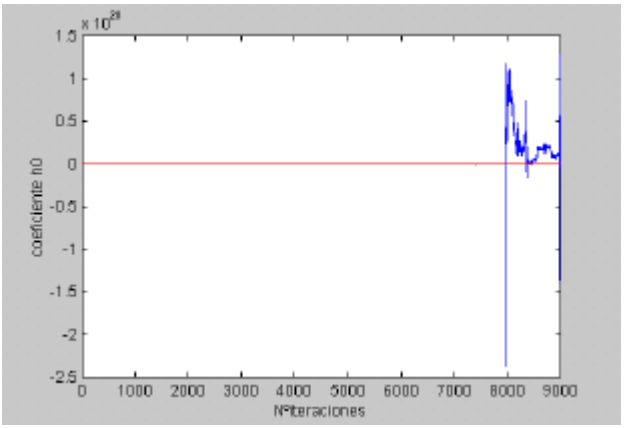


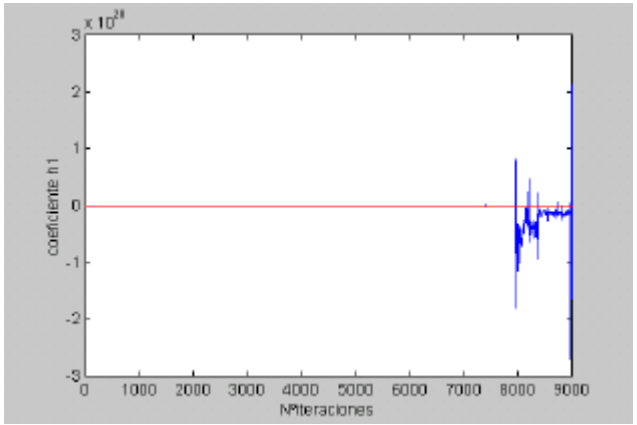



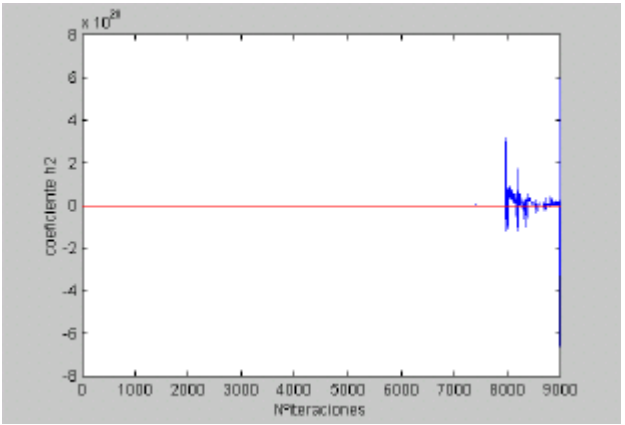

La convergencia se consigue en la iteración 6524 según el criterio del 50% y en la 5518 según el del 55%.

Vamos a subir el paso de adaptación para comprobar que el algoritmo diverge antes al tener un proceso de entrada de media 1. Con **m=0.02**, se ha obtenido:

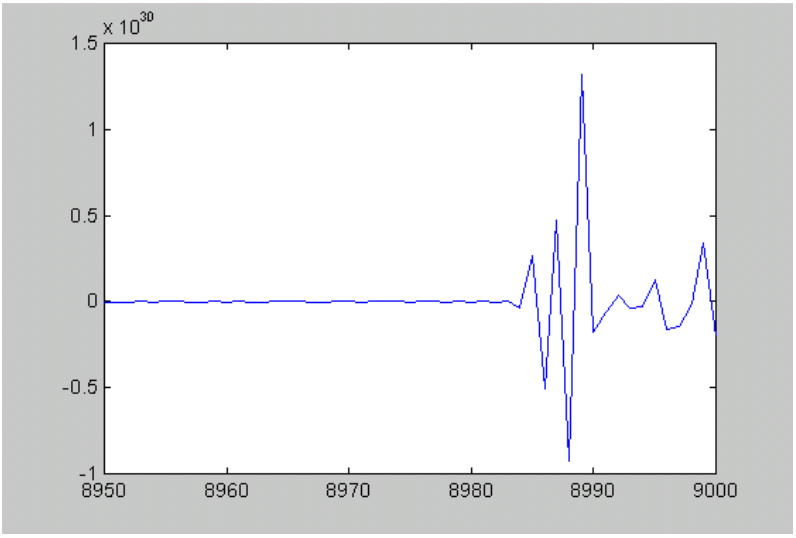


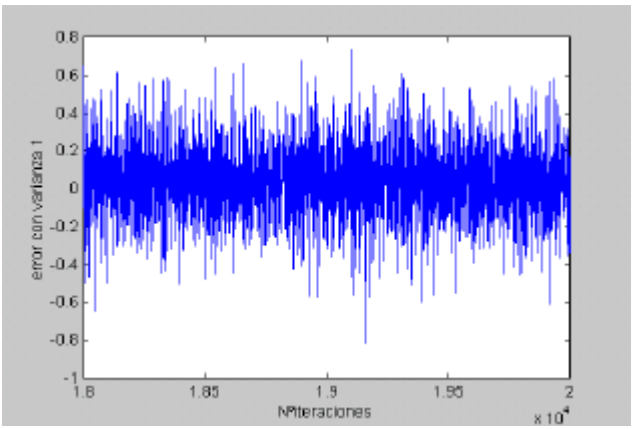



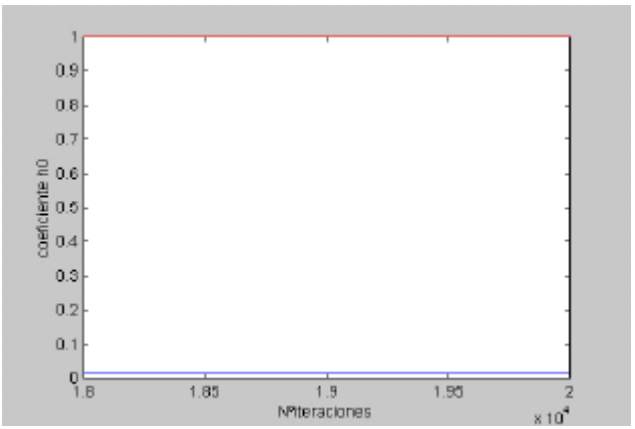



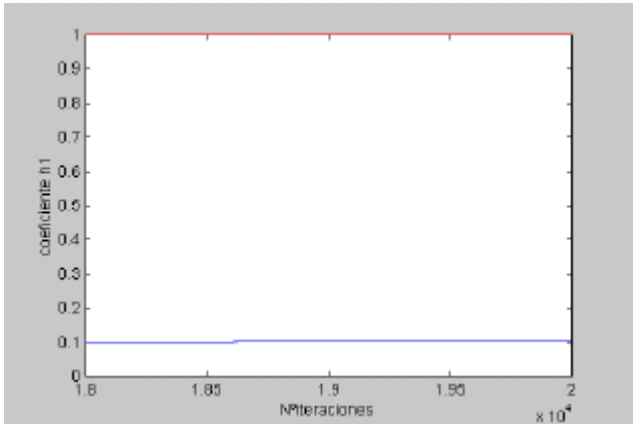
Como era de esperar, el algoritmo diverge. Para verlo mejor representamos sólo las últimas 50 muestras de la señal de error. Estos valores son tan grandes que parece que la señal vale siempre cero lo cual como se ha visto antes no es cierto. Así:

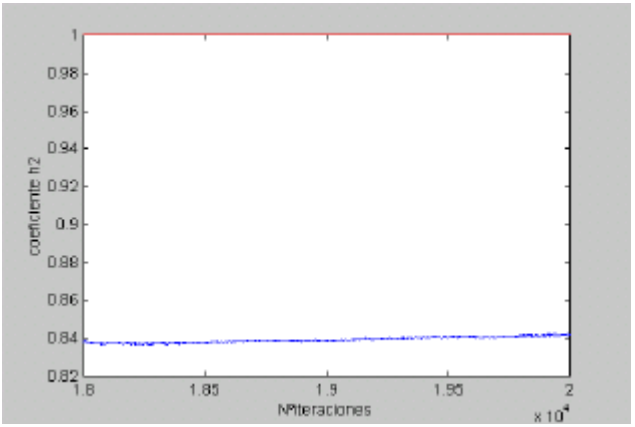



Por último, vamos a hacer una simulación con esta varianza de ruido, 1, y con ***una media del proceso de entrada igual a 10***. Como ya se ha analizado en los ejemplos anteriores, la convergencia es lenta y requiere un **paso de adaptación** pequeño. Se ha considerado un valor igual a $\mathbf{0.1*10^{-4}}$ y se han simulado 20000 iteraciones. Para facilitar la visión de las gráficas, sólo se representan las 2000 últimas iteraciones. Los resultados fueron:






La convergencia se alcanza en la iteración 19930 para ambos criterios del 50% y 55%. Los coeficientes $h_0$ y $h_1$ se adaptan muy mal como ya es habitual en estas situaciones. Sin embargo, la buena adaptación de $h_2$ hace posible que se alcance una convergencia aceptable aunque no del todo buena.

Se ha podido observar una gran dependencia de la señal de entrada en este algoritmo. Cuando la media del proceso de entrada era distinta de cero, la convergencia era lenta y en algunas situaciones el algoritmo divergía.

La presencia del ruido también ha sido un factor importante a la hora de reducir la velocidad de convergencia.

El paso de adaptación debe ser más pequeño cuanto mayor es el ruido o la media del proceso de entrada. Esto provoca que la convergencia sea muy lenta en estas situaciones.

Vamos a expresar todos los resultados obtenidos con este algoritmo en una tabla. La columna convergencia hace referencia a la iteración en la que ésta se obtuvo de acuerdo con el criterio indicado en la columna correspondiente. La columna media indica la media del proceso de entrada que se utiliza en cada caso. Las otras dos columnas hacen referencia a la varianza del ruido y al paso de adaptación. Así:

| Nºiteraciones | $s_w^2$ | media | m | criterio | convergencia |
|---|---|---|---|---|---|
| 3000 | 0 | 0 | **0.01** | 3% | **689** |
| 3000 | 0 | 0 | 0.03 | 3% | 259 |
| 3000 | 0 | 0 | 0.05 | 3% | 2069 |
| 15000 | 0 | 0 | 0.08 | 3% | X |
| 3000 | 0 | 1 | **0.005** | 3% | **1976** |
| 9000 | 0 | 1 | 0.01 | 3% | 4549 |
| 3000 | 0 | 1 | 0.02 | 3% | X |
| 9000 | 0 | 10 | 0.01 | 3% | X |
| 20000 | 0 | 10 | $0.1*10^{-5}$ | 3% | X |
| 20000 | 0 | 10 | $0.1*10^{-6}$ | 3% | X |
| 3000 | 0.05 | 0 | **0.01** | 10%/15% | **2347/310** |
| 6000 | 0.05 | 0 | 0.03 | 10%/15% | 5932/5834 |
| 9000 | 0.05 | 0 | 0.06 | 10%/15% | X |
| 9000 | 0.05 | 1 | 0.01 | 10%/15% | X/8999 |
| 9000 | 0.05 | 1 | 0.03 | 10%/15% | X/X |
| 9000 | 0.05 | 1 | **0.005** | 10%/15% | **8068/3463** |
| 9000 | 0.05 | 10 | $0.1*10^{-3}$ | 10%/15% | X/X |
| 20000 | 0.05 | 10 | $0.1*10^{-6}$ | 10%/15% | X/X |
| 6000 | 1 | 0 | **0.01** | 50%/55% | **5796/1836** |
| 9000 | 1 | 0 | 0.03 | 50%/55% | 8981/8974 |
| 9000 | 1 | 0 | 0.04 | 50%/55% | 8992/8992 |
| 9000 | 1 | 1 | **0.001** | 50%/55% | **6524/5518** |
| 9000 | 1 | 1 | 0.02 | 50%/55% | X/X |
| 20000 | 1 | 10 | $0.1*10^{-4}$ | 50%/55% | 19930/19930 |

Resultados con el algoritmo LMS

- **LMS normalizado:**

A continuación, vamos a analizar este otro algoritmo. Como ya se dijo al principio de este capítulo, es análogo al LMS con la diferencia de que las ecuaciones de adaptación se normalizan respecto de la señal de entrada. Es decir, sería así:

$$H[n+1] = H[n] + \frac{m\ e[n]X[n]}{\|X[n]\|}$$

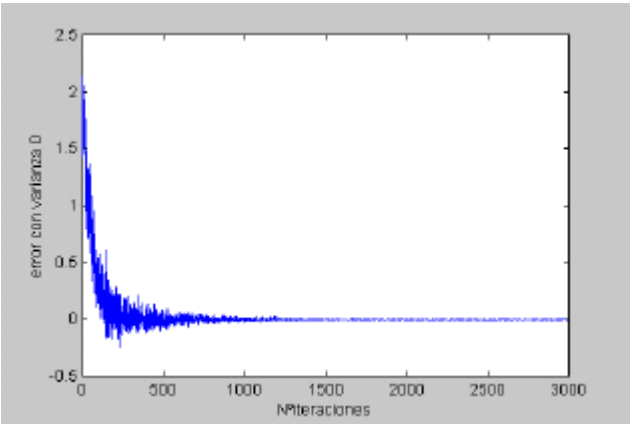
donde $\|X[n]\|$ representa la potencia de la señal de entrada.

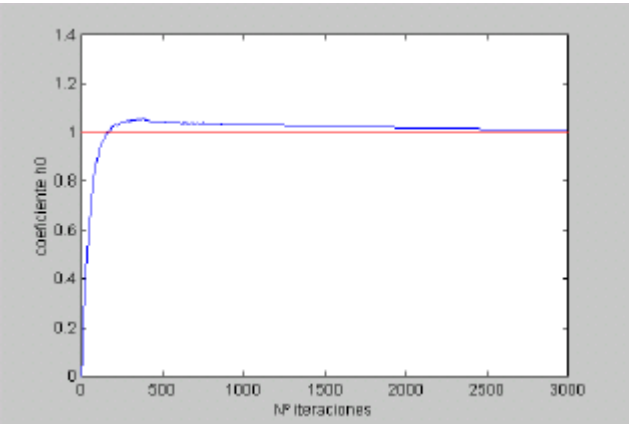
Se analizan los mismos casos que con el algoritmo LMS, es decir, se estudia la presencia o no del ruido, la introducción de un proceso de entrada con media cero y distinta de cero y se analizan diferentes valores del paso de adaptación.

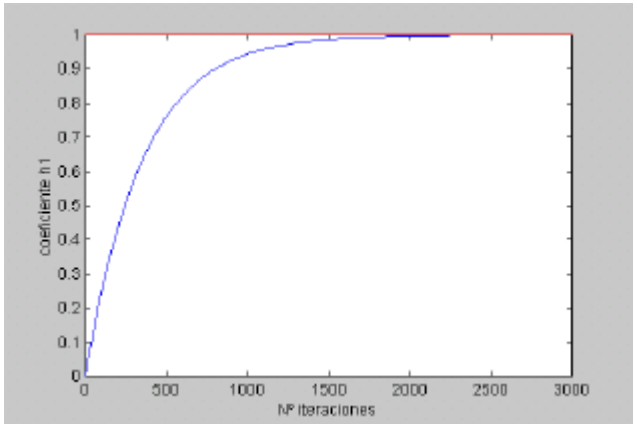
- $s_w^2 = 0$:

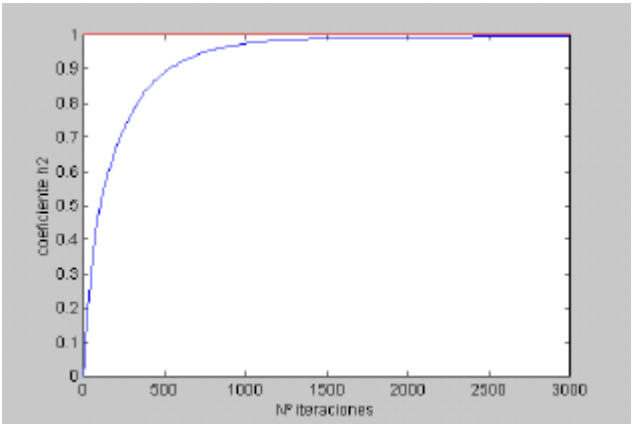
Empezaremos por el caso más sencillo, es decir, ausencia de ruido y ***la media del proceso de entrada igual a cero.***

Con el fin de poderlo comparar con el LMS, se intentará tomar los mismos valores para el paso de adaptación. De esta forma, se comienza simulando para **m=0.01**. Los resultados son:





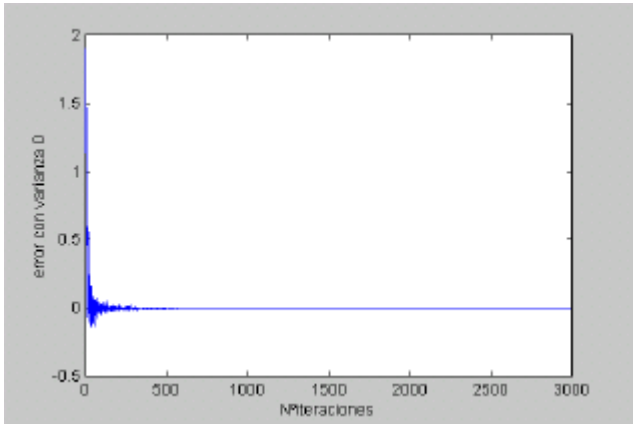



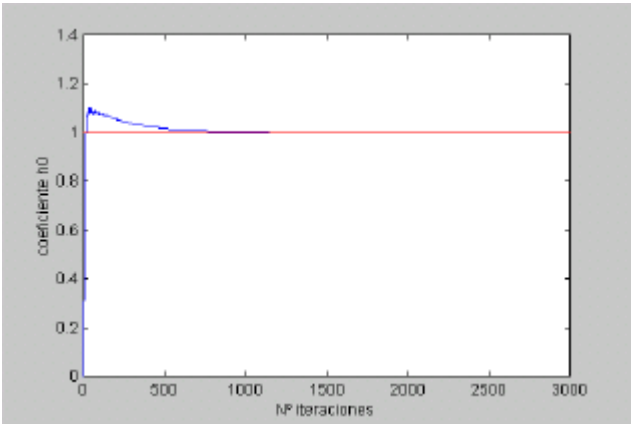

La convergencia es bastante buena. Al igual que en el caso LMS, se utiliza un criterio de convergencia consistente en que la señal de error se mantenga por debajo de un determinado valor. En este caso (ausencia de ruido), éste valor es $\pm 0.03$ (criterio del 3%). De acuerdo con esto, la convergencia se produce en la iteración 869. Este valor es un poco mayor que el que se obtuvo en el mismo caso para el algoritmo LMS, es decir, la convergencia es un poco más lenta.

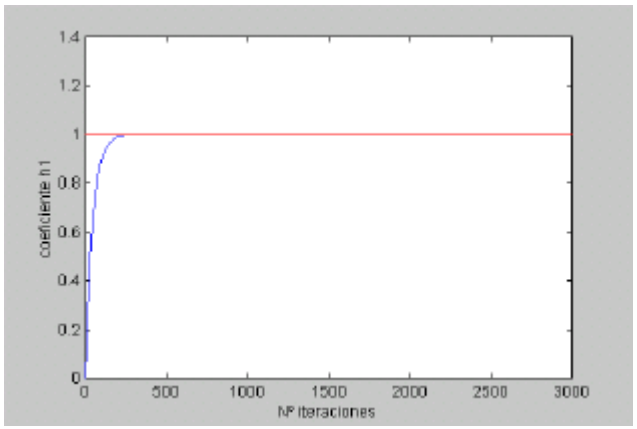
El número de operaciones realizadas es de 46980950, que es mayor que la que se necesita para el LMS.

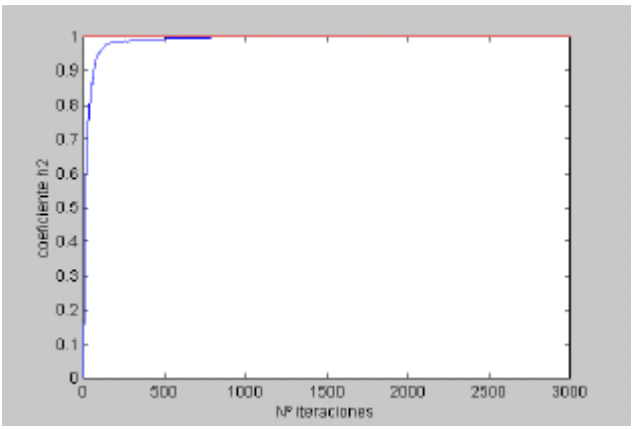
Por tanto, de momento, este algoritmo es peor que el LMS.

Pero si ahora se simula con **m=0.08**, los resultados son:



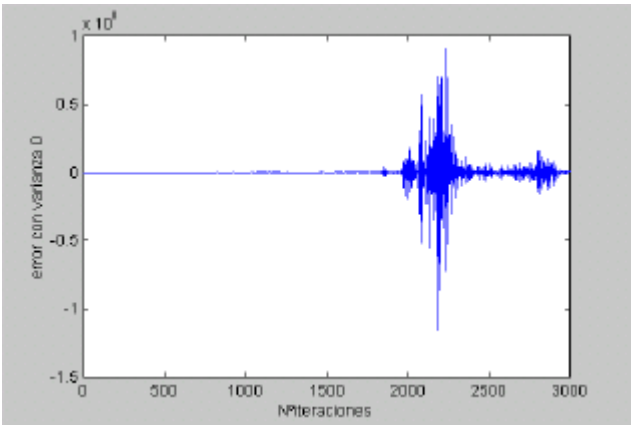


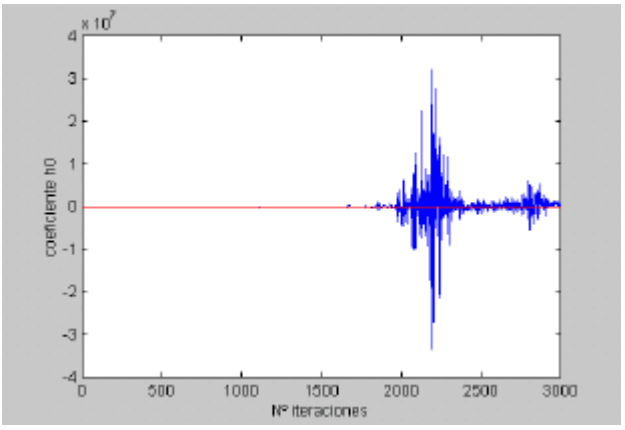



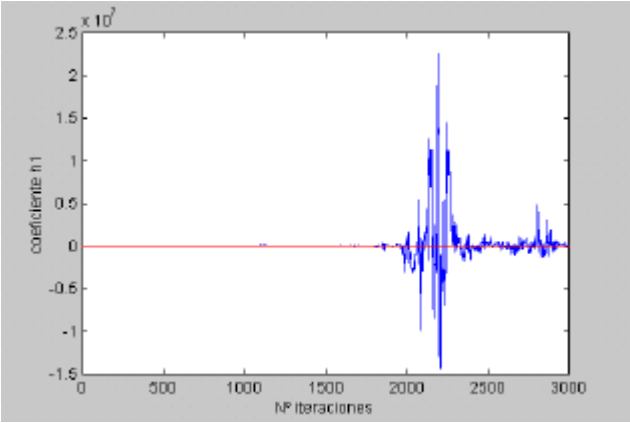

Se ve que la convergencia es bastante buena. De hecho, según el criterio del 3%, ésta se produce en la iteración 131. Esto supone una convergencia rapidísima.

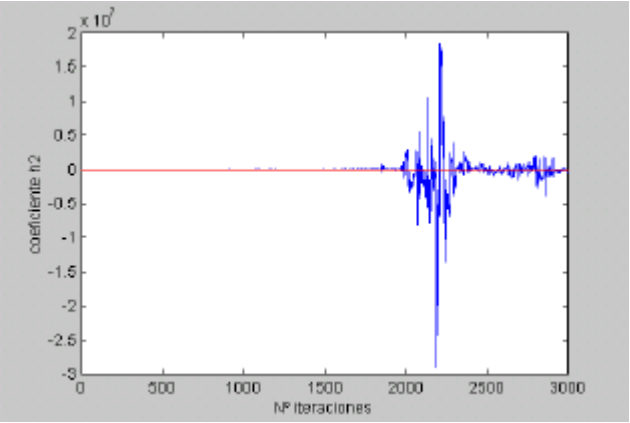
Comparando con el LMS, la mejora es considerable. En el caso LMS, el algoritmo no convergía con estos mismos parámetros. Sin embargo con el LMS normalizado no sólo no diverge, sino que la convergencia es rapidísima. Por tanto podemos decir que el algoritmo LMS normalizado con valores adecuados, converge más rápido que el LMS. La desventaja es que el LMS normalizado requiere unos 47 millones de operaciones en punto flotante mientras que el LMS necesita unos 40 millones de operaciones. Es decir, con el LMS normalizado se consigue una mayor velocidad de convergencia que con el LMS a costa de una mayor complejidad computacional.

Finalmente se simula con **m=0.4** y los resultados son:



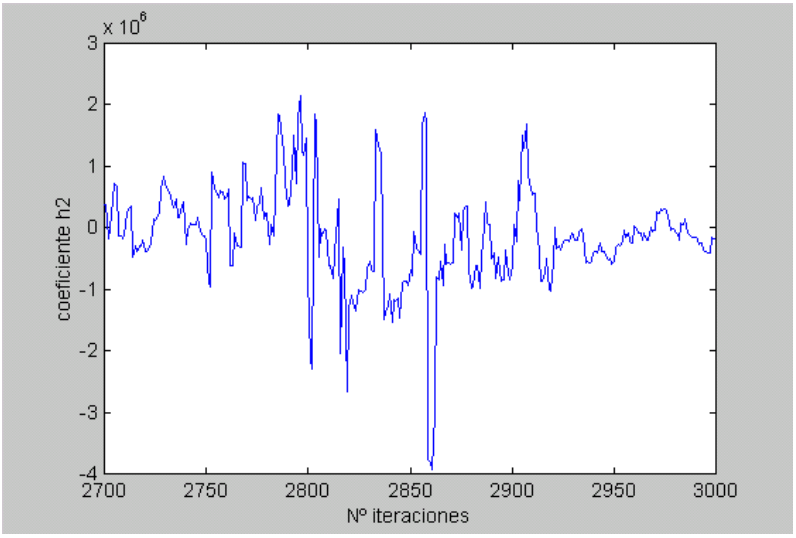





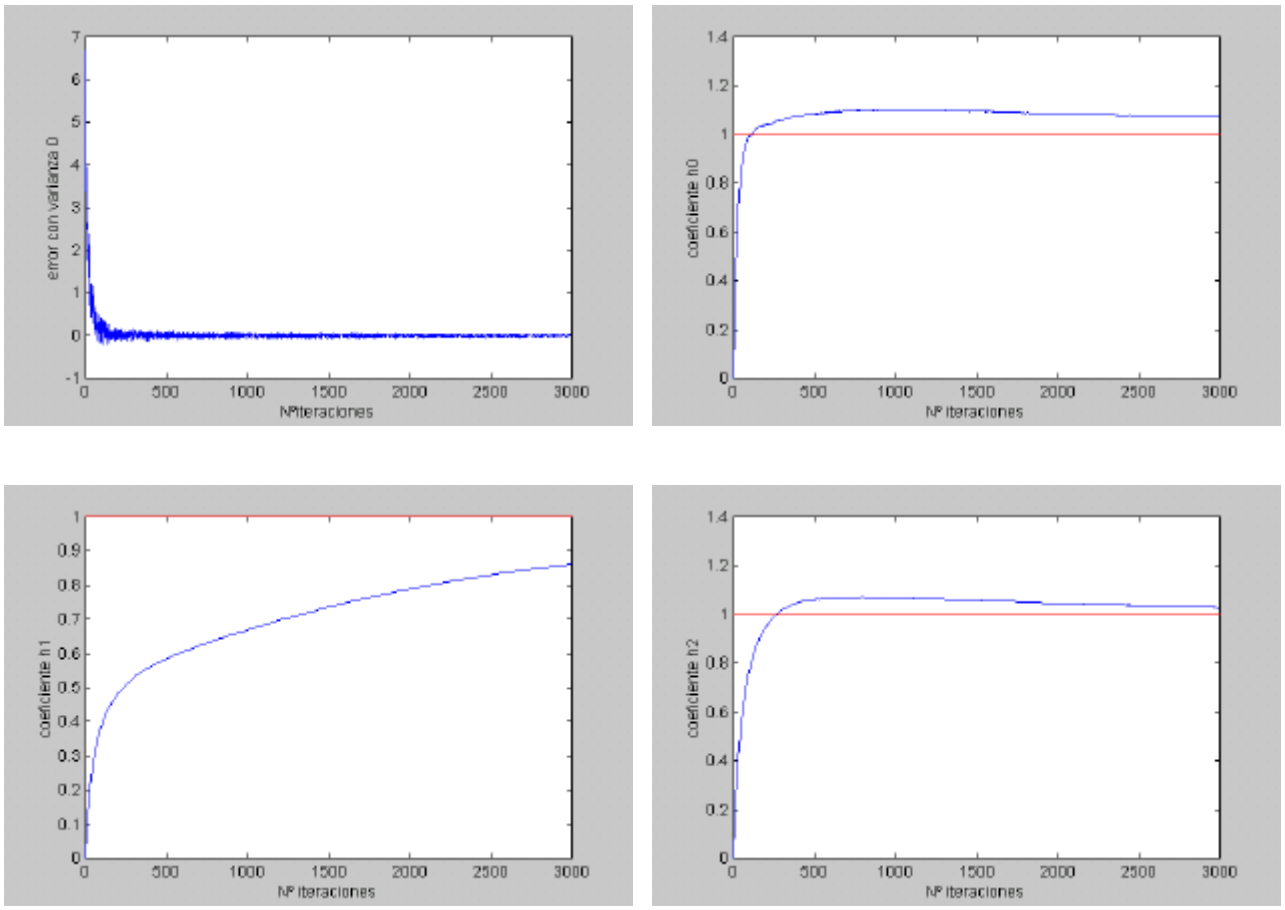

Para este valor del paso de adaptación, el algoritmo comienza a tener problemas de convergencia. De hecho, según el criterio del 3%, no hay convergencia.

Si se representan las últimas 300 muestras del coeficiente $h_2$ se observa que los valores son demasiado grandes, lo que implica la divergencia del algoritmo para estos valores. Así:



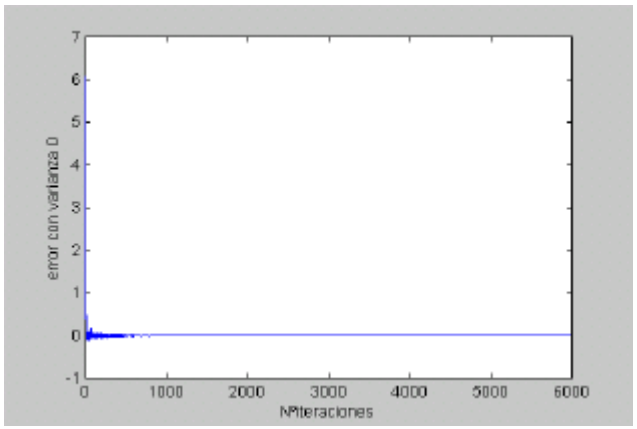

Vamos a analizar que ocurre cuando ***se sube la media del proceso de entrada de 0 a 1***. Conviene recordar que con el algoritmo LMS se tuvieron muchos problemas de convergencia en cuanto se subía la media del proceso de entrada.

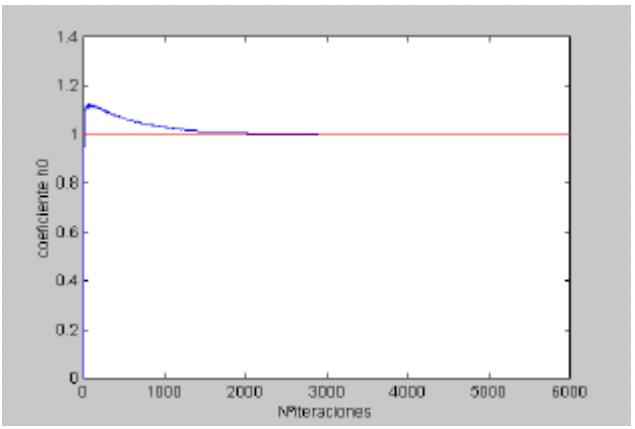
Se comienza con un valor del **paso de adaptación** igual a **0.01** con el fin de poder comparar en igualdad de condiciones con el algoritmo LMS. Así:

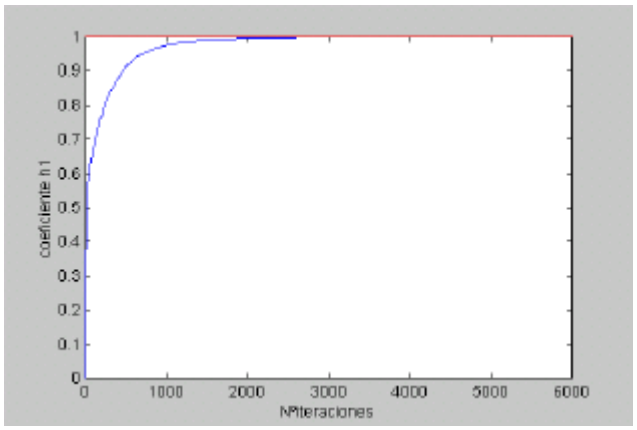


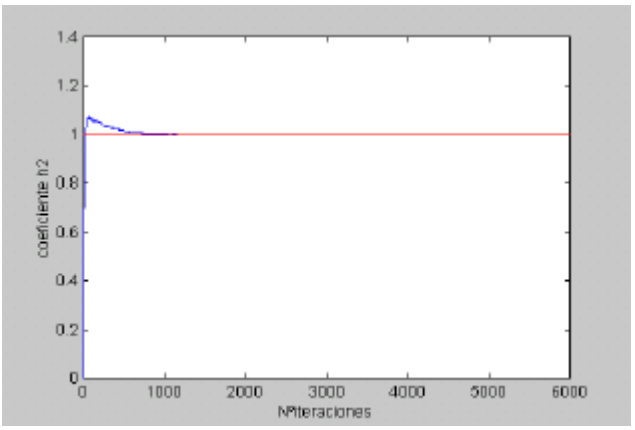

Según el criterio del 3%, la convergencia se produce en la iteración 2941. En estas mismas condiciones, con el LMS, ésta se producía en la 4549. Es decir, la convergencia ahora es bastante más rápida aunque, como se ha visto antes, se necesita un mayor número de operaciones en punto flotante.

Subimos ahora el valor del paso de adaptación a **m=0.08**. Con un valor bastante menor, el algoritmo LMS ya daba problemas con la convergencia. Los resultados son:



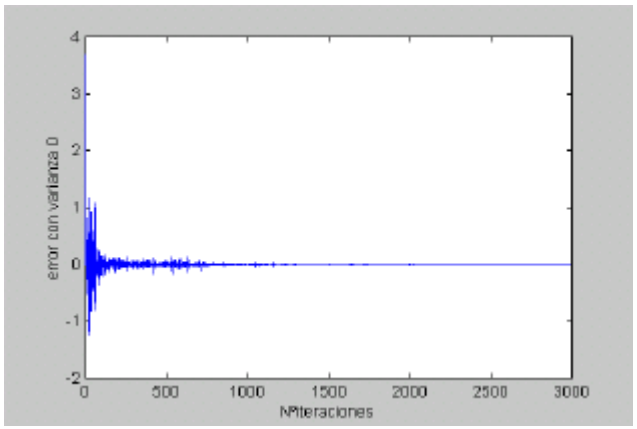



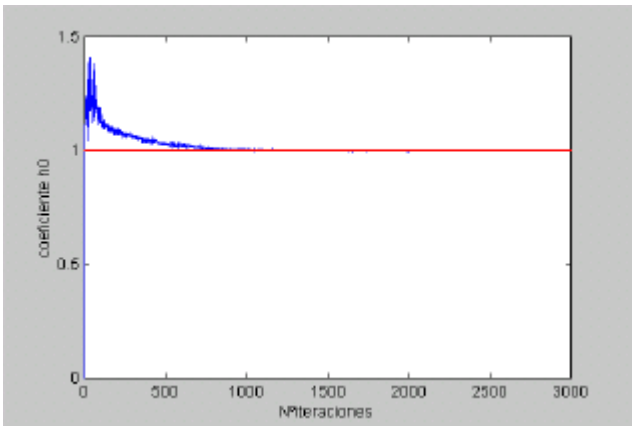



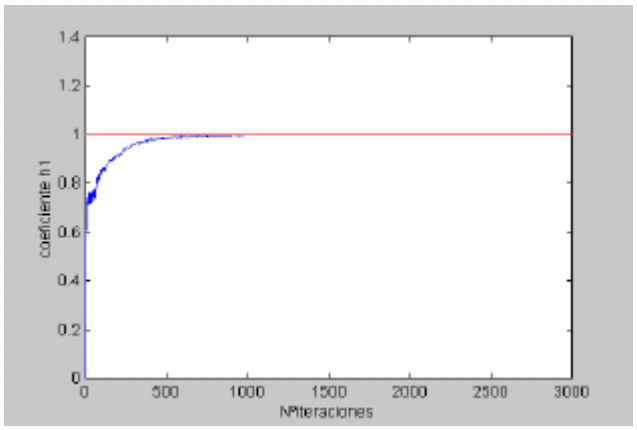



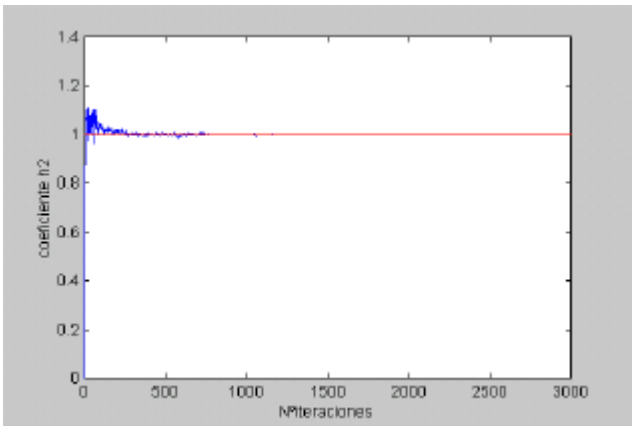

El algoritmo converge bastante bien. De hecho, según el criterio del 3%, ésta se produce en la iteración 459. Como se ha dicho antes, en estas condiciones el algoritmo LMS no convergía. Ahora, no sólo se obtiene la convergencia, sino que ésta se produce con bastante celeridad. Por tanto, se puede concluir que si el proceso de entrada tiene media distinta de cero (un valor pequeño), el algoritmo LMS normalizado funciona muchísimo mejor en cuanto a velocidad de convergencia. Esto es lógico si se tiene en cuenta que en este último algoritmo se normaliza precisamente respecto de la señal de entrada. Es decir, se reduce la influencia de la entrada sobre el algoritmo.

Por último, se simula este mismo caso con un **paso de adaptación** igual a **0.2**. los resultados fueron:



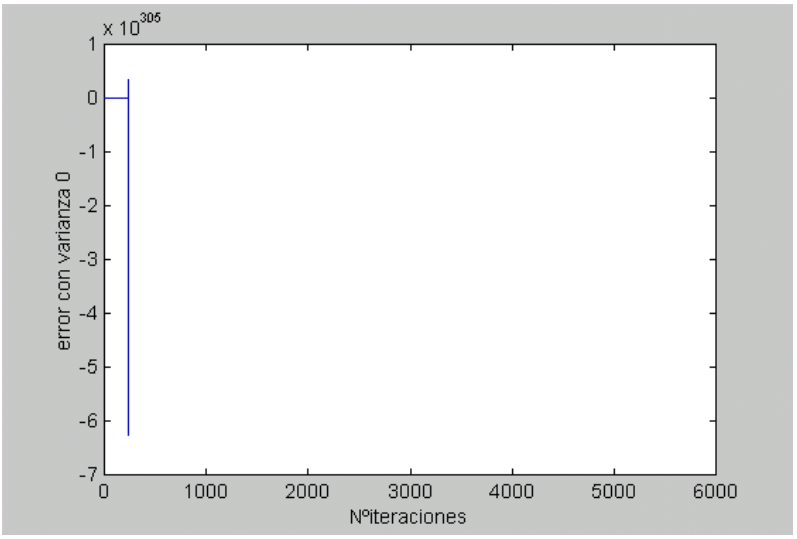



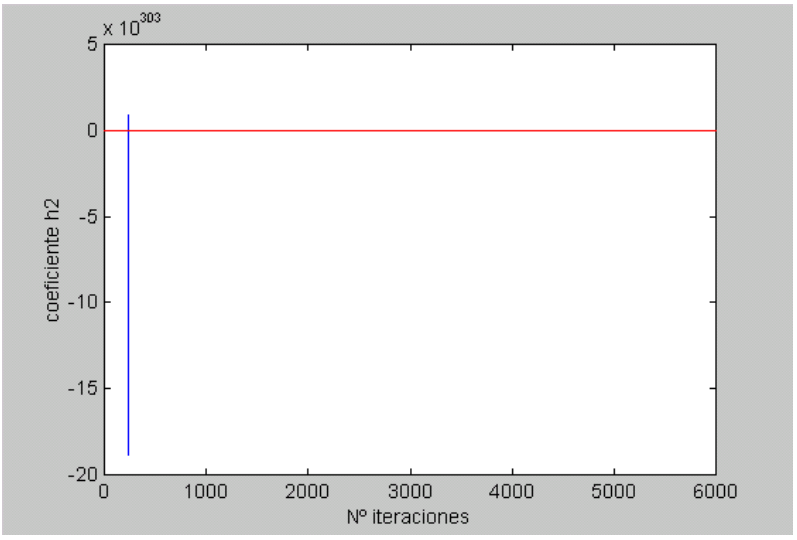




Al aumentar el paso de adaptación, la convergencia se hace más lenta. De acuerdo con el criterio del 3%, ésta se produce en la iteración 1161. Al igual que ocurría con el LMS, la convergencia va empeorando al aumentar el paso de adaptación. La diferencia estriba en que dicho empeoramiento tarda más en producirse en el algoritmo LMS normalizado.

Vamos a subir un poco más la dificultad: se considera ahora ***un proceso de entrada de media 10***. Con el LMS siempre hubo problemas cuando la media era grande. Se analiza a continuación si con la normalización se consiguen reducir esos problemas.

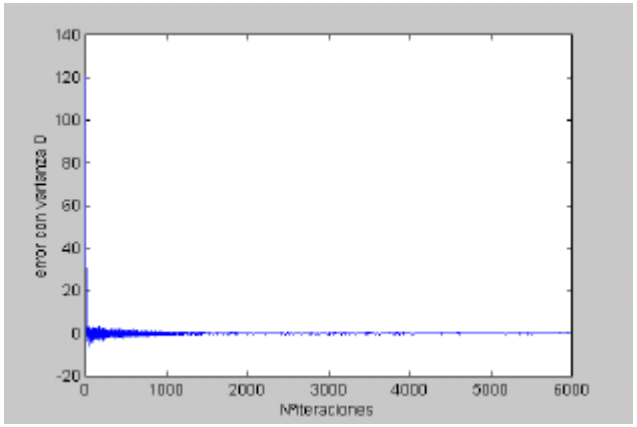
Para empezar, se tomó un **paso de adaptación** igual a **0.08**. En principio, es un valor demasiado grande para una media del proceso de entrada tan elevada. Los resultados fueron:

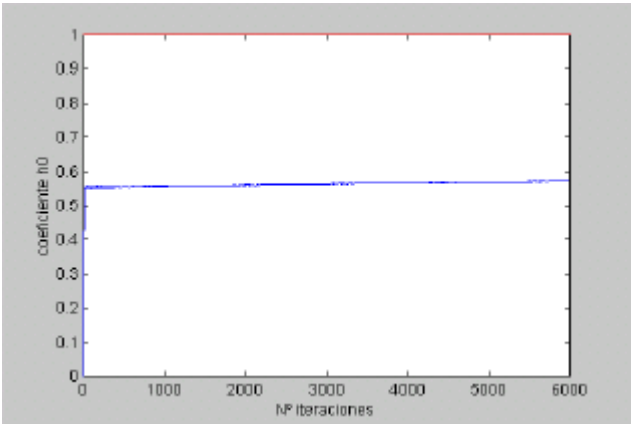


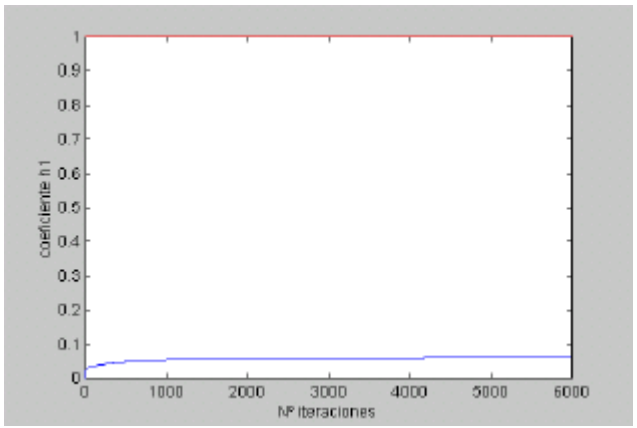



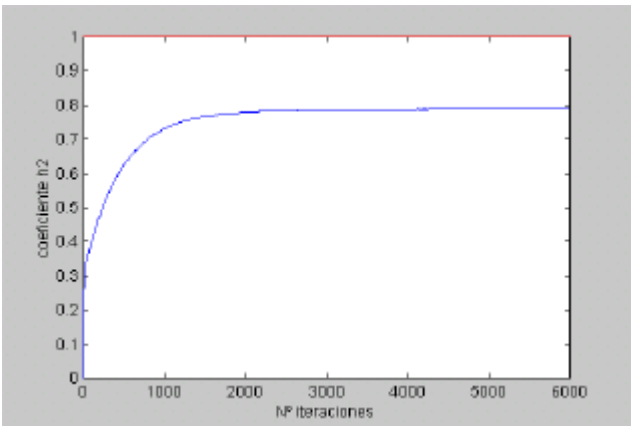

Sólo se ha representado la señal de error y uno de los coeficientes ($h_2$) porque es suficiente para mostrar que el algoritmo diverge. En realidad, con la señal de error ya se ve que el algoritmo no converge. Luego, con este proceso de entrada el paso de adaptación debe ser más pequeño.

Para las siguientes gráficas se ha utilizado un valor de **0.001**. Por un lado es un valor pequeño pero hay que tener en cuenta que es bastante más grande que los que se utilizan en estas condiciones con el algoritmo LMS. Los resultados fueron:



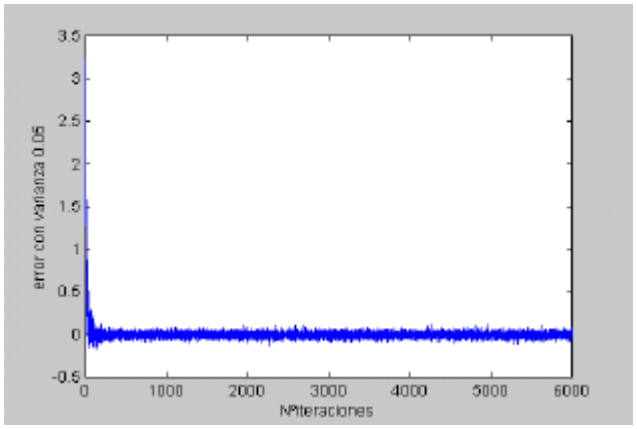



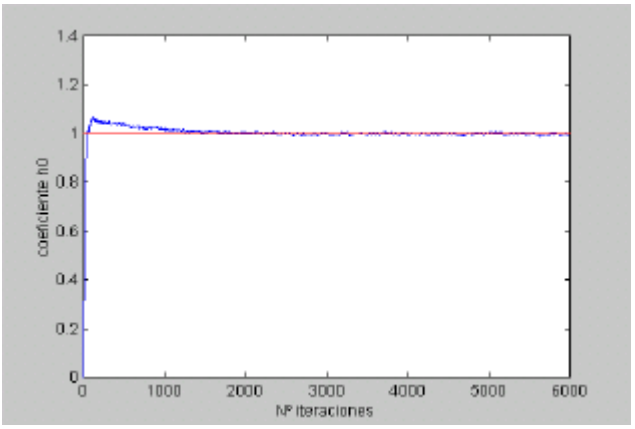

Los resultados no son muy buenos. Según el criterio del 3%, no se alcanza la convergencia en estas 6000 iteraciones. La señal de error parece pequeña pero se debe a la escala que se utiliza en el eje de ordenadas. No obstante, los resultados son bastante mejores que para el algoritmo LMS.

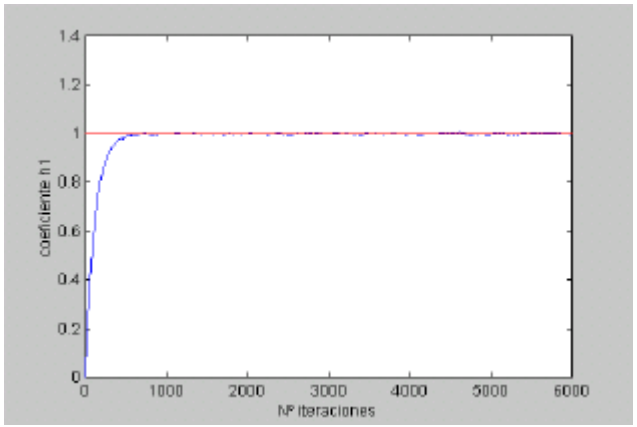
- $\underline{s_w^2 = 0.05}$:

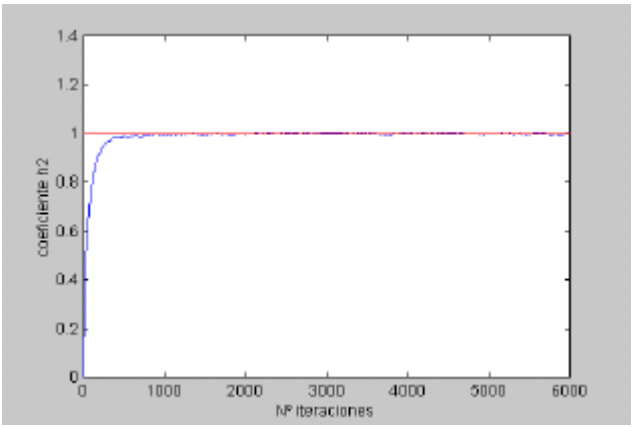
Se va a considerar ahora la situación en la que se tiene un ruido pequeño, de varianza 0.05. Lógicamente el criterio de convergencia se debe cambiar igual que se hizo en el caso del LMS. Con el fin de poder comparar, se considerarán los dos mismos criterios anteriores: el del 10% y el del 15%.

Para empezar, supondremos ***un proceso de entrada con media cero***. Se toma un **paso de adaptación** igual a **0.03**. Los resultados son:



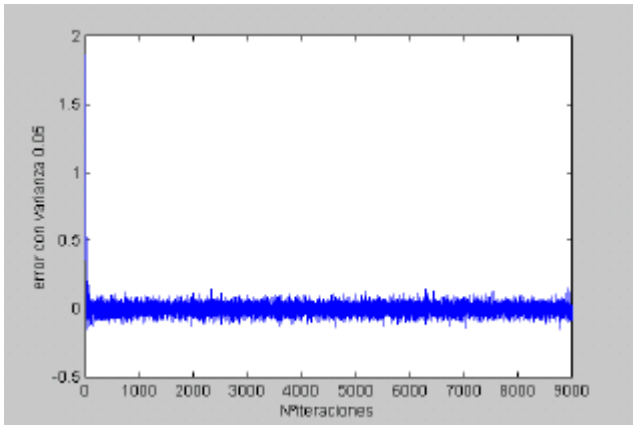



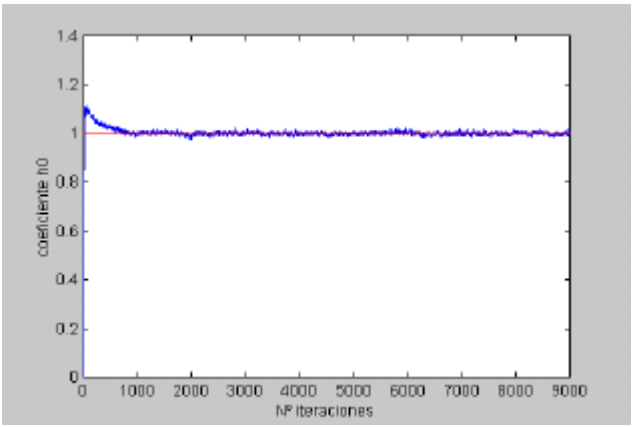



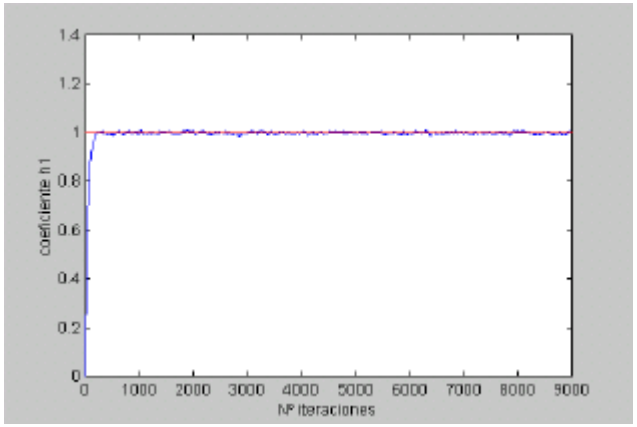

La convergencia se produce en la iteración 5640 según el criterio del 10% y en la 135 según el del 15%. Lo primero que llama la atención es la diferencia tan grande entre un criterio y otro. Esta diferencia tan grande no se daba en el algoritmo LMS. Por tanto, si no se precisa de una convergencia muy precisa, el algoritmo LMS normalizado es muchísimo más rápido que el LMS. Sin embargo, si se quiere una convergencia más exacta, la diferencia entre ambos algoritmos ya no es tan apreciable. Es cierto que con el LMS

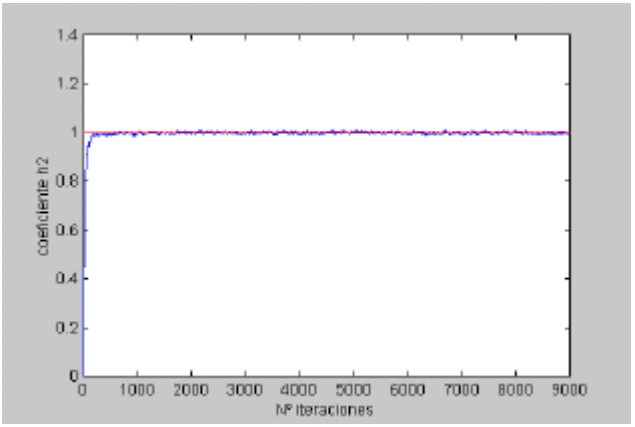
normalizado ésta se produce un poco antes pero también que es necesario un mayor número de operaciones en punto flotante. La elección de uno u otro dependerá de la situación.

Se simula a continuación con un **paso de adaptación** igual a **0.08**. Con un valor un poco menor (0.06) el algoritmo LMS ya no convergía. Los resultados son:



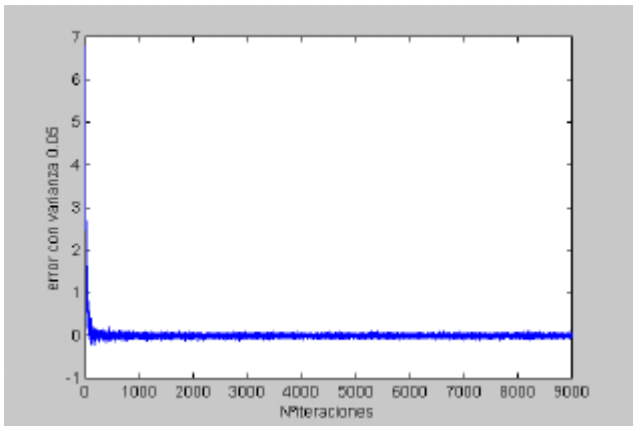



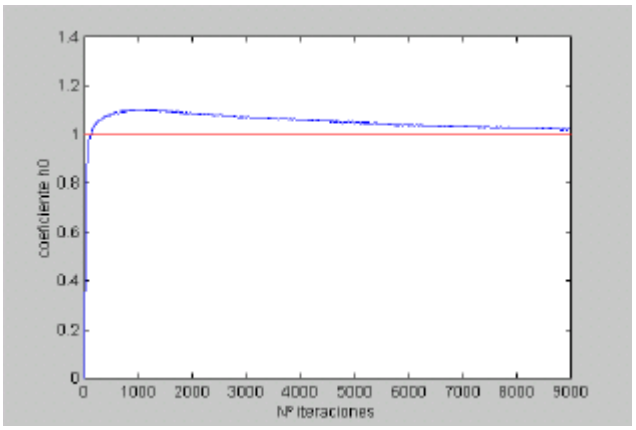

El algoritmo converge en la iteración 8971 según el criterio del 10% y en la 52 según el del 15%. Aquí se pone todavía más de manifiesto la gran diferencia entre ambos criterios. Tal y como se ha dicho antes, si no se precisa una convergencia muy exacta, la elección de este algoritmo LMS normalizado es evidente: la velocidad de convergencia es alarmantemente superior por lo que se compensa la diferencia en el número de operaciones al reducirse las iteraciones necesarias.

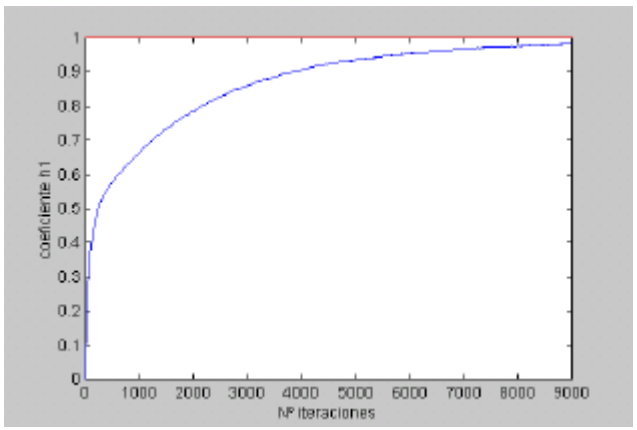
Se considera ahora ***un proceso de entrada de media uno***. En el algoritmo LMS fue necesario un paso de adaptación pequeño para conseguir la convergencia en esta misma

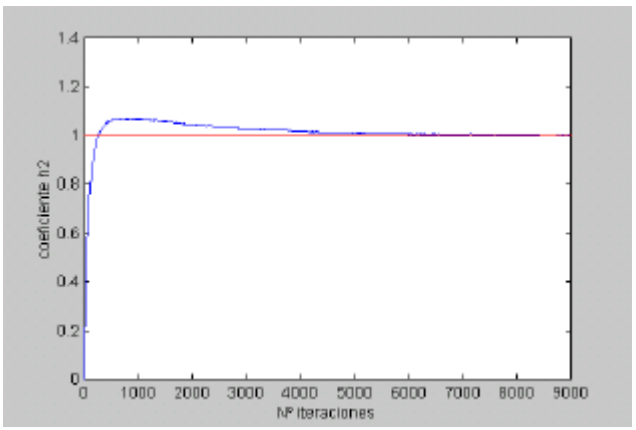
situación. A priori ahora se podrá utilizar un valor un poco más grande que antes puesto que en este algoritmo se normaliza respecto a la entrada.

Así, para poder comparar, se empieza con **m=0.01**. Los resultados son:



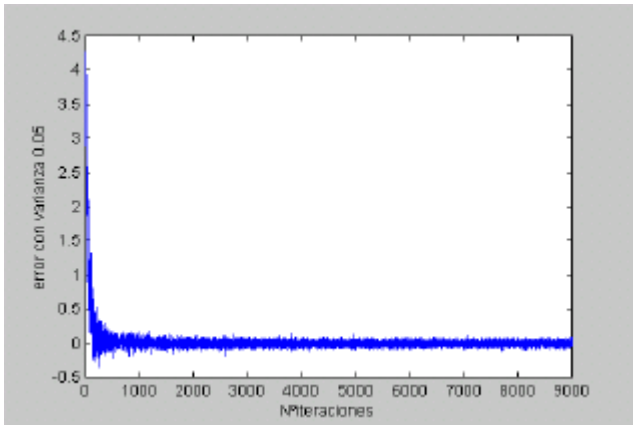



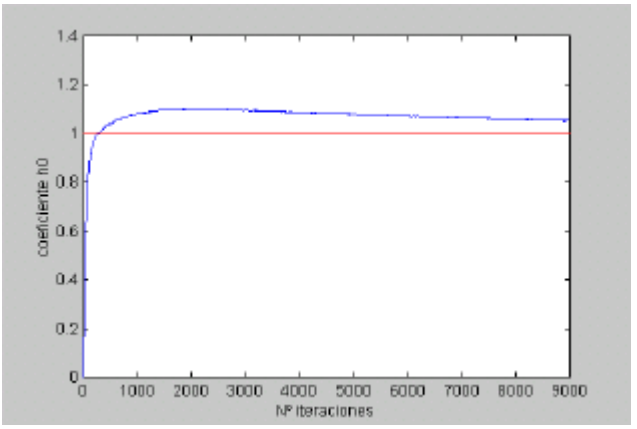



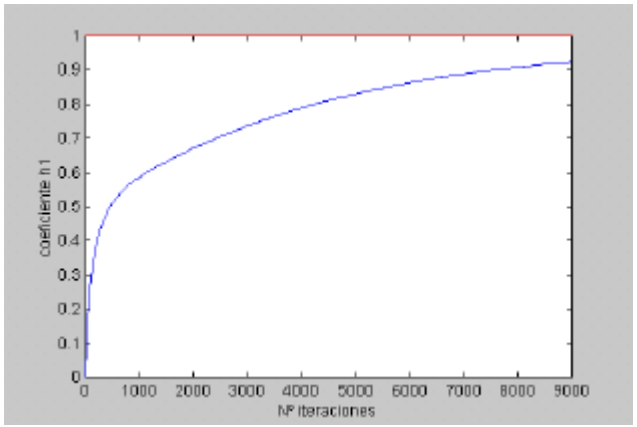



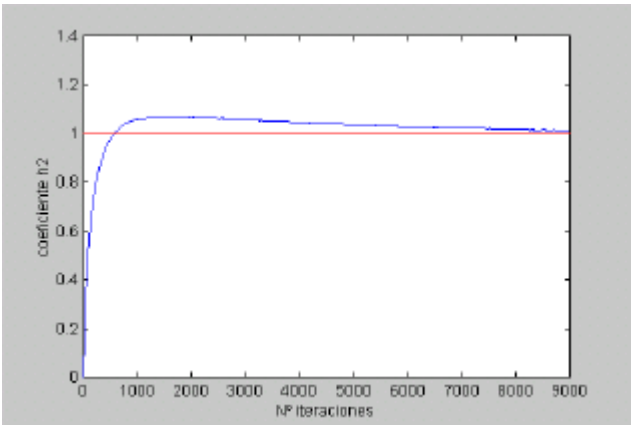

Se consigue la convergencia en las iteraciones 8979 y 435 según los criterios del 10% y del 15%, respectivamente. Nuevamente se pone de manifiesto una gran diferencia entre ambos criterios. Este valor del paso de adaptación era grande para conseguir la convergencia con el algoritmo LMS.

Si bajamos este parámetro a un valor igual a **0.005**, se podrá realizar una comparación más acertada con el algoritmo LMS. Así, los resultados son:



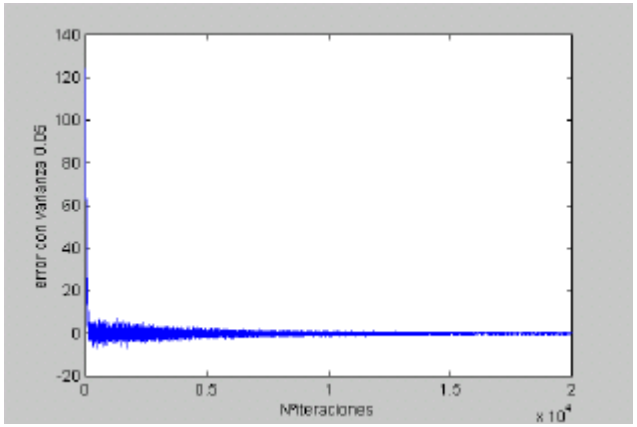



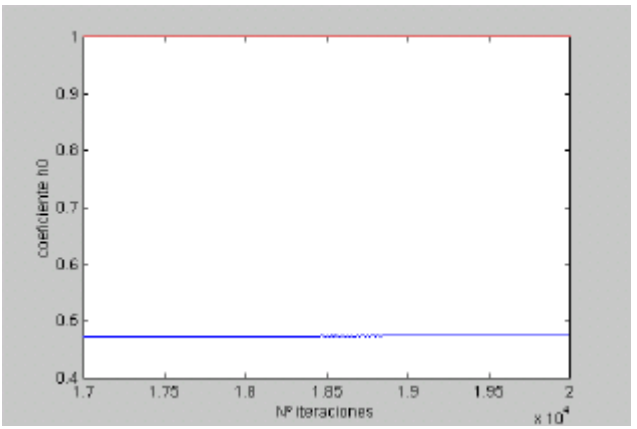



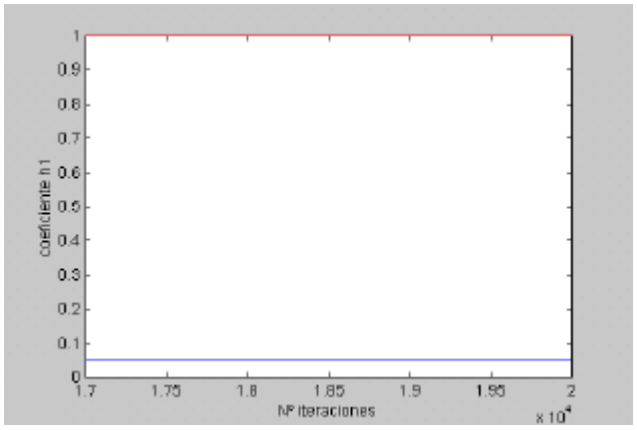



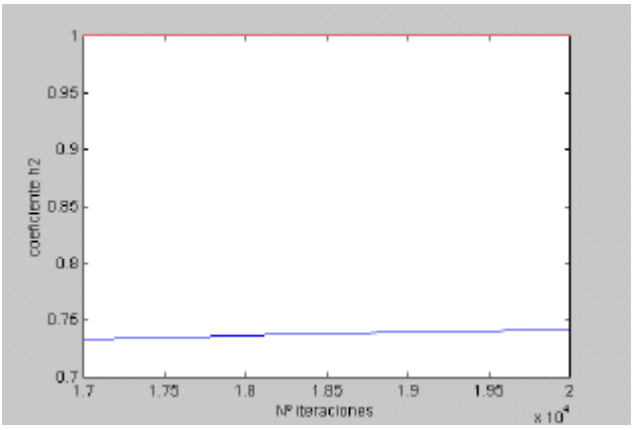

Los valores cambian para este caso. La convergencia se produce en las iteraciones 8158 y 2581 según los criterios del 10% y del 15%, respectivamente. Hay que comentar varias cuestiones. Por un lado, ya no hay una diferencia tan grande entre ambos criterios aunque ésta es aún considerable. Por otro lado, las diferencias con el algoritmo LMS no son tan evidentes como antes. Según el criterio del 10% (convergencia más exacta), el algoritmo LMS converge con más velocidad. Sin embargo, si no se precisa esa convergencia, es más rápido el LMS normalizado. Lógicamente sigue existiendo entre ambos algoritmos una diferencia en el número de operaciones necesarias.

Finalmente, se tiene en cuenta ***un proceso de entrada de media 10***. Se ha utilizado un **paso de adaptación** igual a $\mathbf{0.1*10^{-3}}$. Se ha simulado con 20000 iteraciones. Los coeficientes sólo se representan durante las 3000 últimas iteraciones para verlos mejor. Los resultados son:

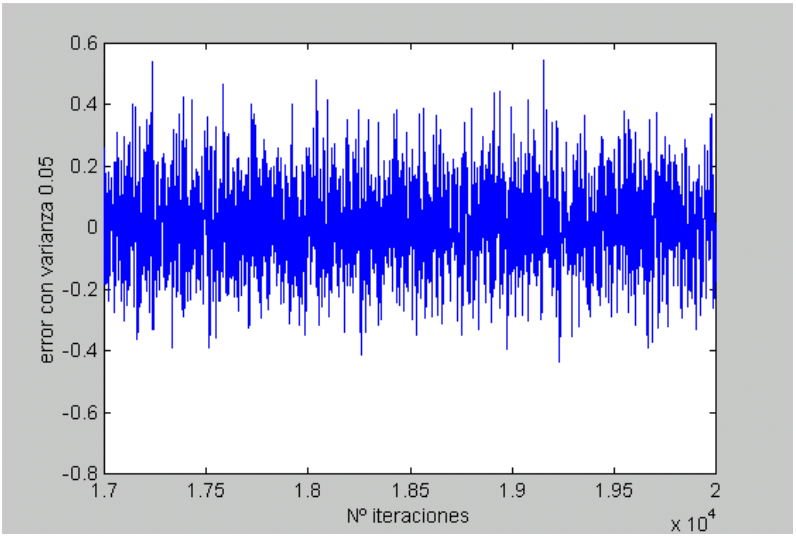



Los resultados no son del todo buenos. El algoritmo no converge con ninguno de los dos criterios. La estimación de los coeficientes es bastante lenta, como siempre que la media del proceso de entrada es igual a 10. No obstante, el coeficiente $h_2$ se aproxima ligeramente a su valor final y es el responsable de que la señal de error sea relativamente pequeña en las últimas iteraciones. Esto es así porque este coeficiente es el que multiplica al cuadrado de la señal. De hecho, representando sólo las 3000 últimas muestras de la señal de error, se observa el cambio de escala en el eje de ordenadas respecto a la gráfica con las 20000 iteraciones. Así, se tiene:

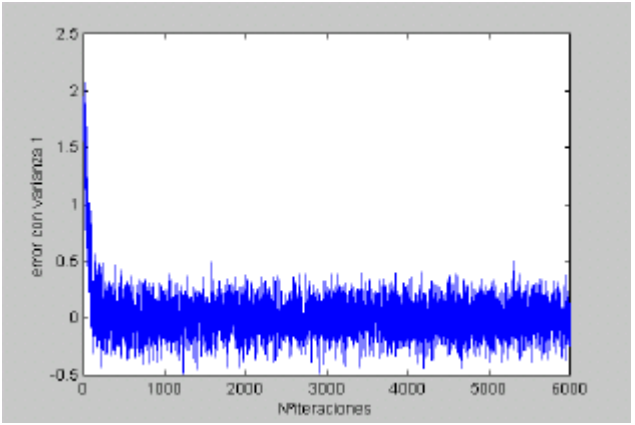


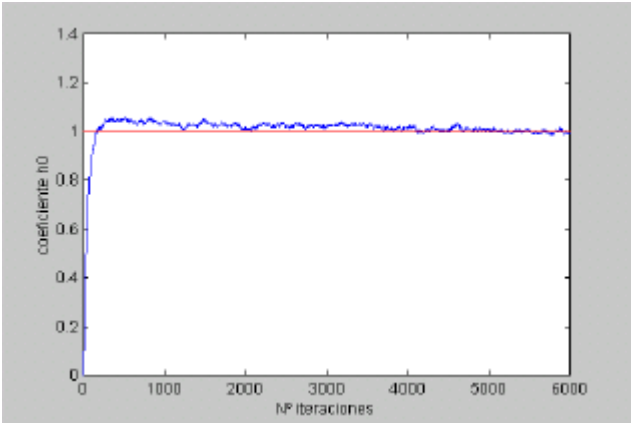

Para concluir con este algoritmo, vamos a estudiar a continuación el caso en el que existe un ruido de varianza igual a 1, que evidentemente es la peor situación posible para estimar bien los coeficientes.

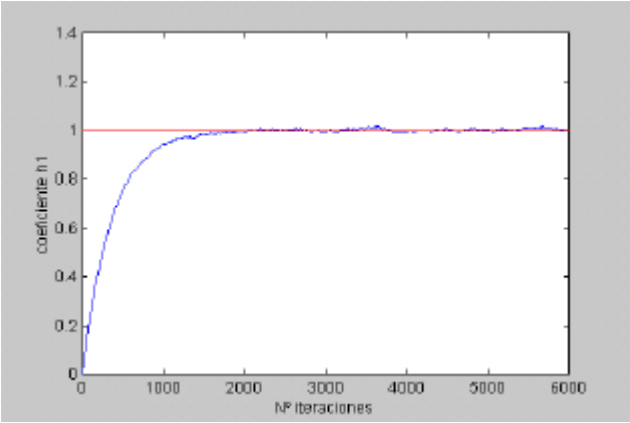
- $\sigma_w^2 = 1$:

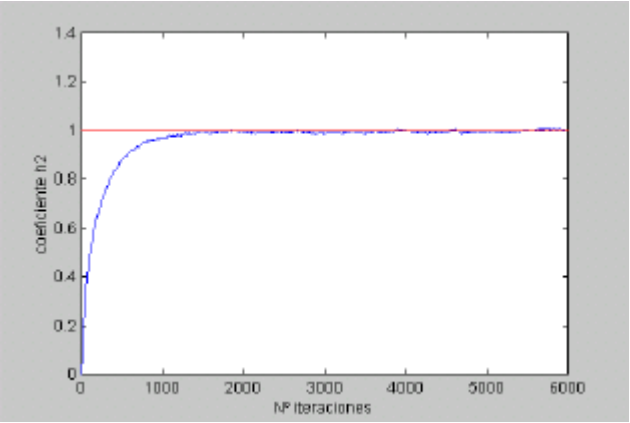
En este último caso se vuelven a utilizar los criterios de convergencia que ya se utilizaron en el algoritmo LMS, es decir, considerar que existe convergencia a partir de que la señal de error se mantenga por debajo de $\pm 0.5$ (criterio del 50%) o de $\pm 0.55$ (criterio del 55%).

Como hasta ahora se comienza con ***un proceso de entrada de media cero***. Para poder comparar se simula con **m=0.01** puesto que es un valor que se utilizó en el algoritmo LMS. Las gráficas son:



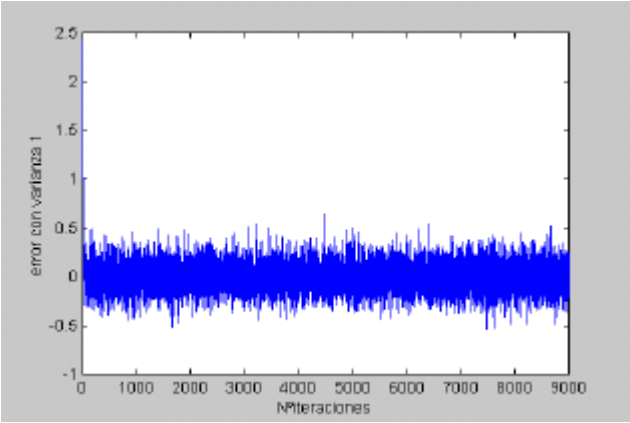



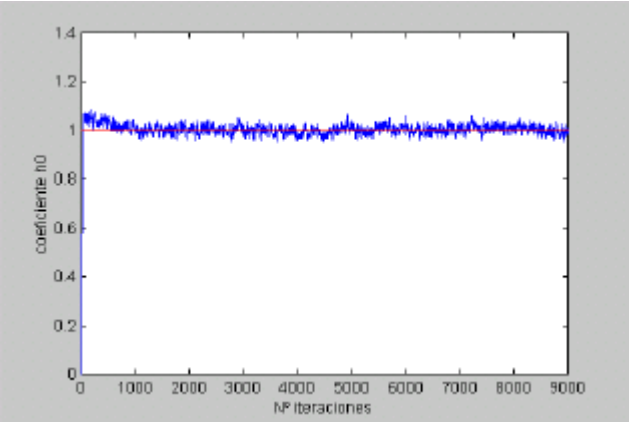


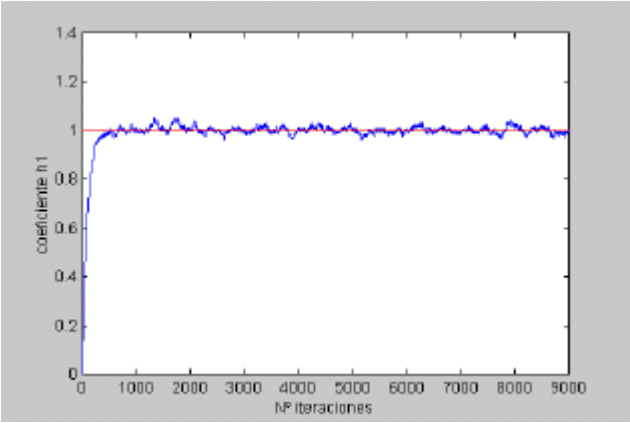



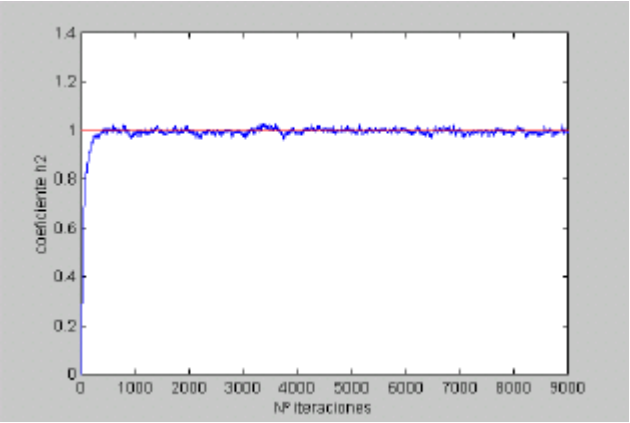

La convergencia es bastante buena. Se produce en las iteraciones 5313 y 153 según el criterio del 50% y del 55%, respectivamente. Con cualquiera de los dos criterios, la convergencia es más rápida que en el caso del LMS. Con respecto al criterio del 50%, la diferencia en cuanto a velocidad de convergencia entre ambos algoritmos es bastante reducida: en el LMS, se converge en la iteración 5796 mientras que en el LMS normalizado, en la 5313. Sin embargo, si se tiene en cuenta el criterio del 55%, la diferencia entre ambos sí es importante: iteración 1836 frente a la 153 que se acaba de obtener. No hay que olvidar que el algoritmo LMS normalizado necesita un mayor número de operaciones. Dependerá de lo que se requiera en cada situación para elegir uno u otro.

Se repiten las simulaciones con un valor del **paso de adaptación** igual a **0.04**, que es el valor para el que comenzaban los problemas en la convergencia con el caso LMS. Los resultados son:



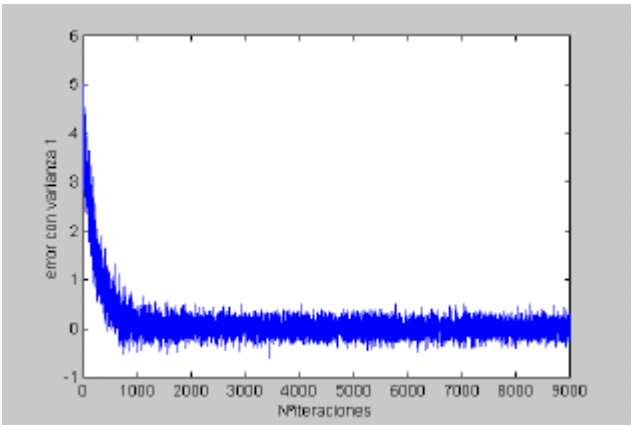


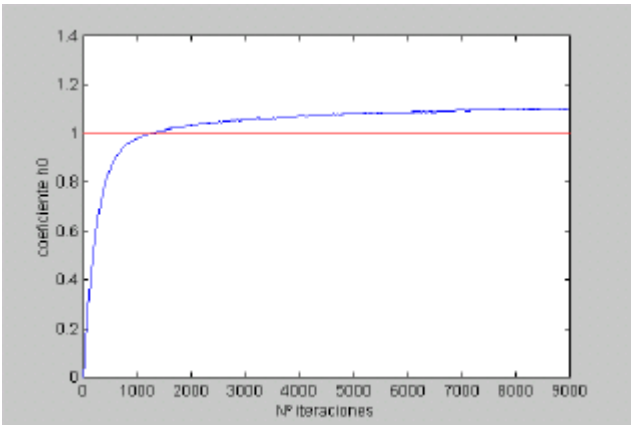



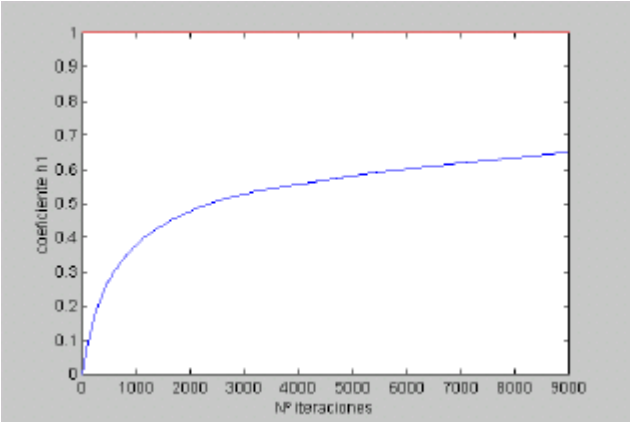

La convergencia se produce en las iteraciones 8670 y 4479 de acuerdo con los criterios del 50% y del 55%, respectivamente. Sigue existiendo una importante diferencia entre ambos criterios. Esto no ocurría en el caso LMS en estas mismas circunstancias, ya que para ambos criterios la convergencia se producía en la iteración 8992. Nuevamente, si se requiere velocidad por encima de exactitud, el LMS normalizado es el algoritmo que se comporta mejor.

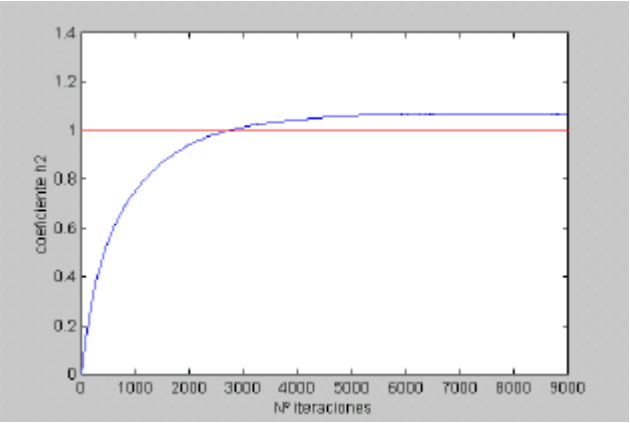
Se trabaja ahora con ***un proceso de entrada de media uno***. Sabemos de las simulaciones anteriores que el valor del paso de adaptación debe reducirse al aumentar la media del proceso de entrada.

Se utilizan los mismos valores que en el algoritmo LMS con el fin de poder compara los resultados. Así, se comienza con un **paso de adaptación** de **0.001**. Se obtiene:



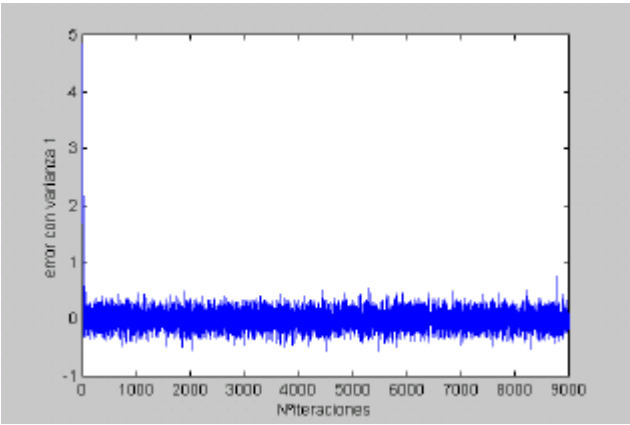



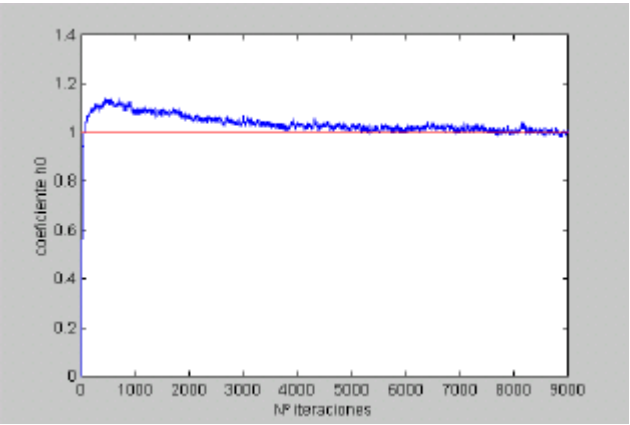


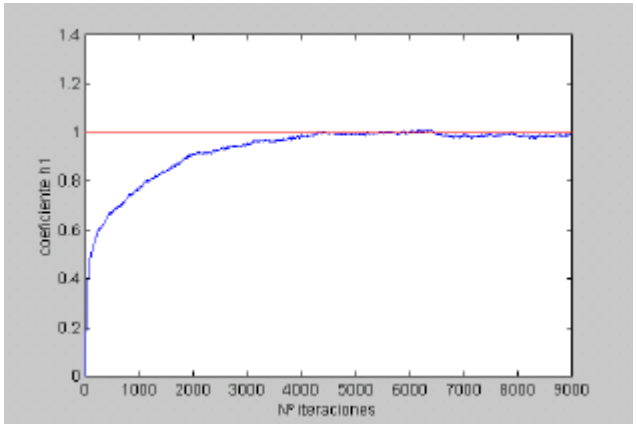



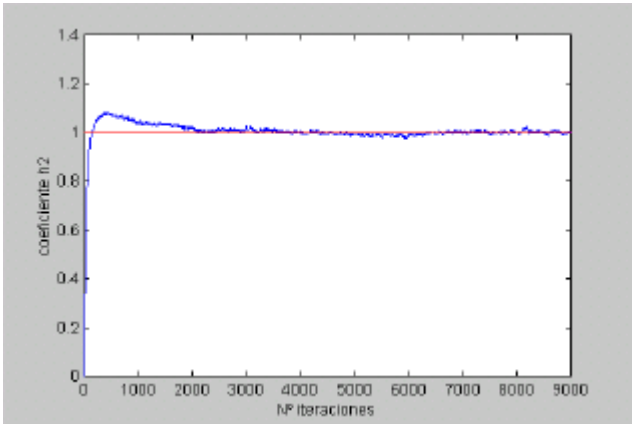

De acuerdo con el criterio del 50%, la convergencia se produce en la iteración 5782 mientras que siguiendo el del 55%, ésta se produce en la 3425. Estos resultados son un poco mejores que para el LMS, donde se obtuvo 6524 y 5518 respectivamente. La presencia del ruido desvirtúa un poco la normalización respecto a la entrada puesto que no se obtiene tanta diferencia entre los algoritmos como cuando no había ruido o éste era pequeño. No obstante, el LMS normalizado es un poco más rápido aunque, como ya se ha dicho varias veces, requiere un mayor número de operaciones.

Con este mismo proceso de entrada se vuelve a simular pero subiendo considerablemente el paso de adaptación: **m=0.02**. Las gráficas son:



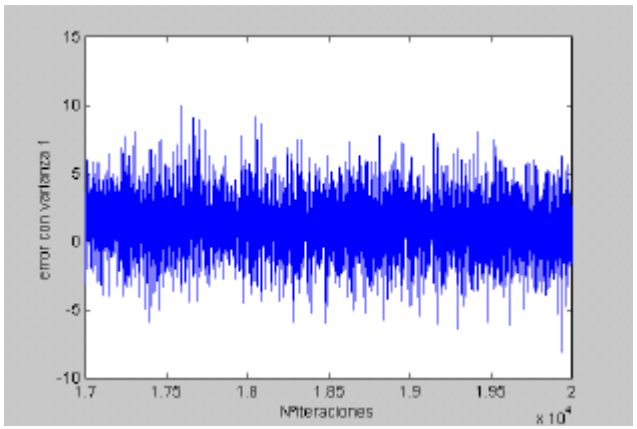


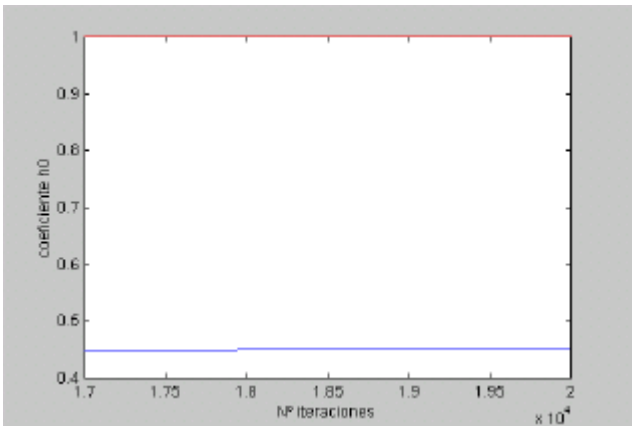



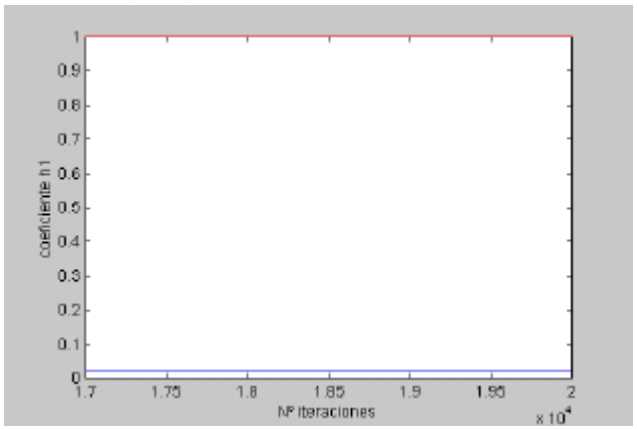

Los resultados en cuanto a convergencia son mejores que en el caso del LMS, para el que no se llegaba a producir la convergencia con ninguno de los dos criterios que tenemos. Sin embargo, en este caso, ésta se produce en la iteración 8764 para ambos criterios. Las conclusiones se repiten nuevamente.

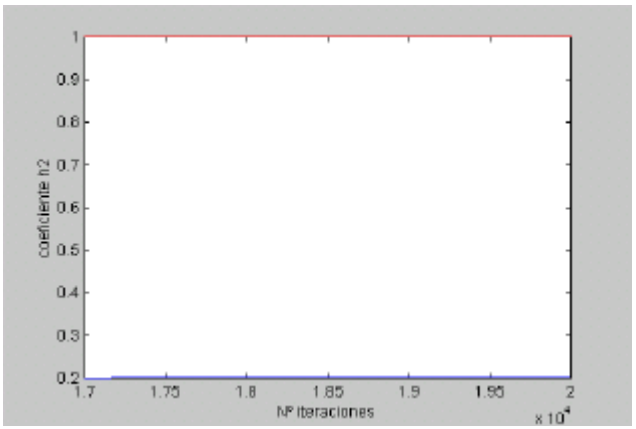
Finalmente, se va a considerar el caso en el que se tiene ***un proceso de entrada de media 10***. Este es el peor caso posible puesto que, además de la presencia de un ruido grande, se tiene una entrada con una media también grande.

Se toma un paso de adaptación pequeño: $\mathbf{m=0.1*10^{-5}}$. Se han simulado 20000 iteraciones aunque sólo se representan las 3000 últimas para observar mejor los valores:






Los resultados son malos. No se llega a obtener convergencia tal y como se aprecia perfectamente en las gráficas. La presencia del ruido y de una entrada de media 10 obligan a tomar un paso de adaptación muy pequeño para que el algoritmo no diverja. El problema es que este paso de adaptación tan pequeño provoca que apenas se avance nada en cada iteración. Serían necesarias demasiadas iteraciones.

Al igual que con el algoritmo LMS, vamos a resumir todos los datos en una tabla:

| Nºiteraciones | $s_w^2$ | media | m | criterio | convergencia |
|---|---|---|---|---|---|
| 3000 | 0 | 0 | 0.01 | 3% | 869 |
| 3000 | 0 | 0 | **0.08** | 3% | **131** |
| 3000 | 0 | 0 | 0.4 | 3% | X |
| 3000 | 0 | 1 | 0.01 | 3% | 2941 |
| 6000 | 0 | 1 | **0.08** | 3% | **459** |
| 3000 | 0 | 1 | 0.2 | 3% | 1161 |
| 6000 | 0 | 10 | 0.08 | 3% | X |
| 6000 | 0 | 10 | 0.001 | 3% | X |
| 6000 | 0.05 | 0 | **0.03** | 10%/15% | **5640/135** |
| 9000 | 0.05 | 0 | 0.08 | 10%/15% | 8971/52 |
| 9000 | 0.05 | 1 | **0.01** | 10%/**15%** | 8979/**435** |
| 9000 | 0.05 | 1 | **0.005** | **10%**/15% | **8158**/2581 |
| 20000 | 0.05 | 10 | $0.1*10^{-3}$ | 10%/15% | X/X |
| 6000 | 1 | 0 | **0.01** | 50%/55% | **5313/153** |
| 9000 | 1 | 0 | 0.04 | 50%/55% | 8670/4479 |
| 9000 | 1 | 1 | **0.001** | 50%/55% | **5782/3425** |
| 9000 | 1 | 1 | 0.02 | 50%/55% | 8764/8764 |
| 20000 | 1 | 10 | $0.1*10^{-5}$ | 50%/55% | X/X |

Resultados con el algoritmo LMS normalizado

- **RLS:**

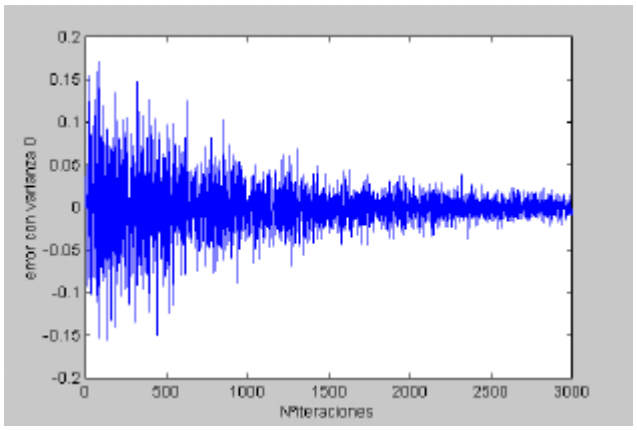
Este algoritmo es completamente distinto a los dos anteriores. Al igual que antes, se simularán distintas situaciones en las que se tendrán en cuenta la presencia o no de ruido y se trabajará con procesos de entrada con distintas medias.

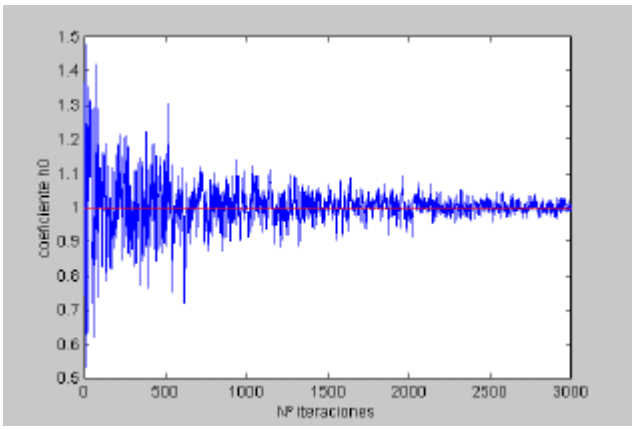
Tal y como se dijo al principio la mayoría de variantes de este tipo de algoritmos RLS no tienen el parámetro paso de adaptación (μ). Sin embargo, aquí sí se tendrá en cuenta para que el análisis sea más completo. En los algoritmos donde no aparece es porque se ha tomado igual a 1.

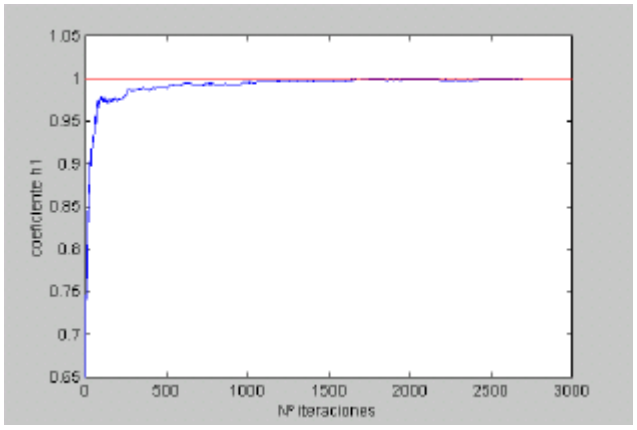
- $s_w^2 = 0$:

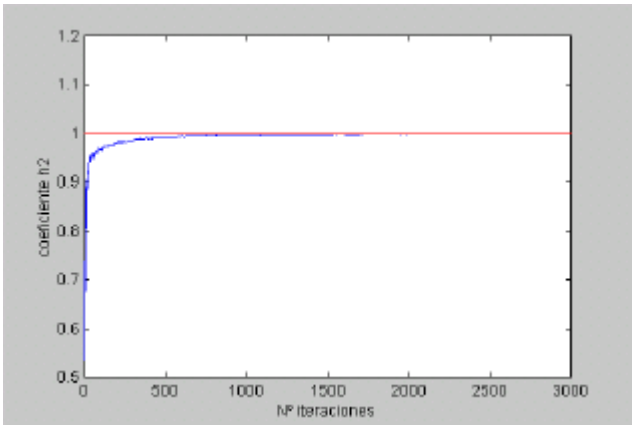
Como siempre, se empieza por la situación en la que no existe ruido. Se considera un ***proceso de entrada de media 0***, es decir, lógicamente se empieza por el caso más simple.

La primera simulación es considerando un **paso de adaptación** igual a **0.5**. se obtiene:



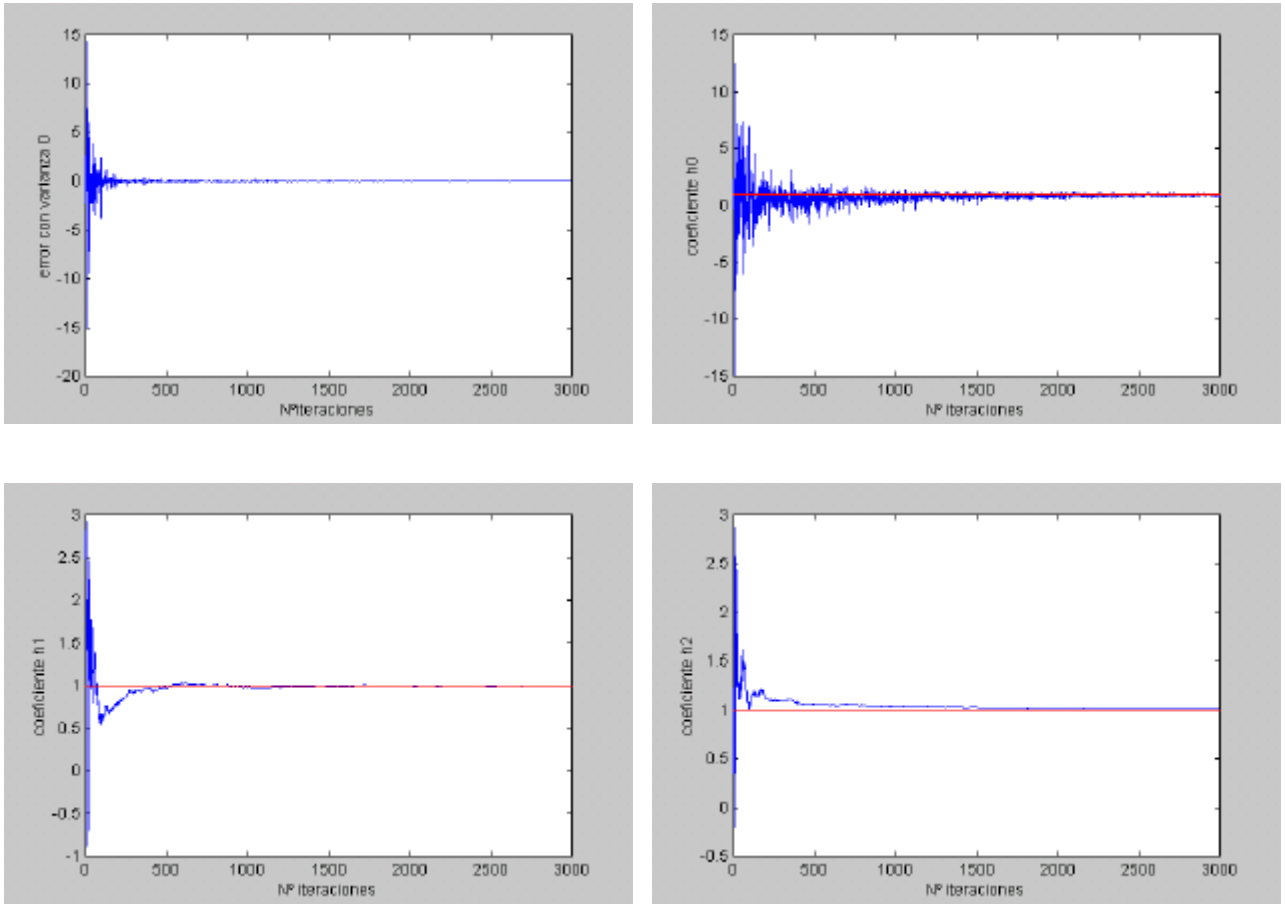



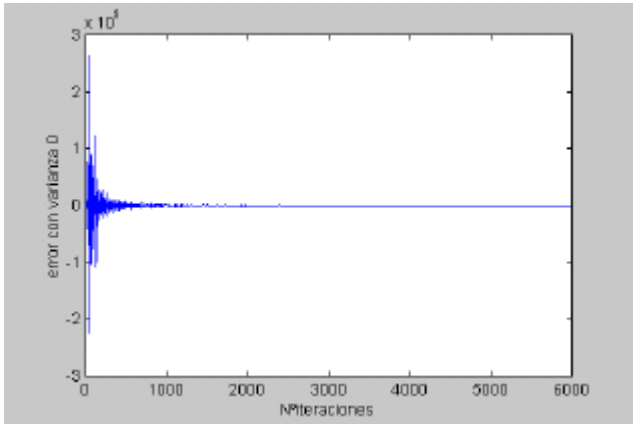
Como no hay ruido, se toma el criterio del 3% para la convergencia con el propósito de poder comparar los tres algoritmos. De acuerdo con este criterio, la convergencia se produce en la iteración 2364. Este valor es más lento que el que se obtuvo para los dos algoritmos anteriores. Además, el número necesario de operaciones es del orden de los 380 millones, es decir, muchísimo mayor que en los dos casos anteriores. Con estos resultados, este algoritmo no es nada bueno.

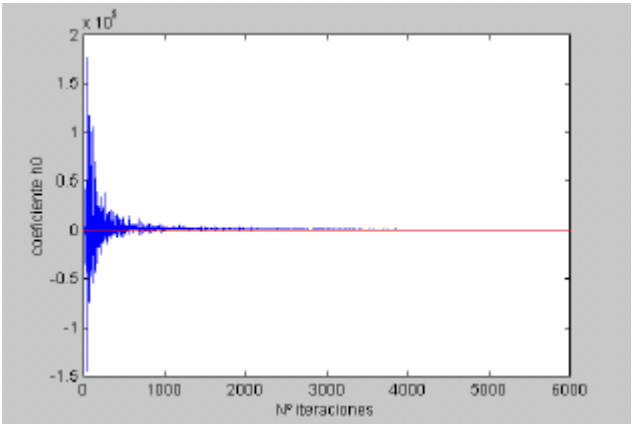
Subimos el **paso de adaptación** hasta un valor igual a **1**. Los resultados son:

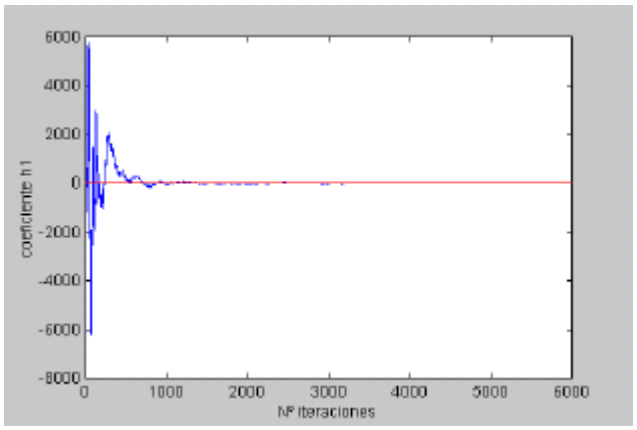


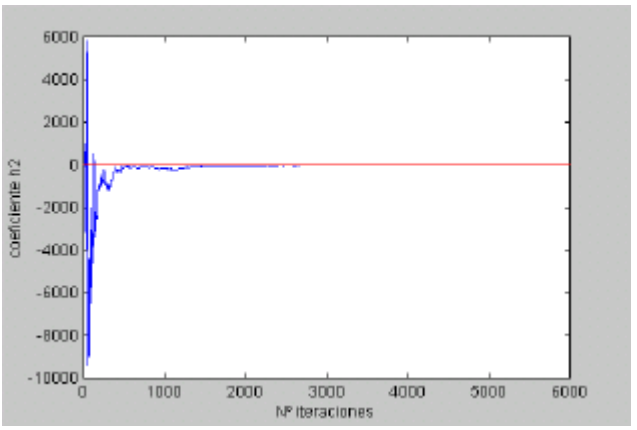

Ahora la convergencia se produce en la iteración 2618, es decir, es más lenta que la de antes. Este valor sigue siendo peor que el de los dos algoritmos anteriores y además el número de operaciones sigue siendo bastante elevado en comparación con los dos algoritmos anteriores.

Al aumentar el **paso de adaptación**, la tendencia es que la convergencia sea más lenta. Así, con **m=1.5**, se obtiene:

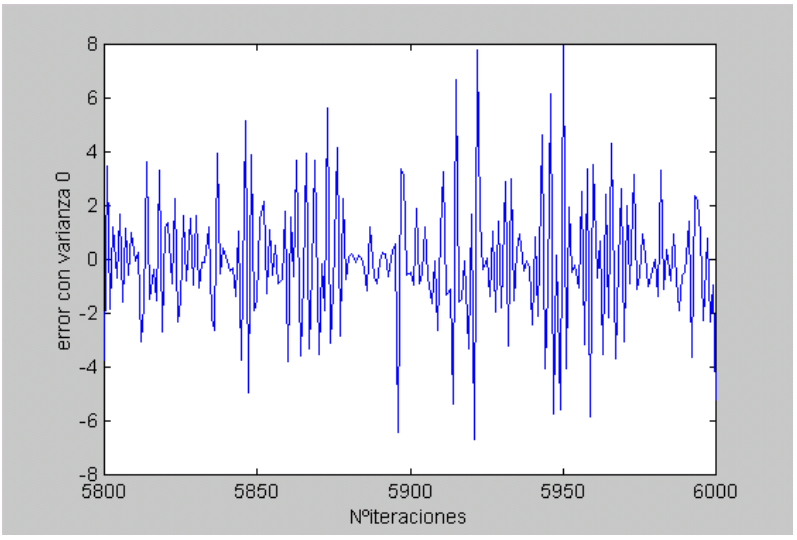


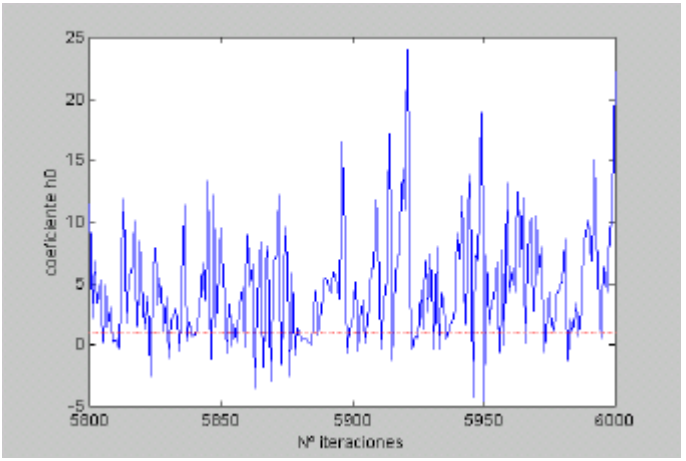



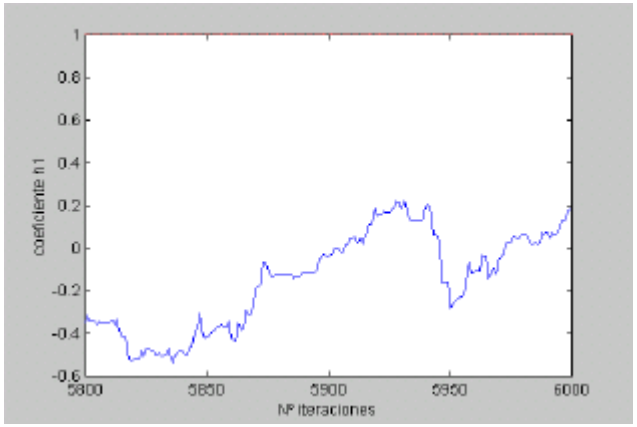



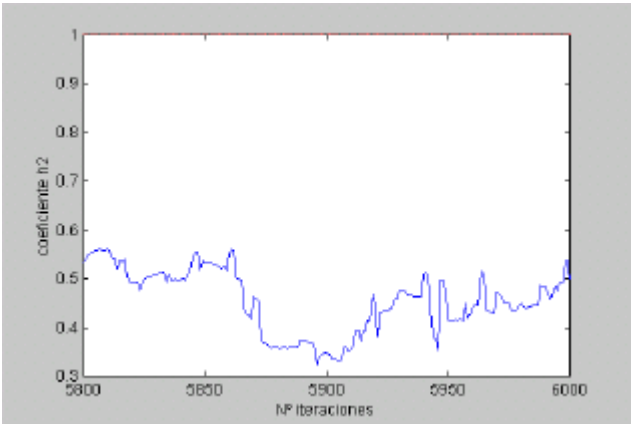

El algoritmo no converge de acuerdo con el criterio del 3%. De todas maneras, con los valores tan grandes del principio no se observan bien los valores finales. Por eso se representan a continuación sólo las 200 últimas muestras de cada una de las cuatro gráficas anteriores:



Se ve que los valores del eje de ordenadas son muy grandes para ser las últimas muestras. Los coeficientes serían:



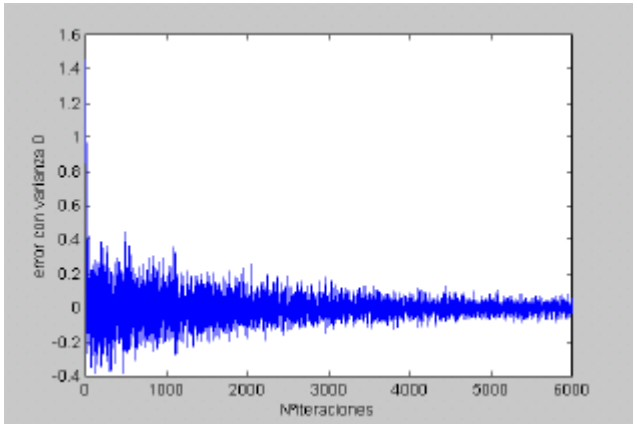



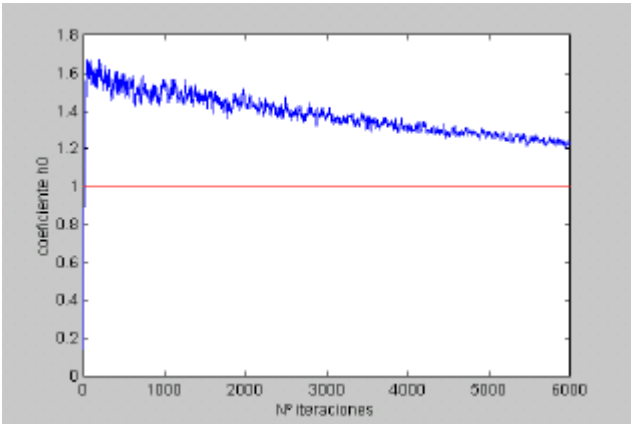



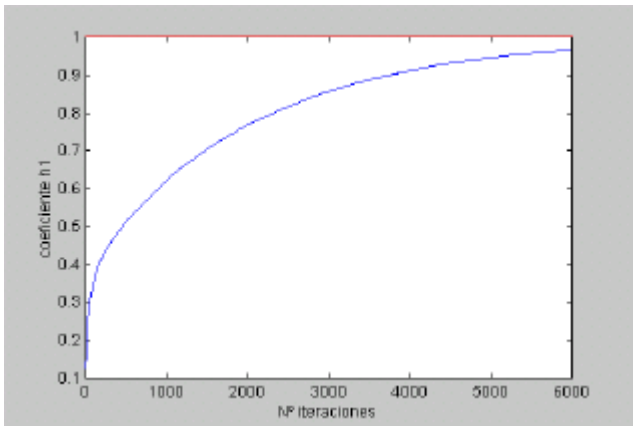

El coeficiente $h_0$ está completamente perdido con valores bastante elevados. Los otros dos no alcanzan valores tan elevados pero tampoco convergen.

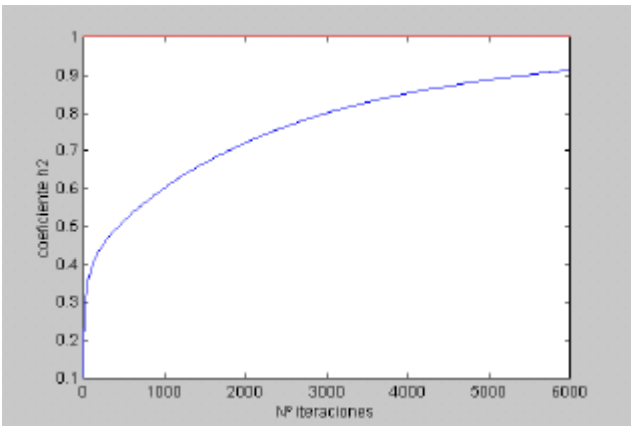
A priori ha podido sorprender que se consideren valores tan elevados para el paso de adaptación si se comparan con los de los algoritmos anteriores. Esto es debido a que para esta clase de algoritmos (RLS) el valor por defecto de este parámetro es la unidad tal y como se comentó al principio. Por eso se toman valores en torno a este dato y no tan pequeños como antes.

Para comprobar este último comentario, se va a simular con un **paso de adaptación** igual a **0.1**. Este valor sigue siendo mayor que los que se han usado en los algoritmos anteriores. Aun así, se obtiene:



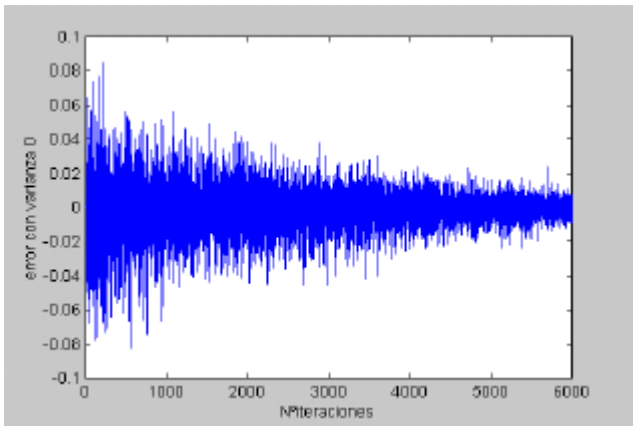



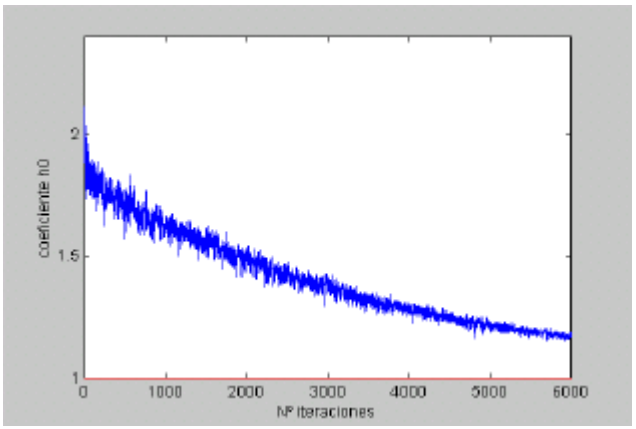



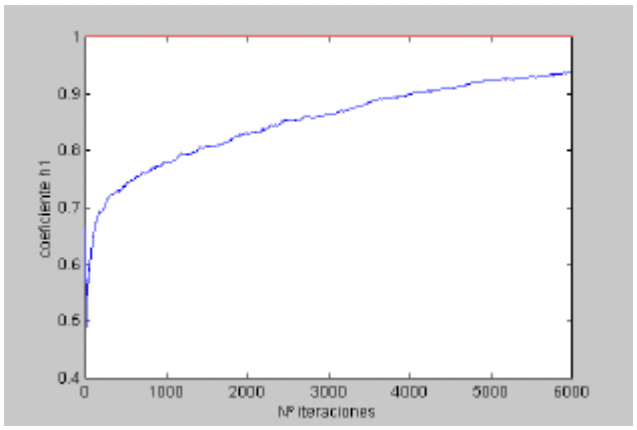



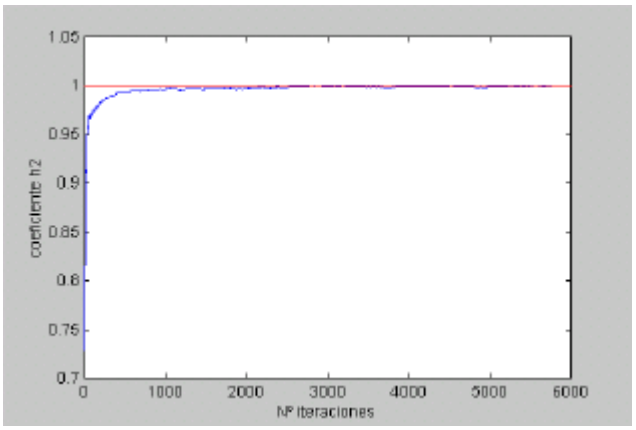

Con el criterio del 3% no se llega a alcanzar la convergencia, aunque se ve claro en las gráficas que con más iteraciones se podría alcanzar. Es decir, al ir bajando el paso de adaptación, la convergencia se hace cada vez más lenta. De esta manera que los valores que se usan para el paso de adaptación en los algoritmos anteriores son demasiado pequeños para este algoritmo RLS.

Ahora se va a considerar un ***proceso de entrada con media 1***. Se empieza con un valor para el **paso de adaptación** igual a **0.5**. Las gráficas son:



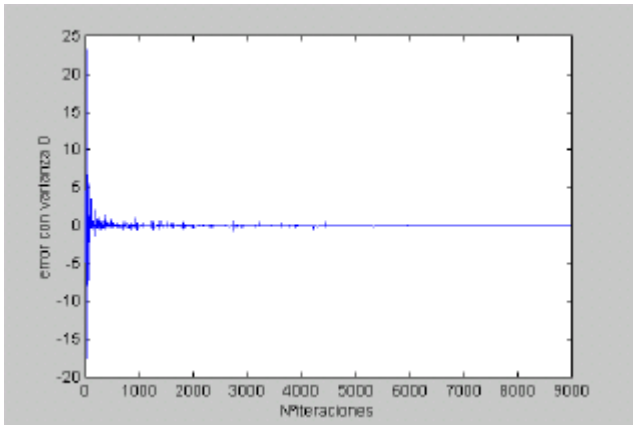



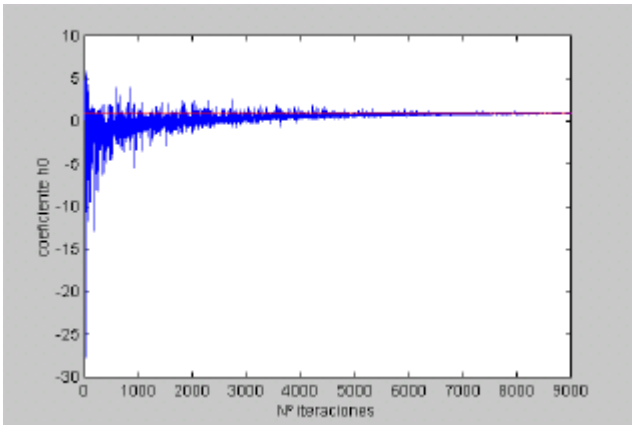



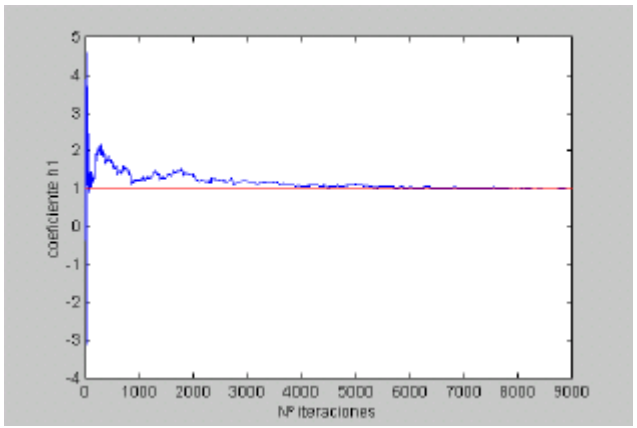

De acuerdo con nuestro criterio del 3%, la convergencia se produce en la iteración 3783, es decir, un poco más lenta que cuando teníamos un proceso de entrada de media 0. Esto es lo lógico y es lo que se viene obteniendo en todos los algoritmos.

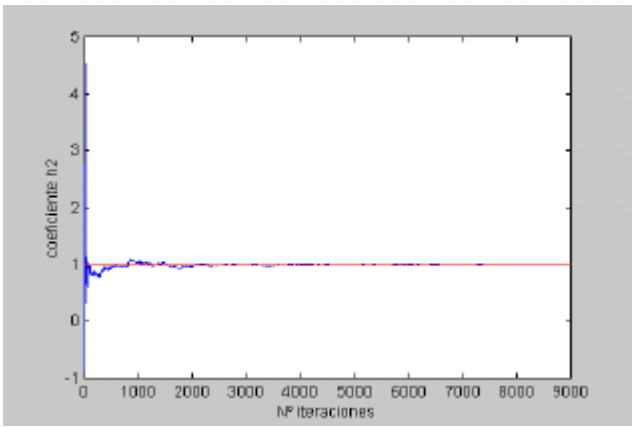
La señal de error parece muy grande pero nuevamente hay que fijarse en el eje de ordenadas. La escala de este eje es bastante pequeña.

Al aumentar la media del proceso de entrada, el paso de adaptación debe reducirse para evitar que el algoritmo diverja. Si se simula con un **paso de adaptación** igual a **1**, se obtiene las siguientes gráficas:



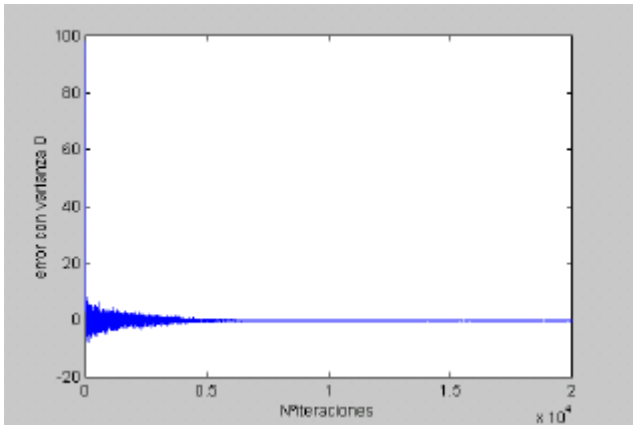



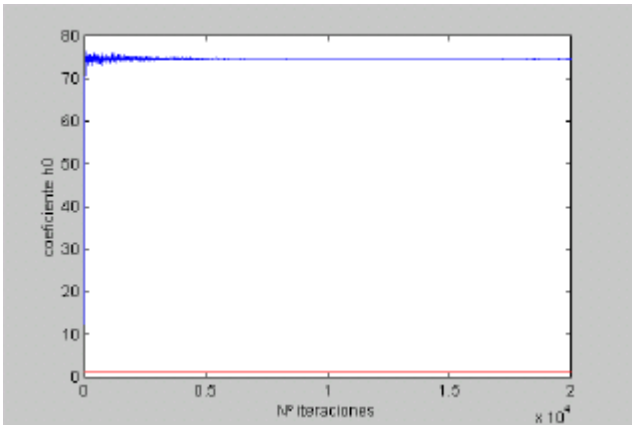



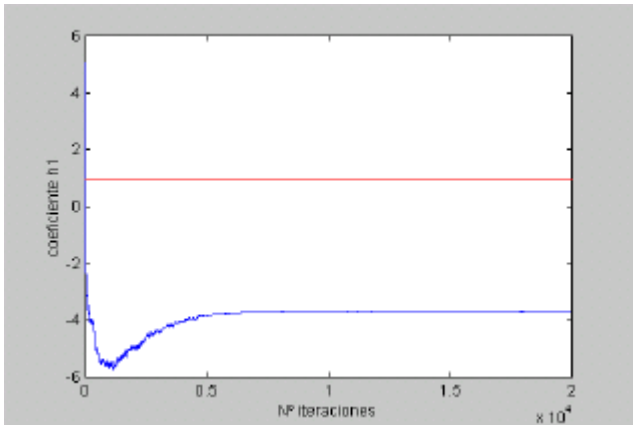

De acuerdo con el criterio del 3%, la convergencia se consigue en la iteración 8222, es decir, es mucho más lenta que con un paso de adaptación igual a 0.5.

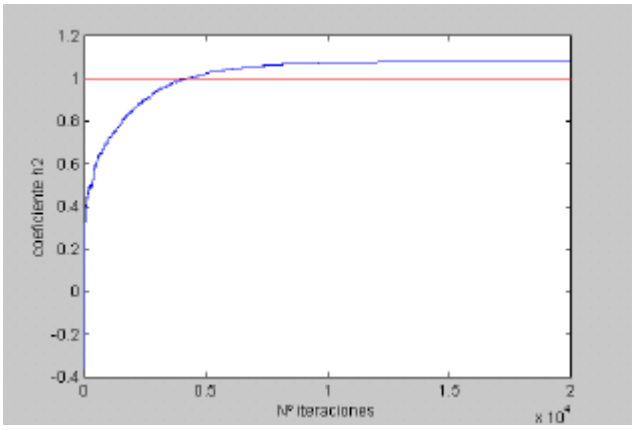
Este hecho se viene manifestando desde el principio de las simulaciones. Al aumentar la media del proceso de entrada y/o el ruido presente en el sistema, el paso de adaptación debe ser más pequeño que cuando la media es cero y no hay ruido para evitar que el algoritmo diverja. Esto es bastante lógico puesto que este parámetro es el que controla la velocidad de convergencia. Por eso si el sistema se complica (ruido y/o media distinta de cero) se deben dar los pasos con más cuidado (reducir el parámetro en cuestión) para evitar “descarrilar” (divergir).

Por último, vamos a considerar un ***proceso de entrada de media 10***. El **paso de adaptación** debe ser algo más pequeño para evitar que el algoritmo diverja debido a la media del proceso de entrada. Así, se toma **m=0.1** y se obtiene:

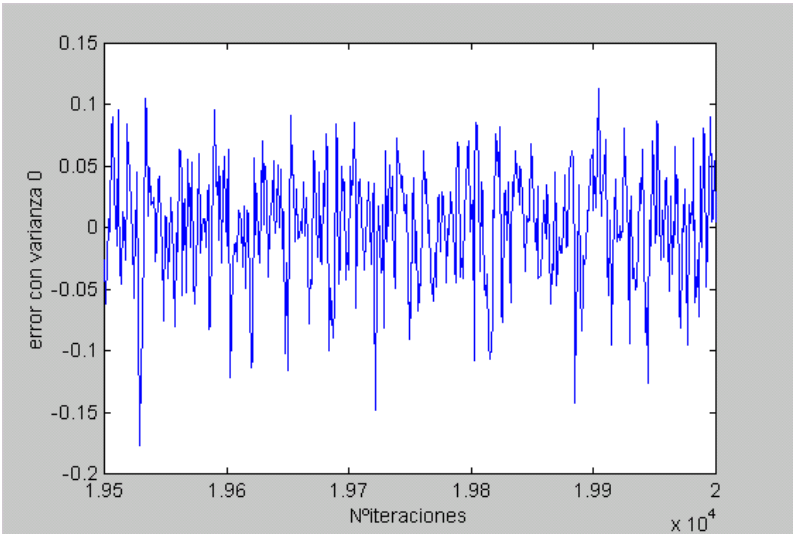



Con el criterio del 3% no se consigue la convergencia en estas 20000 iteraciones. Los coeficientes $h_0$ y $h_1$ se adaptan bastante mal. Sin embargo, el coeficiente $h_2$ consigue una adaptación bastante buena y es el responsable de que la señal de error sea relativamente pequeña en las últimas iteraciones. De hecho, si representamos sólo las últimas 50 muestras de esta señal:




Se ve en el eje de ordenadas que los valores son relativamente pequeños aunque no lo suficiente como para alcanzar la convergencia de acuerdo con nuestro criterio del 3%.

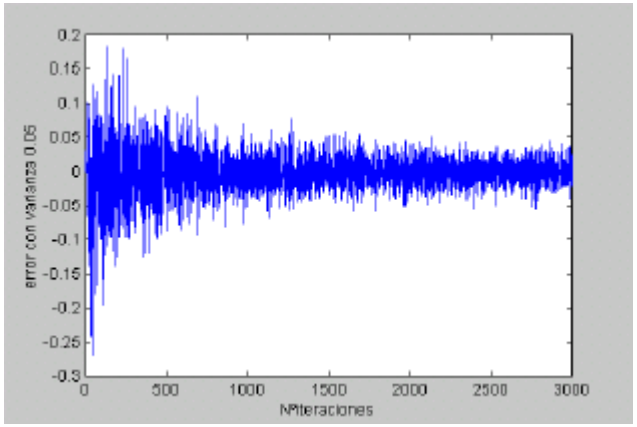
Tal y como se ha visto en otras ocasiones, tanto este algoritmo como los dos anteriores funcionan mejor con los coeficientes cuadráticos que con los lineales. Esto es lógico teniendo en cuenta que el principal problema del que se partía era conseguir trabajar con sistemas no lineales, que, en nuestra situación, es la parte cuadrática.

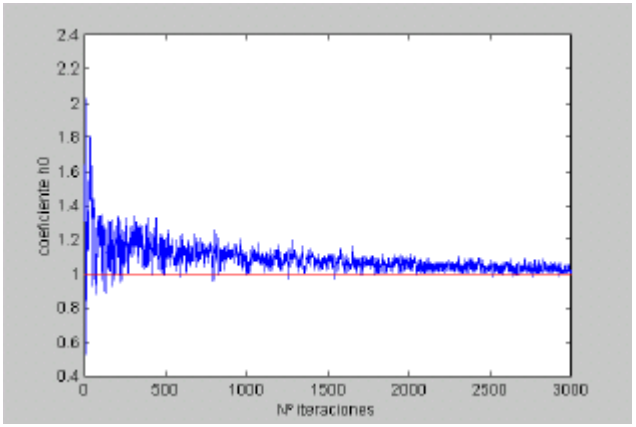
- $\underline{s_w^2 = 0.05}$:

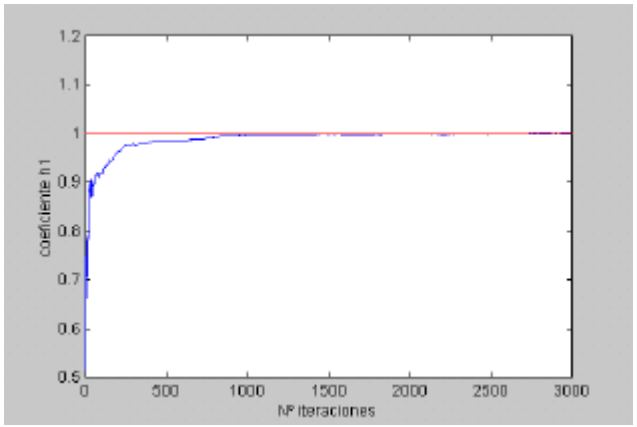
A continuación se estudia la presencia de un ruido pequeño con varianza igual a 0.05.

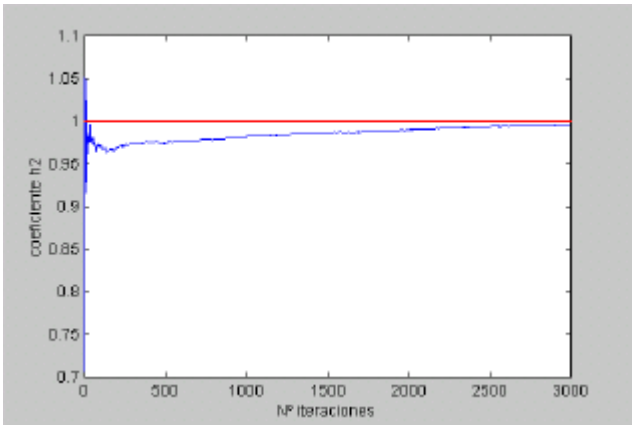
Al aumentar el ruido, hay que flexibilizar el criterio de convergencia. Al igual que en los anteriores algoritmos, se considerarán dos criterios: el del 10% (convergencia si la señal de error se mantiene por debajo de $\pm 0.10$); y el del 15% (convergencia si la señal de error se mantiene por debajo de $\pm 0.15$).

Como siempre, se empieza por considerar ***un proceso de entrada de media cero***. Se considera un **paso de adaptación** igual a **0.5**. Las gráficas son las siguientes:



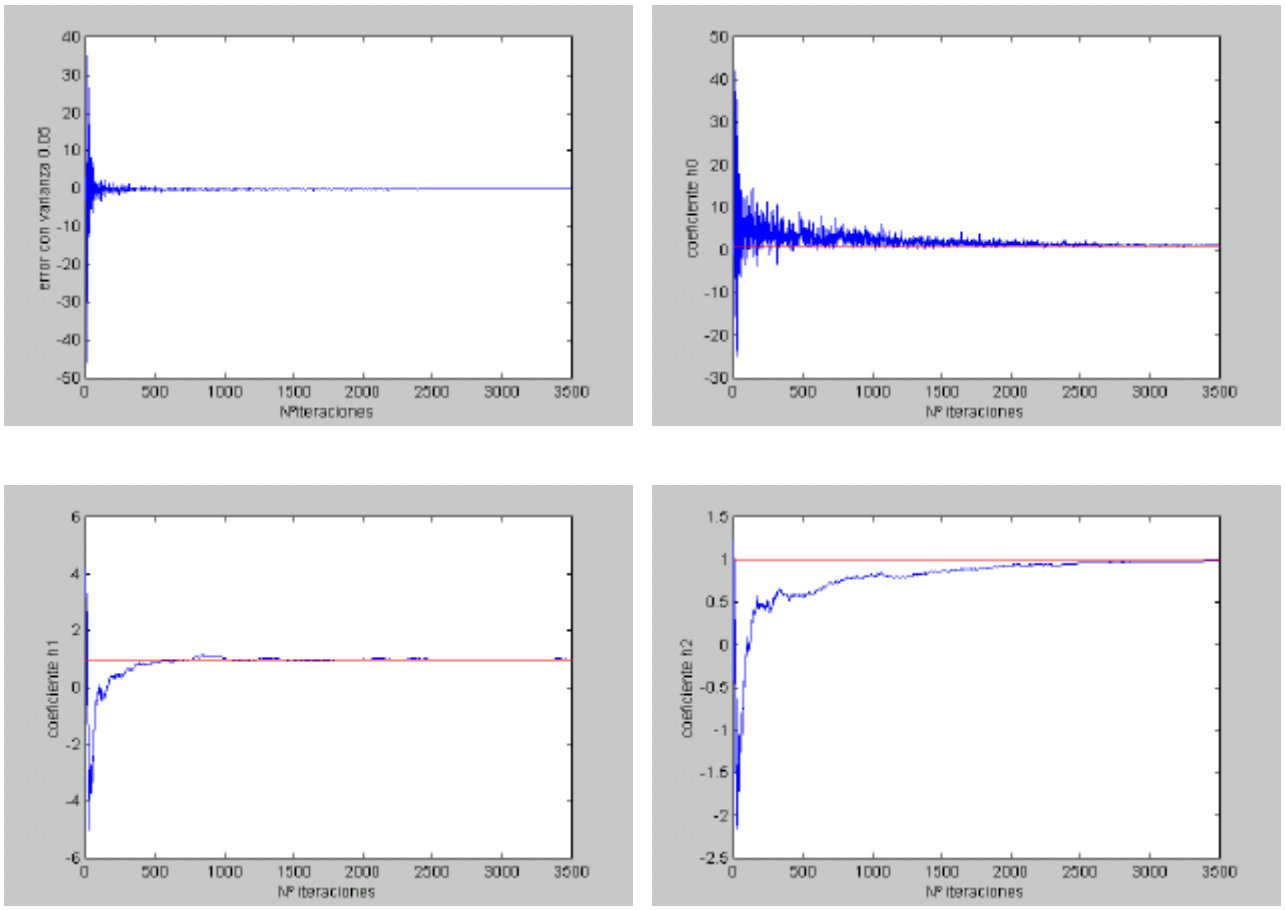



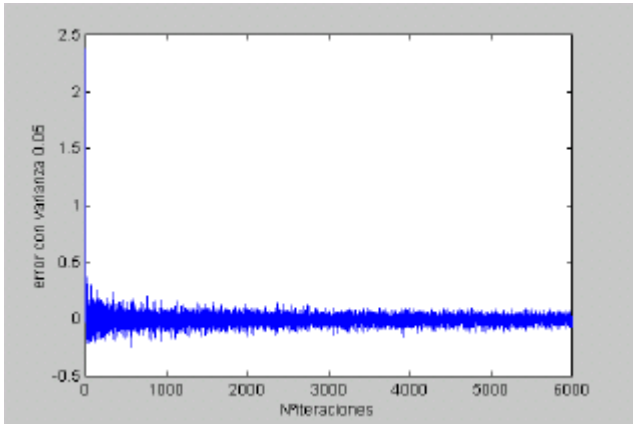
La convergencia se produce en las iteraciones 690 y 255 según los criterios del 10% y del 15%, respectivamente. Bajo el criterio del 10%, la convergencia es mucho más rápida que en los dos algoritmos anteriores. Esto quiere decir que la presencia del ruido no afecta tanto a este algoritmo como les ocurría a los otros dos.

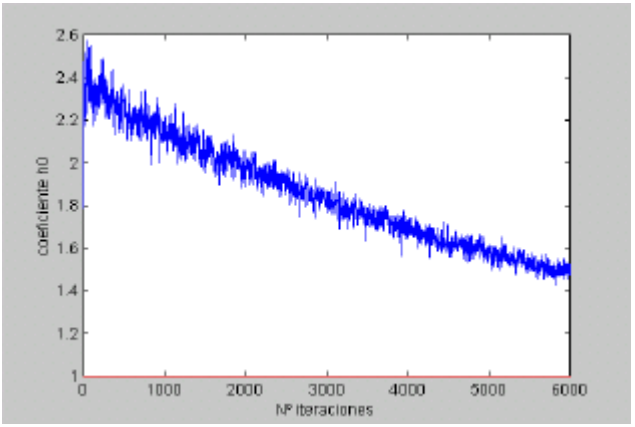
Vamos a ver que ocurre al simular con un **paso de adaptación** igual a **1**. Los resultados son:

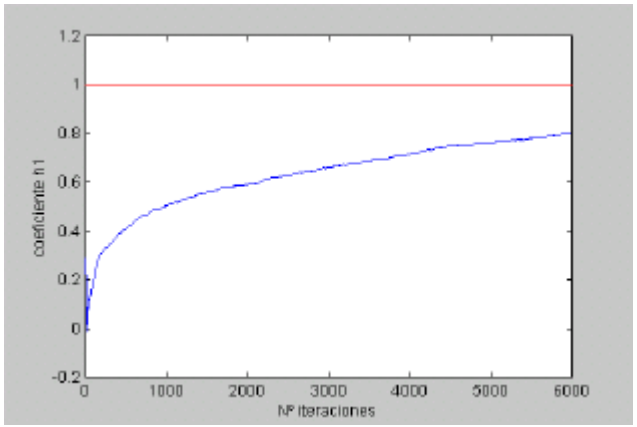


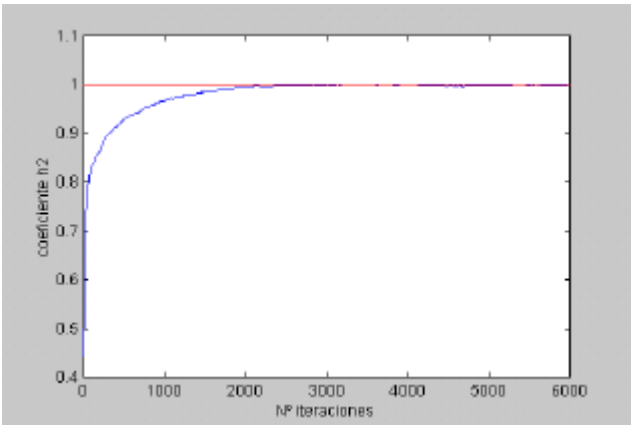

La convergencia se produce en la iteración 2383 para ambos criterios. Como era de esperar, al aumentar el paso de adaptación, la convergencia se ha hecho más lenta al igual que ocurrió cuando no había ruido.

Se considera ahora la situación en la que la ***media del proceso de entrada es igual a 1***. Para empezar se simula con un **paso de adaptación** igual a **0.3**. Se toma un valor un poco más pequeño debido a la media del proceso de entrada. Las gráficas son:



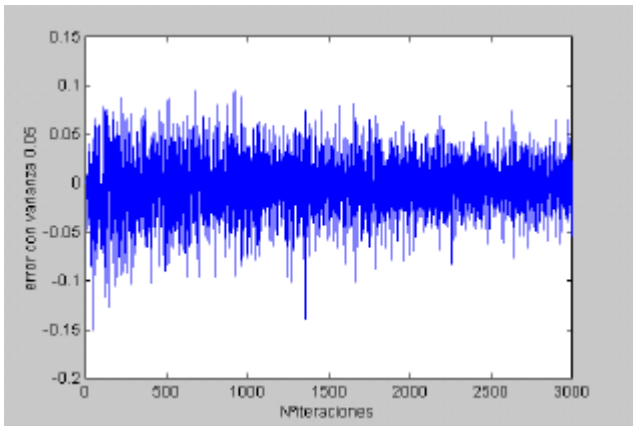



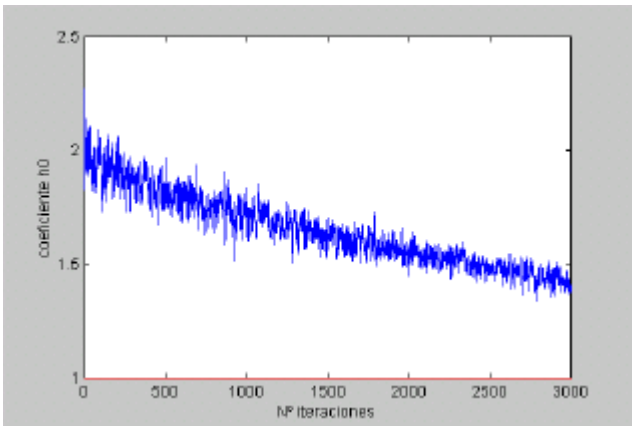



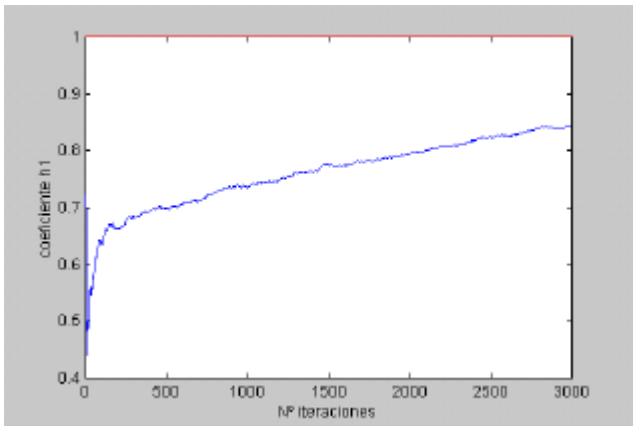



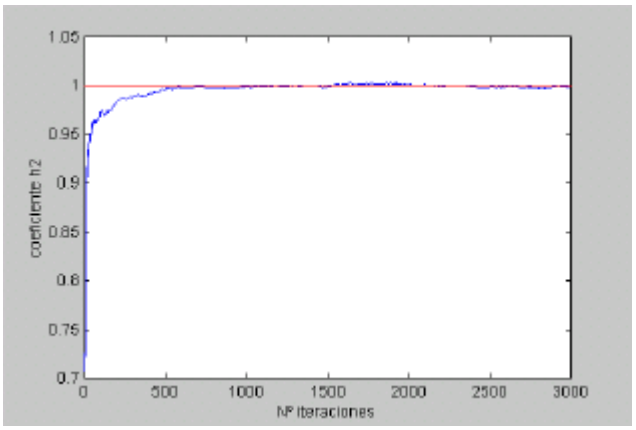

La convergencia se produce en las iteraciones 5218 y 3933 de acuerdo con los criterios del 10% y del 15%, respectivamente. Hay que destacar de nuevo que, según el criterio más exacto (10%), la convergencia es mucho más rápida que en los dos algoritmos anteriores en estas mismas circunstancias lo que continua verificando que el algoritmo RLS es más inmune al ruido.

Subimos un poco el **paso de adaptación** porque se ve menos afectado por el ruido que en los algoritmos anteriores. Así, se simula ahora con **m=0.5**. Se sigue considerando un proceso de entrada de media 1. Los resultados son:



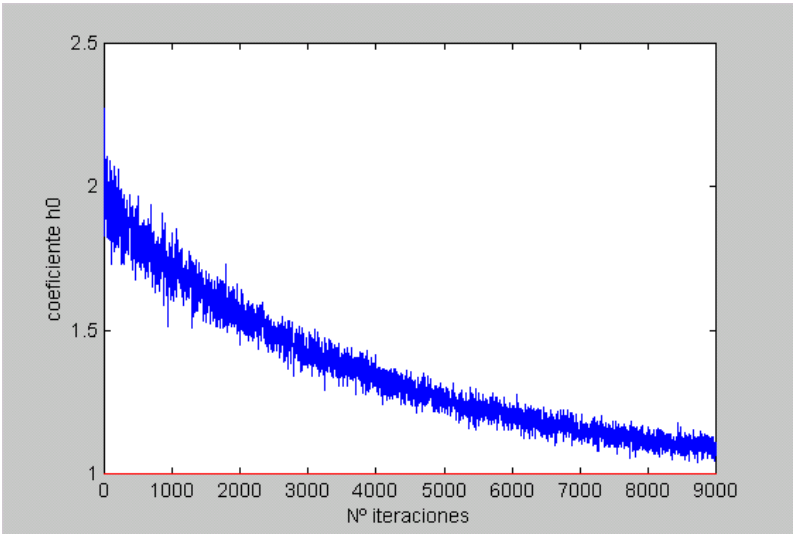

La convergencia se produce en las iteraciones 1660 y 39 según los criterios del 10% y del 15%. Hay que comentar varias cosas.

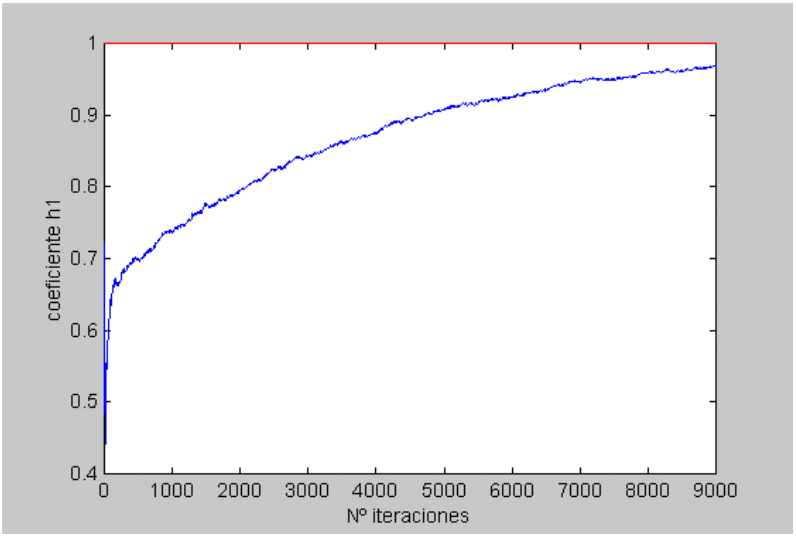
Primero, esta convergencia es más rápida que la de la situación anterior con el paso de adaptación igual a 0.3. esto se debe a lo que ya se comentó respecto a que el paso de adaptación suele estar en torno a la unidad. Al subir la media del proceso de entrada se debe bajar un poco pero no tanto como en el caso anterior.

Segundo, la convergencia también es mucho más rápida que la que se obtuvo en estas circunstancias con los otros algoritmos. La desventaja es que el número de operaciones necesarias en este algoritmo es muy elevado.

Por último comentar que nuevamente es el coeficiente $h_2$ el que converge a mayor velocidad. De hecho, los otros dos coeficientes no llegan a converger totalmente con las 3000 iteraciones simuladas. Se han realizado 9000 iteraciones para demostrar que ambos coeficientes acaban convergiendo a sus

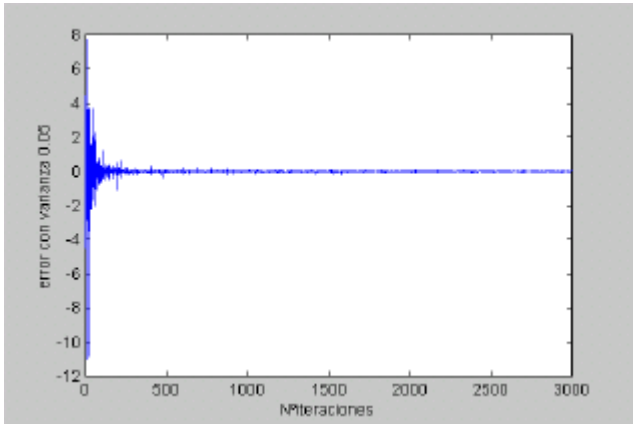
respectivos valores finales. Las gráficas correspondientes a ambos coeficientes son:

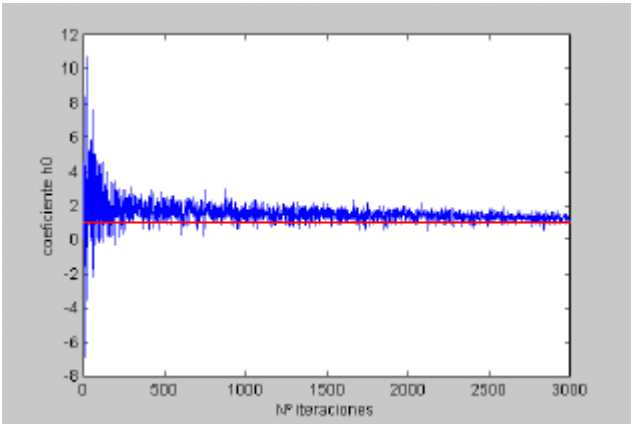


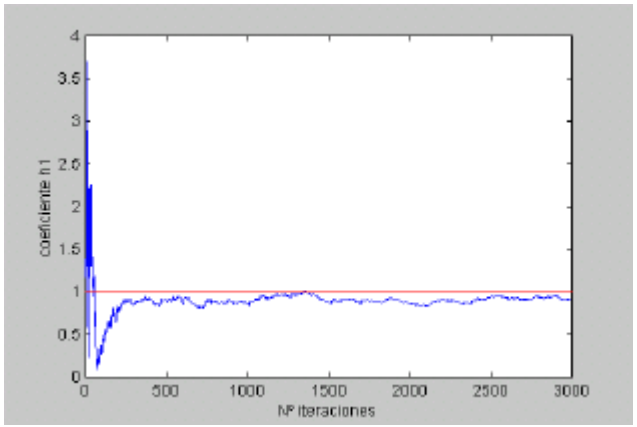



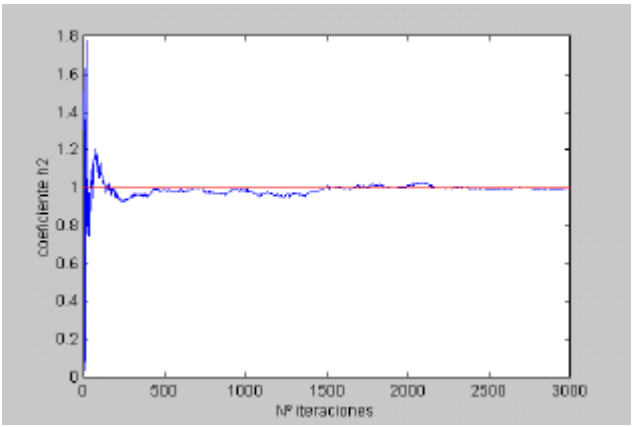

La convergencia para estos coeficientes es más lenta pero se alcanza. No obstante, el coeficiente más importante es el cuadrático, es decir, $h_2$.

Siguiendo con un proceso de entrada de media 1, se simula ahora con un **paso de adaptación** igual a **1**. Las gráficas son:



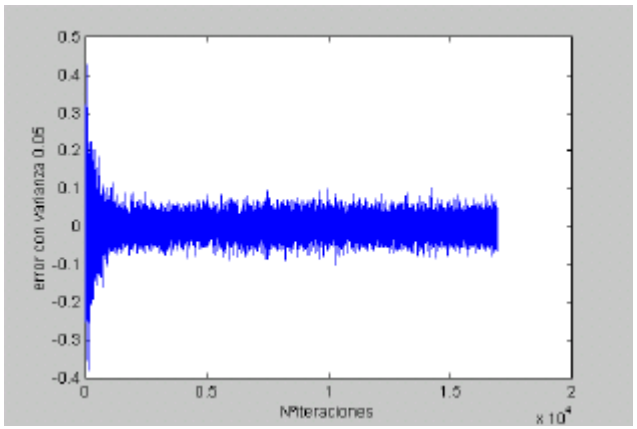



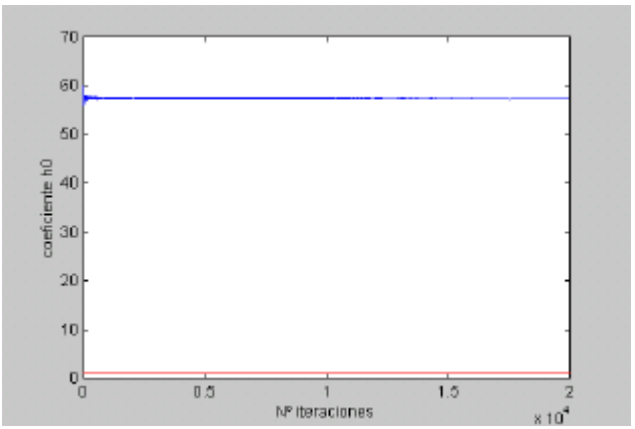



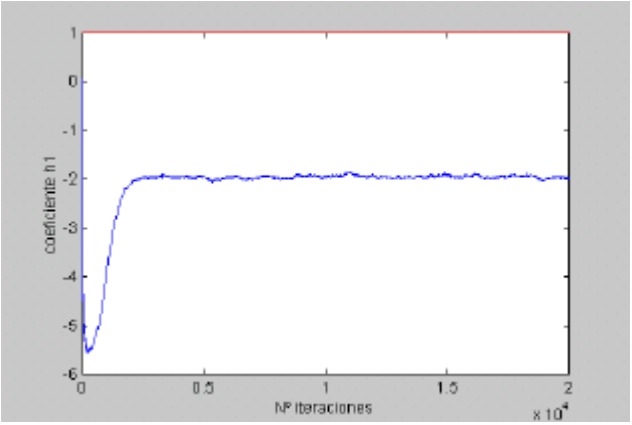



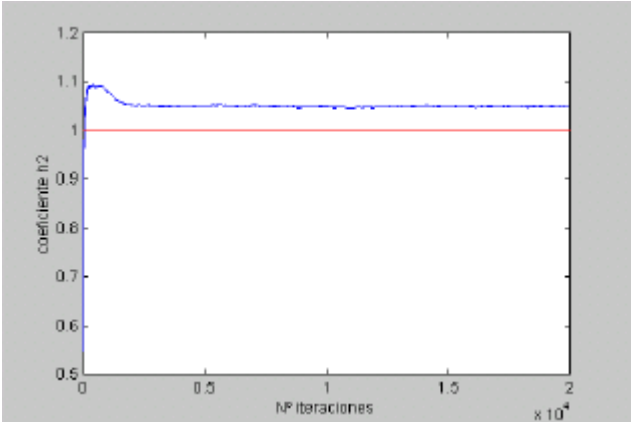

De acuerdo con el criterio del 10%, la convergencia se produce en la iteración 2709 mientras que según el criterio del 15%, ésta se obtiene en la 1522. La convergencia se ralentiza respecto a la situación anterior.

Finalmente, se considera ahora ***un proceso de entrada de media 10***. Se toma un **paso de adaptación** igual a **0.5**. Las gráficas son:






Con el criterio del 10% no se llega a alcanzar la convergencia. Sin embargo, con el del 15%, ésta se alcanza en la iteración 687. Este último resultado es muy positivo puesto que con los algoritmos anteriores no se conseguía una buena convergencia en esta situación (proceso de entrada con media 10). Ahora no se consigue la convergencia más exacta pero sí se consigue una convergencia rápida si se suavizan ligeramente las condiciones.

Resumiendo un poco, se puede decir que, en condiciones desfavorables, este algoritmo funciona mejor en cuanto a velocidad y precisión de la convergencia que los dos anteriores. Estas condiciones desfavorables hacen referencia tanto a la presencia de ruido como a la existencia de un proceso de entrada con media distinta de cero.

Sin embargo, no se puede olvidar que la carga computacional es mucho más elevada en este algoritmo que en los dos anteriores.

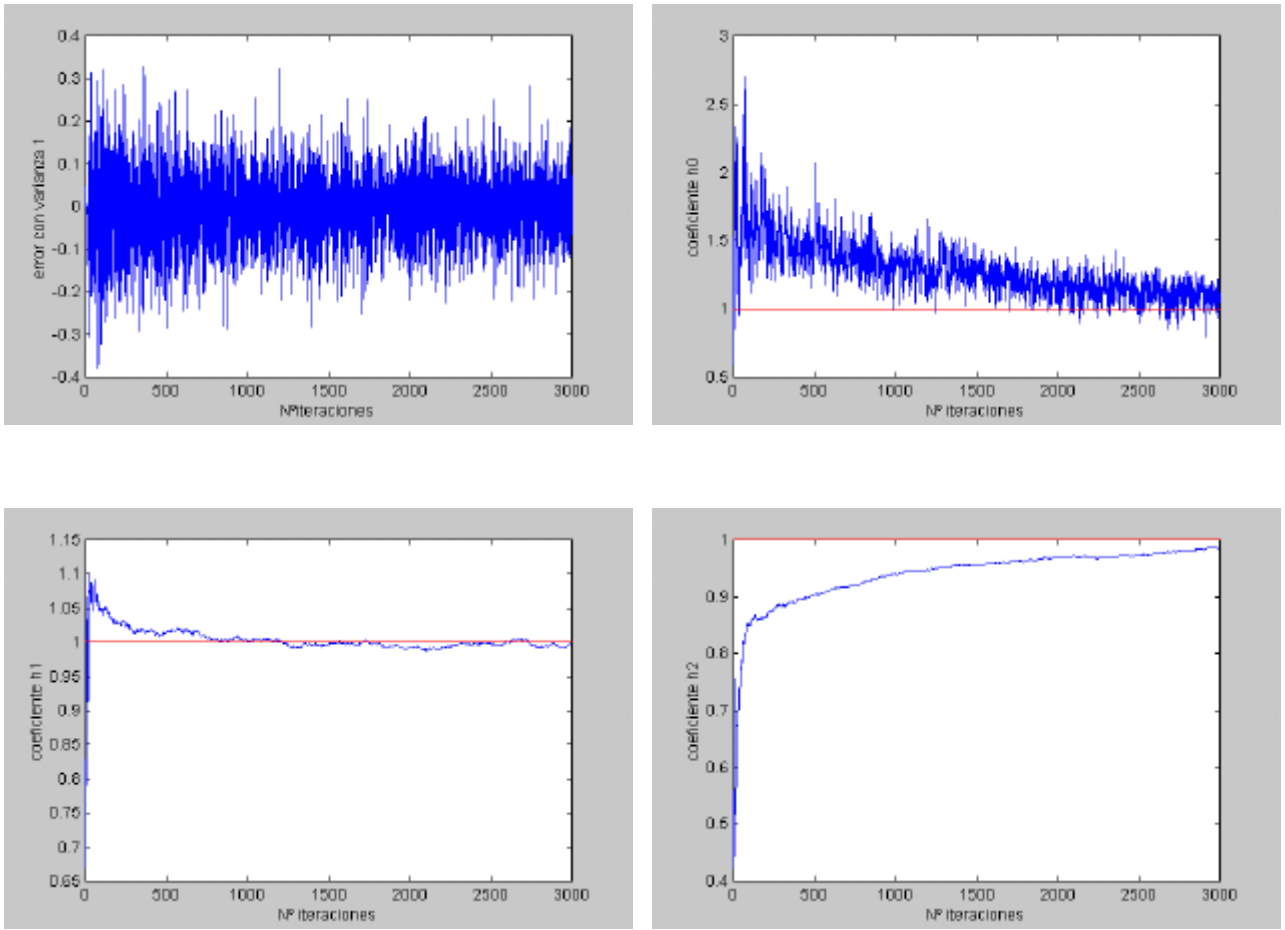
Por tanto, en cada situación habrá que valorar lo que se tiene y lo que se necesita obtener.

- $\underline{s_w^2 = 1}$:

A continuación se va a analizar la presencia de un ruido grande (varianza 1). Al igual que se hizo en los algoritmos anteriores, se van a utilizar dos criterios para evaluar la convergencia: los criterios del 50% y del 55%.

En principio cabe esperar un mejor comportamiento de este algoritmo puesto que antes se ha visto que es más inmune al ruido.

Se comienza como siempre considerando ***un proceso de entrada de media cero***. Para esta primera simulación, se toma un **paso de adaptación** igual a **0.5**. Las gráficas son:




Lo primero que llama la atención es que la convergencia es muy buena: los valores de la señal de error son pequeños y los coeficientes alcanzan sus respectivos valores finales.

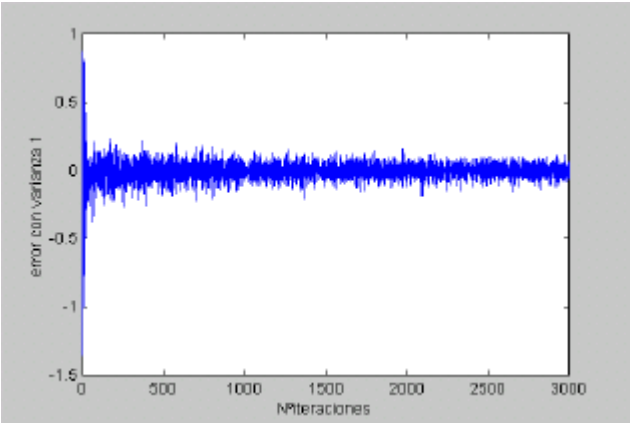
Si nos sometemos al criterio de convergencia que se ha utilizado para los algoritmos anteriores (50% y 55%), se concluye que estos criterios son demasiado flexibles para

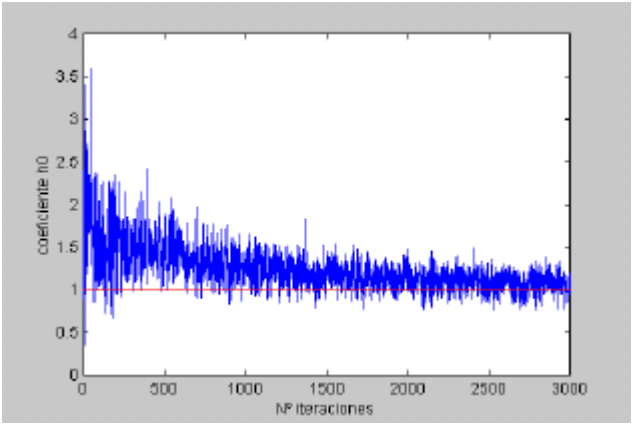
este algoritmo, es decir, la señal de error siempre se mantiene por debajo de los valores estipulados en ambos criterios. Esto vuelve a confirmar el hecho de que el comportamiento de este algoritmo, en cuanto a convergencia, frente al ruido es bastante mejor que el de los otros dos algoritmos.

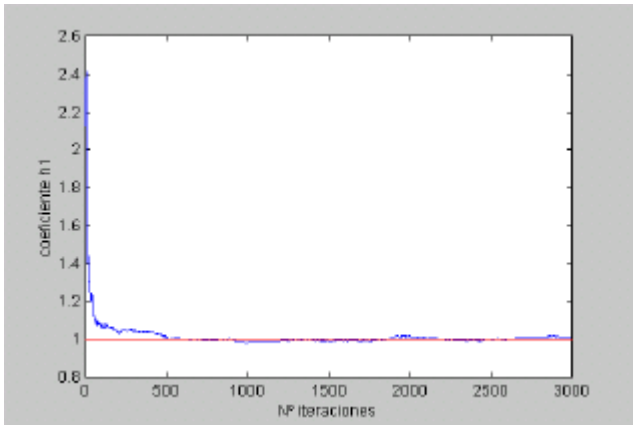
No obstante se han endurecido estos criterios para poder comparar estas simulaciones para diferentes valores del paso de adaptación. Se ha estudiado la convergencia para varios criterios y los resultados se resumirán al final en una tabla. Aquí, nos centraremos en dos de ellos: los criterios del 20% y 25%.

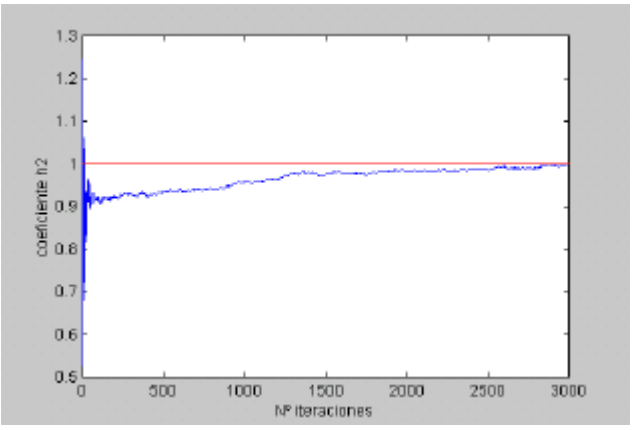
Para esta primera simulación, la convergencia se alcanzó en las iteraciones 2899 y 2735 según los criterios del 20% y 25%, respectivamente.

Vamos a simular ahora con un paso de adaptación intermedio entre 0.5 y 1 para ir encontrando el valor óptimo. En simulaciones anteriores se ha visto que el valor 0.5 era el óptimo. Con valores más pequeños, la convergencia era más lenta y con un valor igual a 1, también. Por los valores que se han obtenido, el óptimo es mayor o igual que 0.5 y menor que 1. Las siguientes gráficas son para un **paso de adaptación** igual a **0.7**:



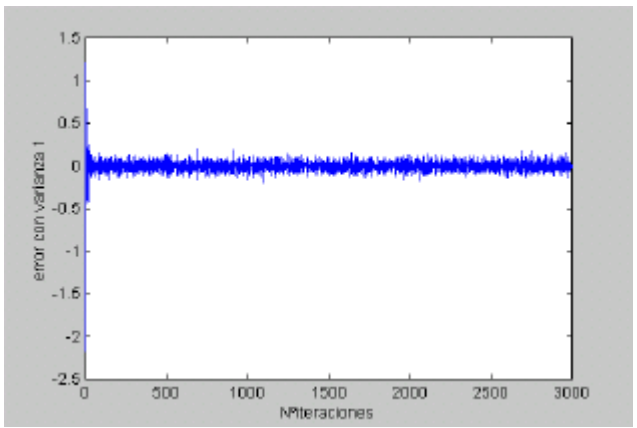


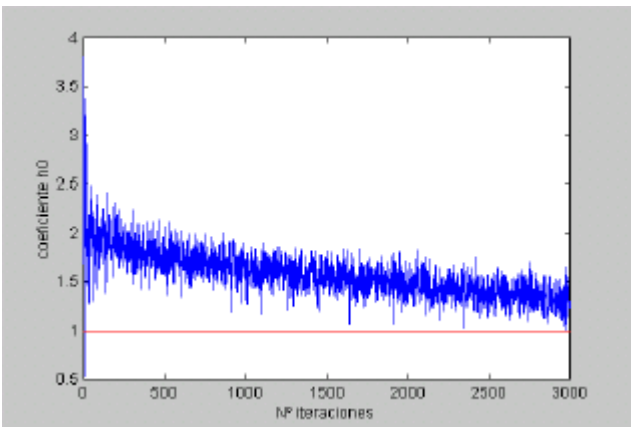



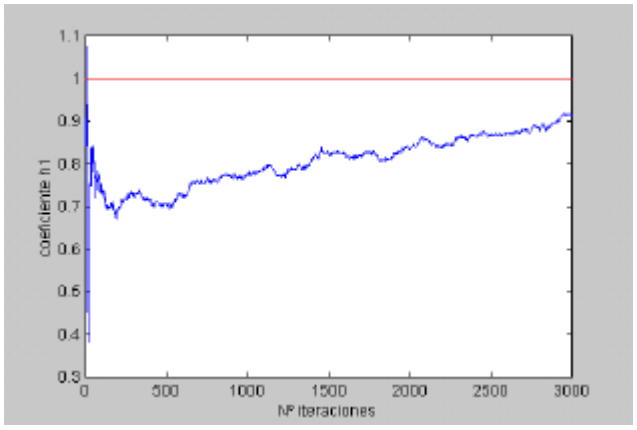

La convergencia se alcanza en las iteraciones 1373 y 395 de acuerdo con los criterios del 20% y del 25% respectivamente. Esto quiere decir que el valor 0.7 es mejor que el de 0.5 puesto que la velocidad de convergencia es superior.

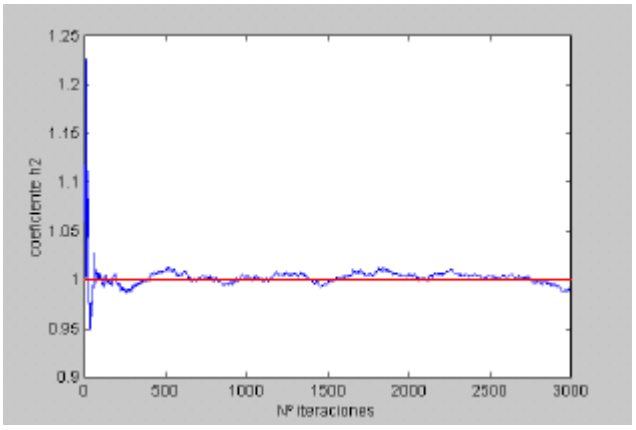
A continuación se considera ***un proceso de entrada de media 1***. Puesto que este algoritmo se comporta bastante bien frente al ruido, lo normal es que se alcance la convergencia con bastante rapidez.

Se toma un valor para el **paso de adaptación** igual a **0.7**. Las gráficas que se obtienen son:



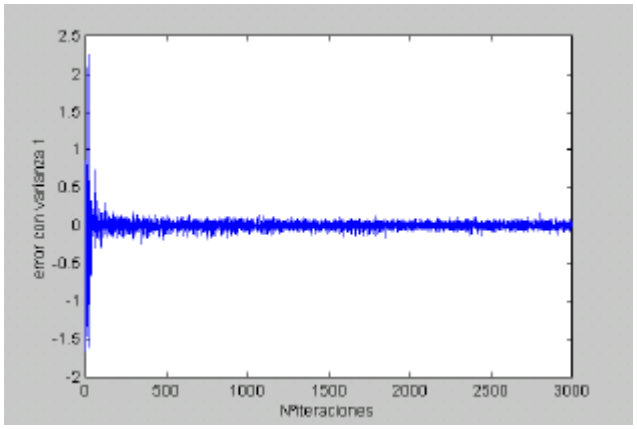


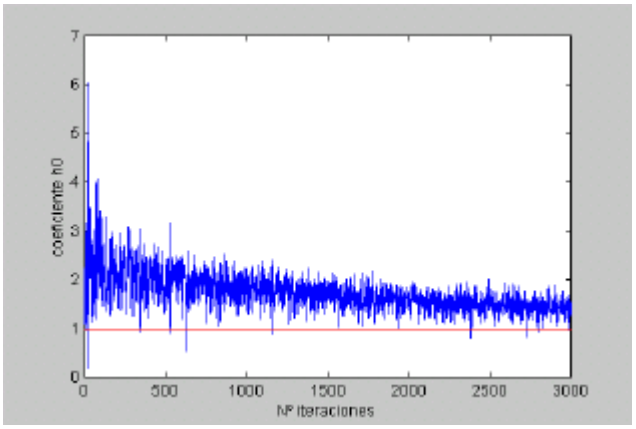



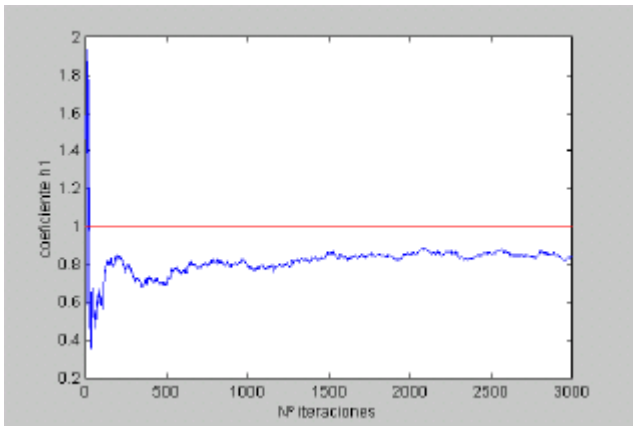

Se obtiene la convergencia en las iteraciones 24 y 22 según los criterios del 20% y 25%. Estos valores son bastante pequeños lo que implica una alta velocidad de convergencia. De hecho esta velocidad es mucho mayor que la que se obtuvo para un proceso de entrada de media cero.

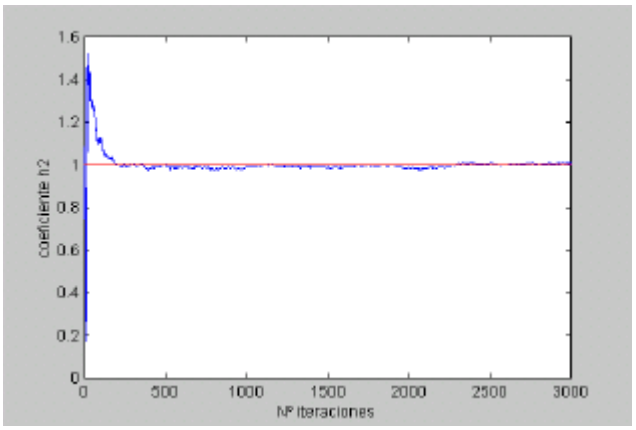
Si se endurece más el criterio de convergencia, esta diferencia cambia de sentido. Se considera un criterio de convergencia del 15%. Ahora, ésta se produce en la iteración 2659 para un proceso de entrada de media cero y en la 2961 para uno de media uno.

Ahora se simula considerando un **paso de adaptación** igual a **0.8**. De esta manera se puede precisar mejor cual es el valor óptimo para μ. Las gráficas son:



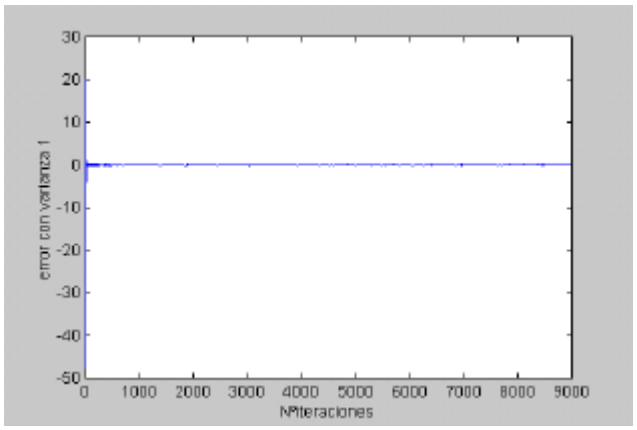



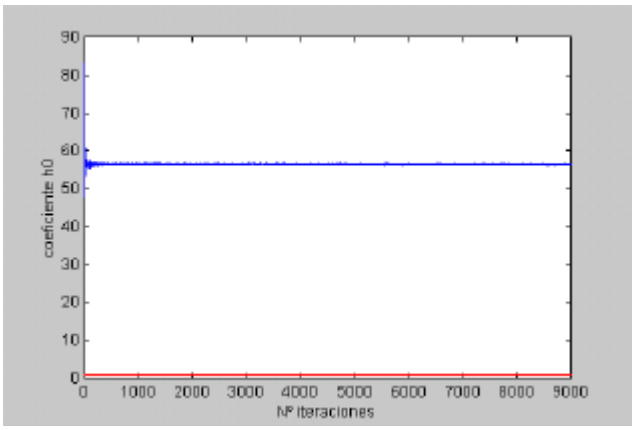


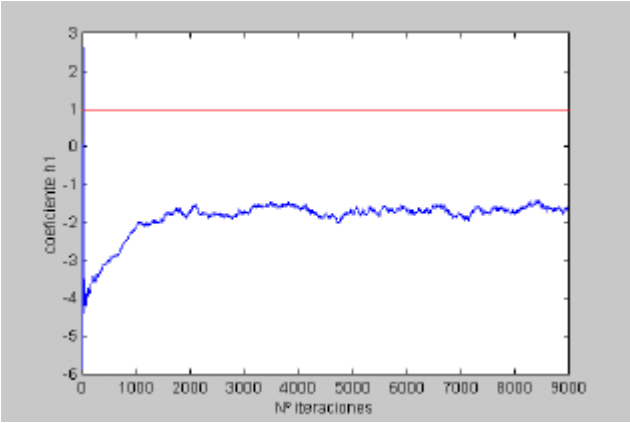



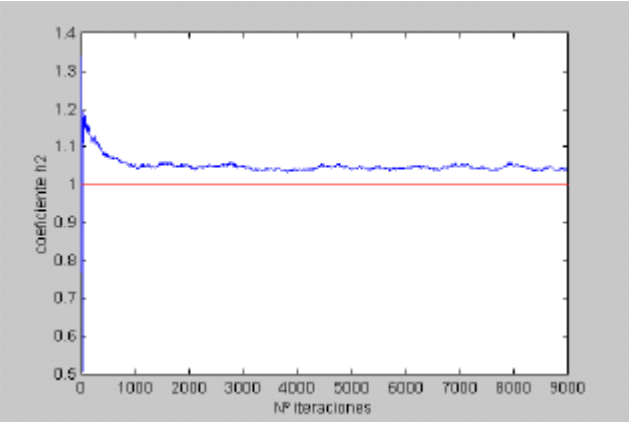

De acuerdo con los criterios del 20% y del 25%, la convergencia se produce en las iteraciones 339 121 respectivamente. Esto implica una convergencia un poco más lenta que para el caso anterior, en el que μ se tomó igual a 0.7. Luego se puede concluir que el valor óptimo para el paso de adaptación es 0.7.

Finalmente, se realiza una simulación para ***un proceso de entrada de media 10***. Se elige **0.7** como el valor para el **paso de adaptación**. Los resultados son:






La convergencia se alcanza en las iteraciones 335 y 246 según los criterios del 20% y 25%. Este resultado es muy bueno porque en esta situación siempre había problemas con la convergencia para los anteriores algoritmos.

No obstante, hay que destacar que la convergencia se produce gracias a la buena adaptación del coeficiente cuadrático ($h_2$). Los otros dos coeficientes se adaptan bastante mal. Esto verifica una vez más el hecho de que estos algoritmos están pensados para trabajar con la parte no lineal, que en este caso es la cuadrática.

Para terminar, comentar lo que ya se ha visto en las simulaciones. El algoritmo RLS se comporta mejor en cuanto a velocidad de convergencia cuando existe ruido que los otros dos algoritmos. También se ve menos afectado por la presencia de un proceso de entrada de media distinta de cero.

La gran desventaja radica en el elevado número de operaciones que son necesarias.

Vamos a resumir todos los valores en una tabla:

| Nºiteraciones | $s_w^2$ | media | m | criterio | convergencia |
|---|---|---|---|---|---|
| 3000 | 0 | 0 | **0.5** | 3% | **2364** |
| 3000 | 0 | 0 | 1 | 3% | 2618 |
| 6000 | 0 | 0 | 1.5 | 3% | X |
| 6000 | 0 | 0 | 0.1 | 3% | X |
| 6000 | 0 | 1 | **0.5** | 3% | **3783** |
| 9000 | 0 | 1 | 1 | 3% | 8222 |
| 20000 | 0 | 10 | 0.1 | 3% | X |
| 3000 | 0.05 | 0 | **0.5** | 10%/15% | **690/255** |
| 6000 | 0.05 | 0 | 1 | 10%/15% | 2383/2383 |
| 6000 | 0.05 | 1 | 0.3 | 10%/15% | 5218/3933 |
| 3000 | 0.05 | 1 | **0.5** | 10%/15% | **1660/39** |
| 3000 | 0.05 | 1 | 1 | 10%/15% | 2709/1522 |
| 20000 | 0.05 | 10 | 0.5 | 10%/15% | X/687 |
| 3000 | 1 | 0 | 0.5 | 10%/15%<br>20%/25%<br>30%/35% | 2993/2993<br>2899/2735<br>1201/82 |
| 3000 | 1 | 0 | **0.7** | 10%/15%<br>20%/25%<br>30%/35% | **2969/2659**<br>**1373/395**<br>**74/53** |
| 3000 | 1 | 1 | **0.7** | 10%/15%<br>20%/25%<br>30%/35% | 2978/2961<br>**24/22**<br>**19/19** |
| 3000 | 1 | 1 | **0.8** | 10%/15%<br>20%/25%<br>30%/35% | **2908/2805**<br>339/121<br>66/59 |
| 9000 | 1 | 10 | 0.7 | 10%/15%<br>20%/25%<br>30%/35% | 8971/8471<br>335/246<br>98/30 |

Resultados con el algoritmo RLS

Antes se dijo que el valor óptimo del paso de adaptación era 0.7. Esto no es del todo cierto tal y como se muestra en la tabla. Depende del criterio de convergencia que se tome. Por ejemplo, para el caso de varianza de ruido y media del proceso de entrada iguales ambas a 1, hay dos simulaciones con pasos de adaptación iguales a 0.7 y 0.8. Pues bien, de acuerdo con los criterios del 10% y del 15%, el valor óptimo es 0.8 mientras que según los otros criterios más flexibles, el óptimo es 0.7. Dependiendo del grado de convergencia que se quiera, habrá que utilizar un valor u otro para el paso de adaptación.

## 5.2.- Ejemplo 2

Se trata de identificar el siguiente sistema por LMS, LMS normalizado y RLS Volterra:

$$y(n) = -0.78x(n) - 1.48x(n-1) + 1.39x(n-2) + 0.04x(n-3) +$$

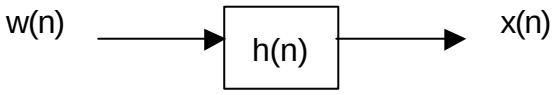
$$+ 0.54x^2(n) + 3.72x(n)x(n-1) + 1.86x(n)x(n-2) - 0.76x(n)x(n-3) -$$

$$- 1.62x^2(n-1) + 1.41x^2(n-2) - 1.52x(n-2)x(n-3) - 0.13x^2(n-3)$$

La señal x(n) se obtiene pasando un ruido, w(n), gausiano de varianza 0.05 y media 0 por el siguiente filtro:

$$h(n) = 0.9045\,\delta(n) + \delta(n-1) + 0.9045\,\delta(n-2)$$

Es decir:




En este ejemplo se va a estudiar el comportamiento de un sistema bastante más complejo que el del ejemplo anterior. Aquí se analizará la convergencia teniendo en cuenta distintos valores del parámetro μ.

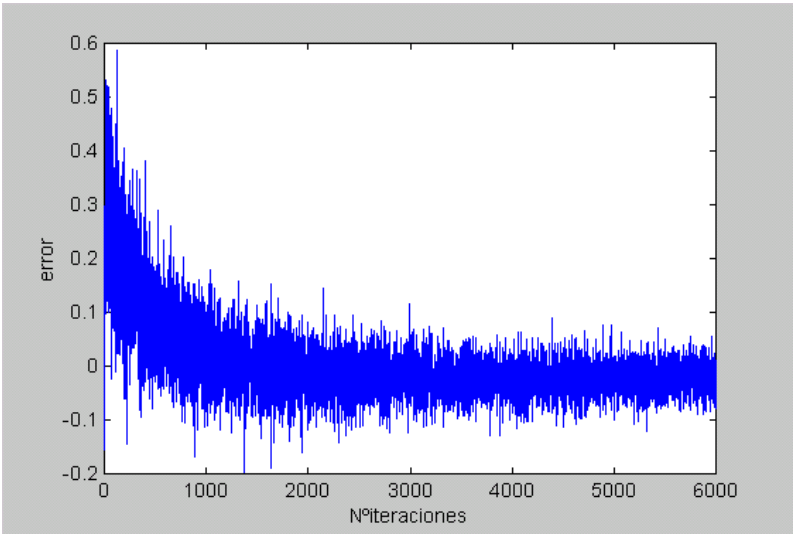
Las gráficas que se van a representar en cada caso son: el vector de error, es decir, la diferencia entre la señal deseada y la respuesta del sistema; y varias gráficas con la convergencia de los distintos coeficientes.

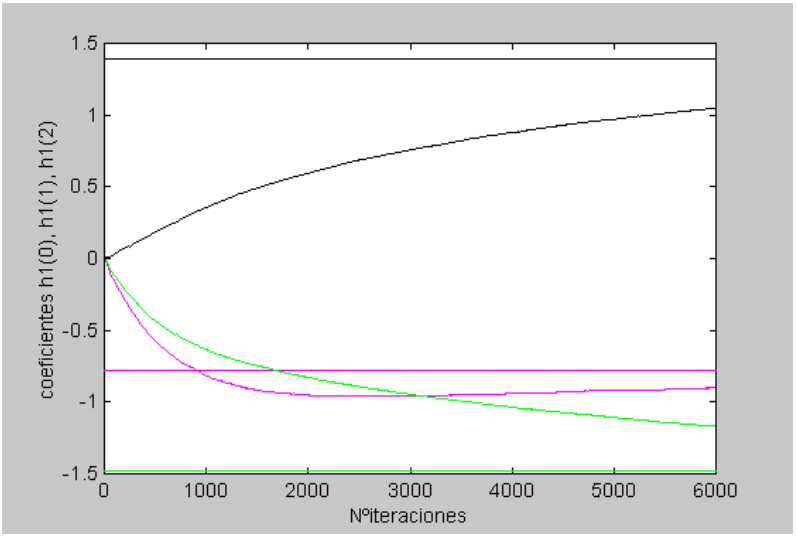
- **LMS:**

Se van a considerar varios valores del paso de adaptación para poder analizar mejor la convergencia.

Los coeficientes se representan de tres en tres con distintos colores para poderlos distinguir.

Para empezar se toma un valor de **m=0.01**. Las gráficas son:

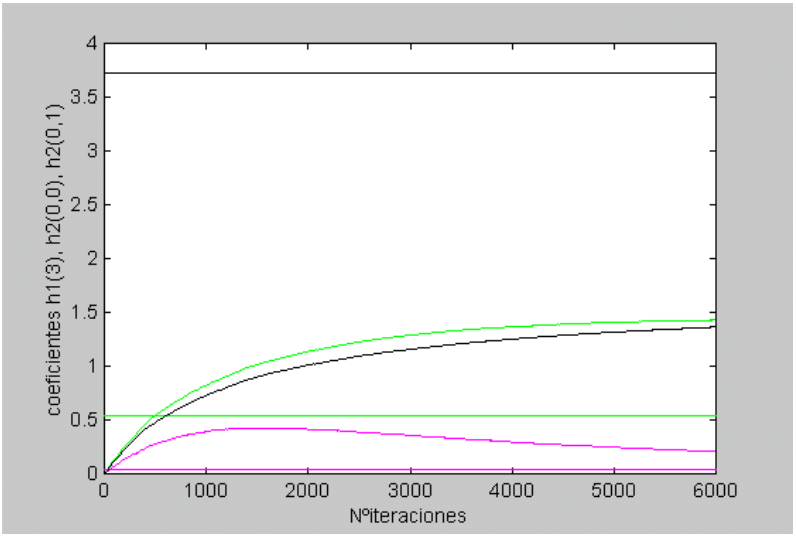



$h_1(0)$ y $\hat{h}_1(0)$

$h_1(1)$ y $\hat{h}_1(1)$

$h_1(2)$ y $\hat{h}_1(2)$




$h_1(3)$ y $\hat{h}_1(3)$

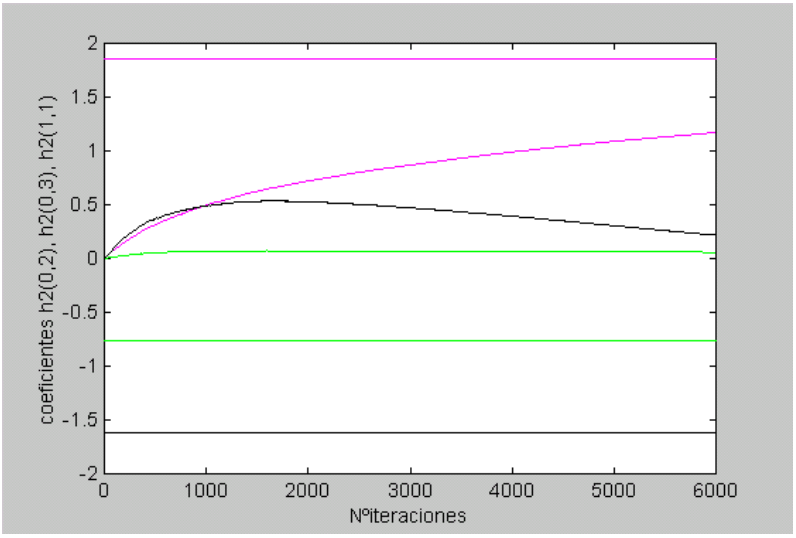
$h_2(0,0)$ y $\hat{h}_2(0,0)$

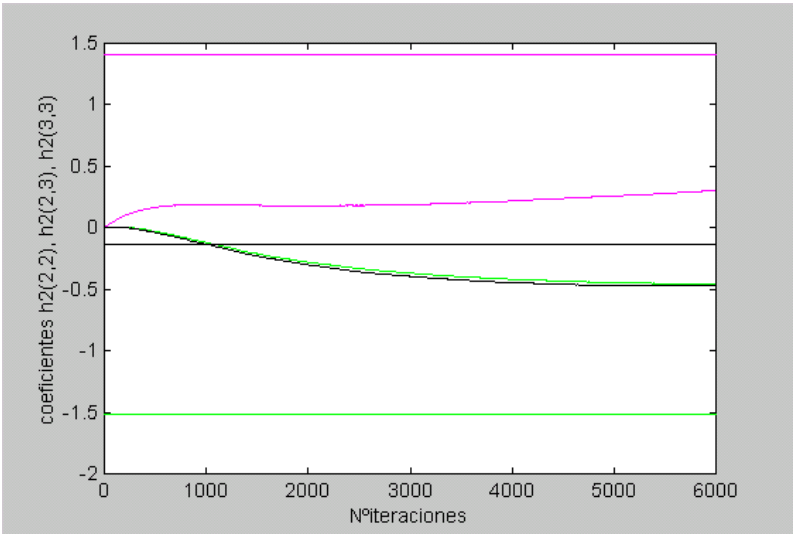
$h_2(0,1)$ y $\hat{h}_2(0,1)$

$h_2(0,2)$ y $\hat{h}_2(0,2)$

$h_2(0,3)$ y $\hat{h}_2(0,3)$

$h_2(1,1)$ y $\hat{h}_2(1,1)$




$h_2(2,2)$ y $\hat{h}_2(2,2)$

$h_2(2,3)$ y $\hat{h}_2(2,3)$

$h_2(3,3)$ y $\hat{h}_2(3,3)$

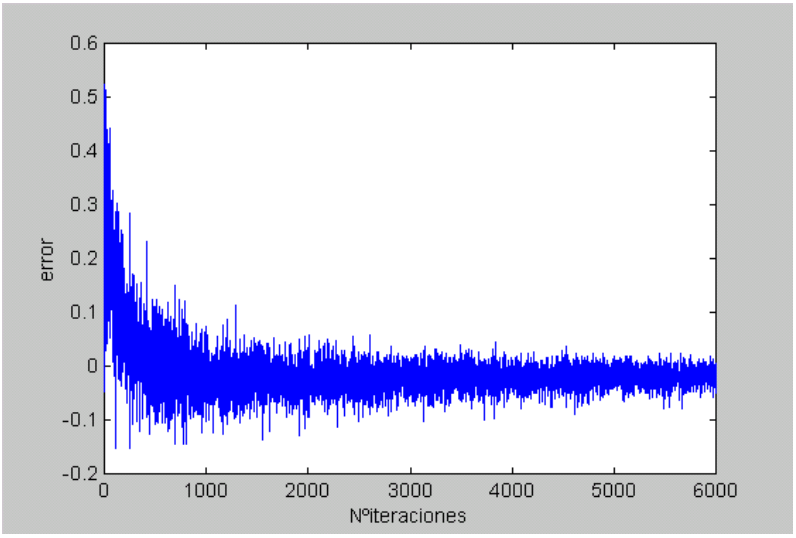
Se han considerado tres criterios distintos de convergencia para tener un estudio más completo. Al final del algoritmo, se mostrarán todos los resultados en una tabla para verlos mejor.

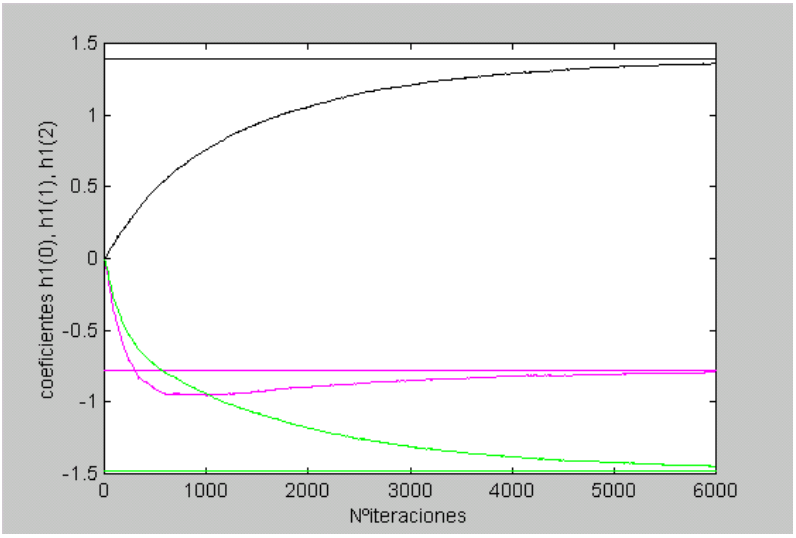
Los criterios de convergencia son análogos a los del ejemplo anterior. Así, se considera convergencia cuando la señal de error se mantiene por debajo de un determinado valor: $\pm 0.03$ (criterio del 3%), $\pm 0.1$ (criterio del 10%) y $\pm 0.15$ (criterio del 15%).

Para este caso, la convergencia se produce en las iteraciones 5326 y 1937 de acuerdo con los criterios del 10% y del 15% respectivamente. El criterio del 3% es demasiado exigente para alcanzar la convergencia con 6000 iteraciones.

Los primeros coeficientes (lineales) son los que mejor se adaptan con este valor del paso de adaptación.

Consideramos ahora un **paso de adaptación** igual a **0.03**. Las gráficas son:






— $h_1(0)$ y $\hat{h}_1(0)$

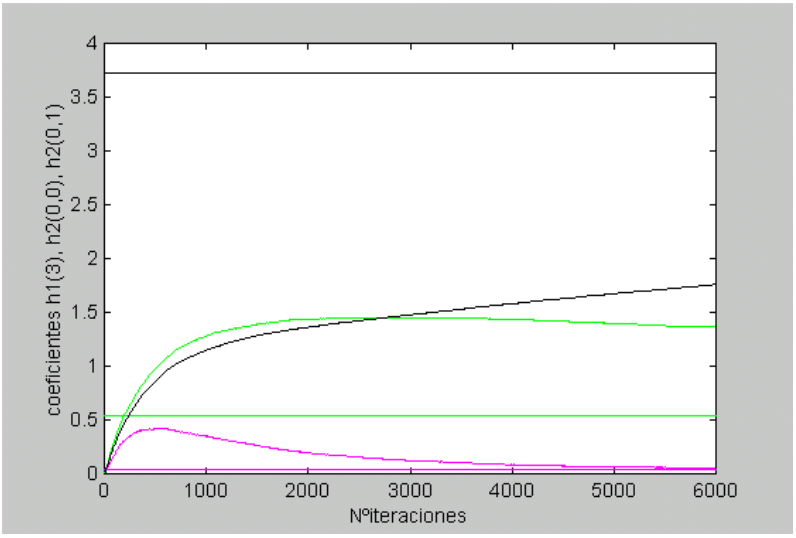
— $h_1(1)$ y $\hat{h}_1(1)$

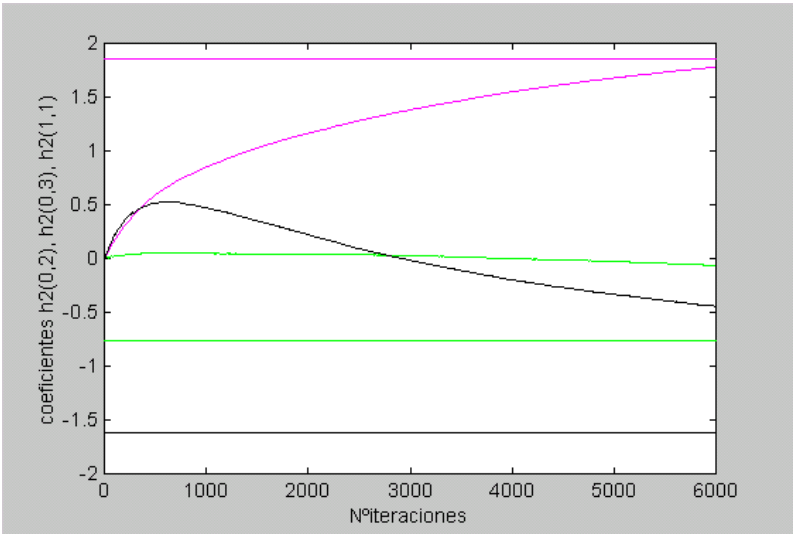
— $h_1(2)$ y $\hat{h}_1(2)$

— $h_1(3)$ y $\hat{h}_1(3)$

— $h_2(0,0)$ y $\hat{h}_2(0,0)$

— $h_2(0,1)$ y $\hat{h}_2(0,1)$

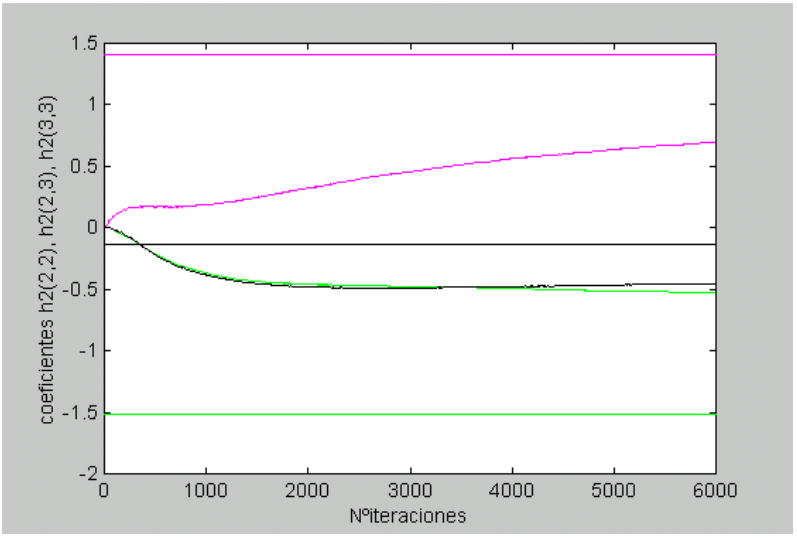



— $h_2(0,2)$ y $\hat{h}_2(0,2)$

— $h_2(0,3)$ y $\hat{h}_2(0,3)$

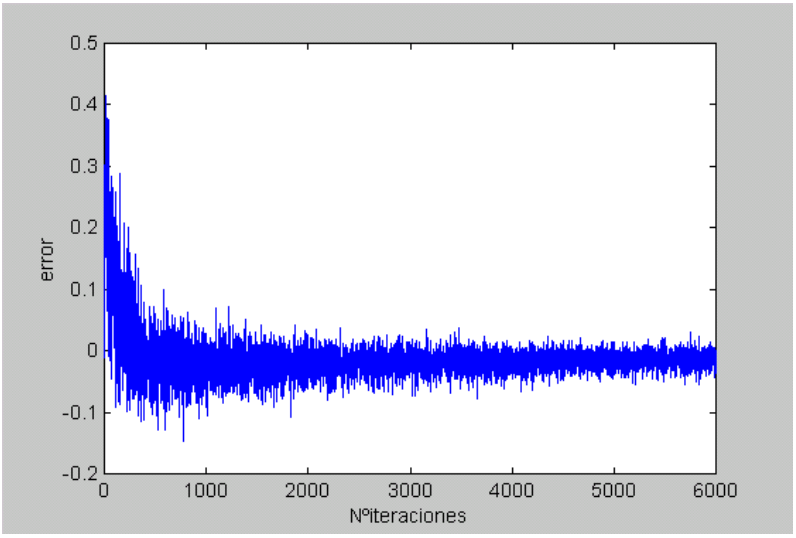
— $h_2(1,1)$ y $\hat{h}_2(1,1)$

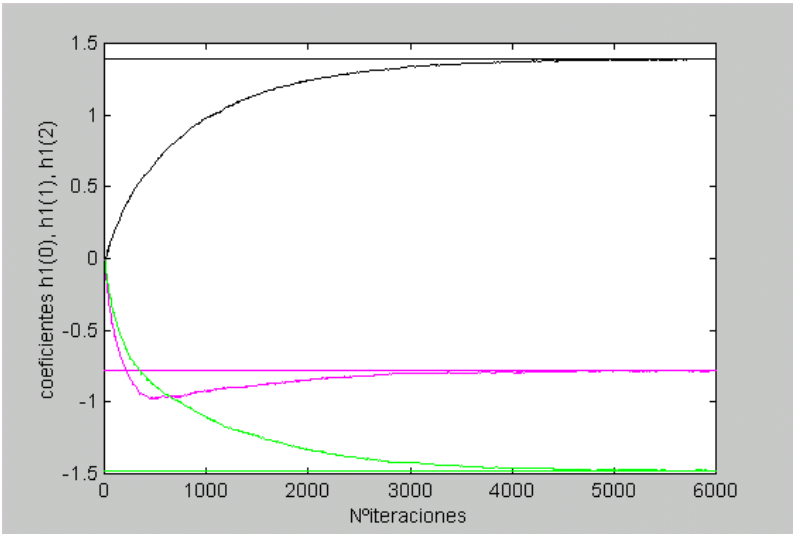



La convergencia se produce en las iteraciones 3751 y 414 de acuerdo con los criterios del 10% y del 15% respectivamente. Según el del 3%, no llega a producirse la convergencia.

Los resultados son mejores que antes. Los coeficientes lineales son siempre los que mejor se adaptan.

Se simula ahora con **m=0.05**. Los resultados son:






— $h_1(0)$ y $\hat{h}_1(0)$

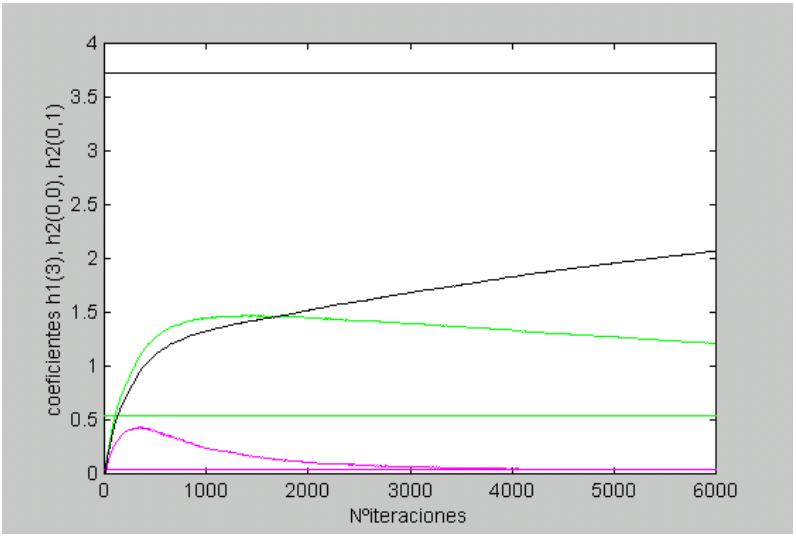
— $h_1(1)$ y $\hat{h}_1(1)$

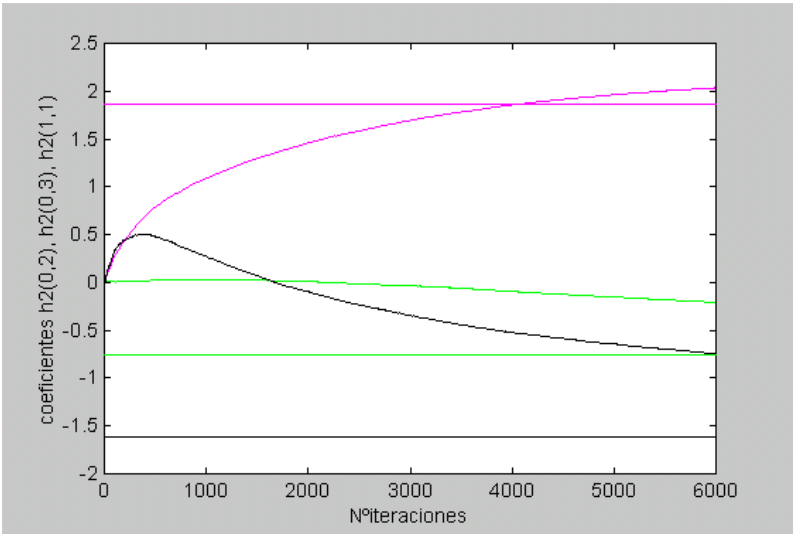
— $h_1(2)$ y $\hat{h}_1(2)$

— $h_1(3)$ y $\hat{h}_1(3)$

— $h_2(0,0)$ y $\hat{h}_2(0,0)$

— $h_2(0,1)$ y $\hat{h}_2(0,1)$

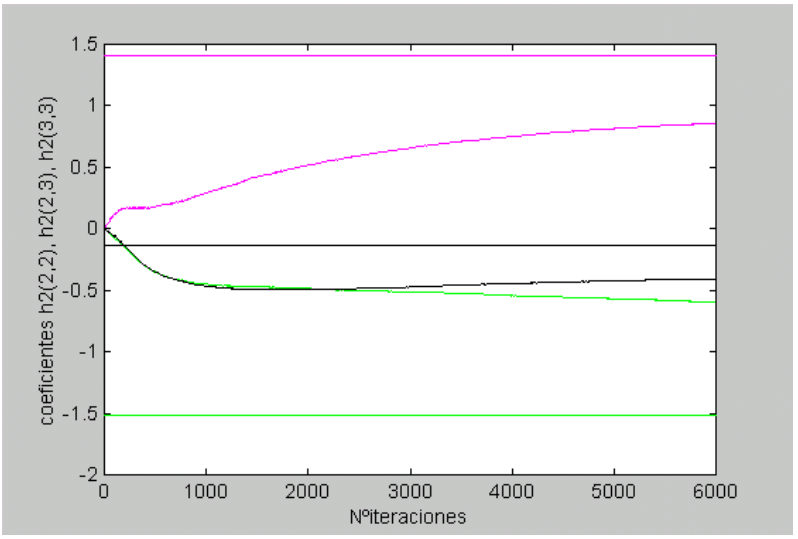



— $h_2(0,2)$ y $\hat{h}_2(0,2)$

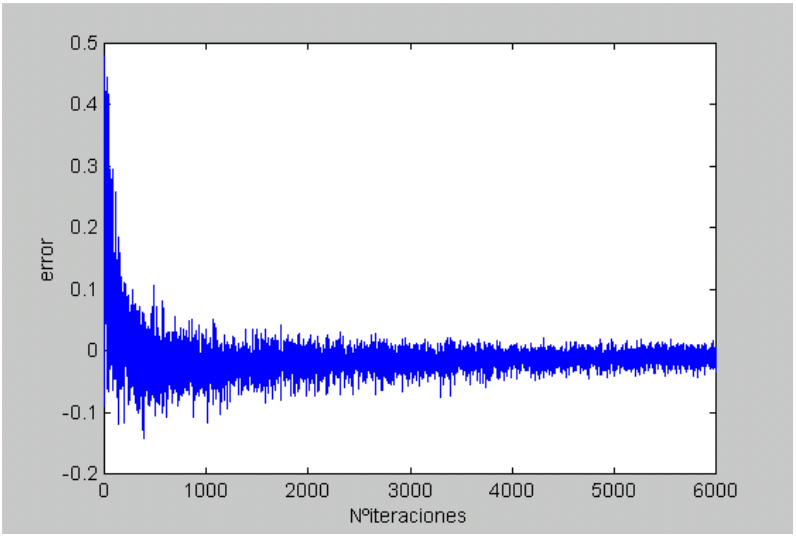
— $h_2(0,3)$ y $\hat{h}_2(0,3)$

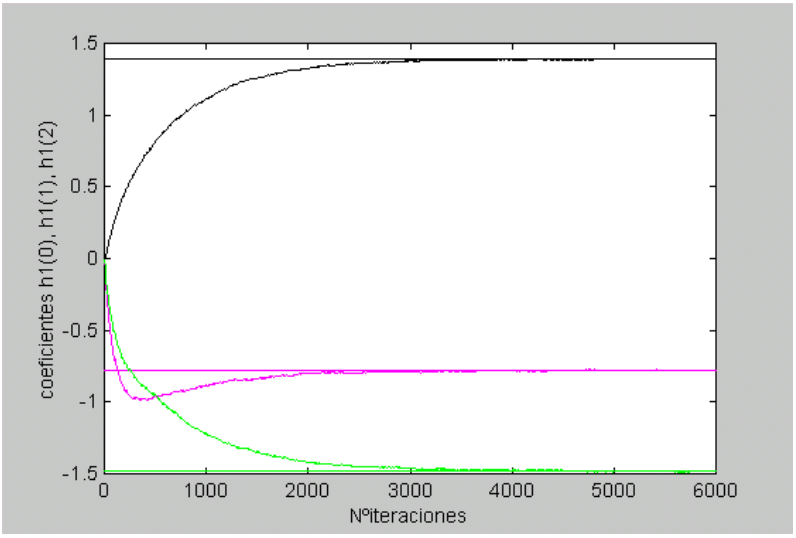
— $h_2(1,1)$ y $\hat{h}_2(1,1)$




El paso de adaptación sigue siendo pequeño para que se produzca la convergencia de acuerdo con el criterio del 3%. Según los criterios del 10% y del 15%, ésta se produce en las iteraciones 1834 y 307, respectivamente. Se sigue mejorando respecto a los casos anteriores.

Se vuelve a subir el **paso de adaptación**. Ahora se simula con un valor igual a **0.07**. Se obtiene:






— $h_1(0)$ y $\hat{h}_1(0)$

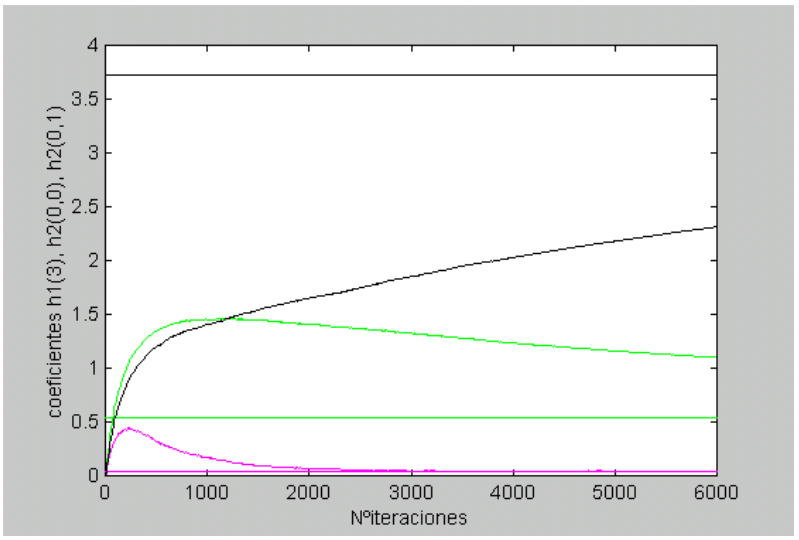
— $h_1(1)$ y $\hat{h}_1(1)$

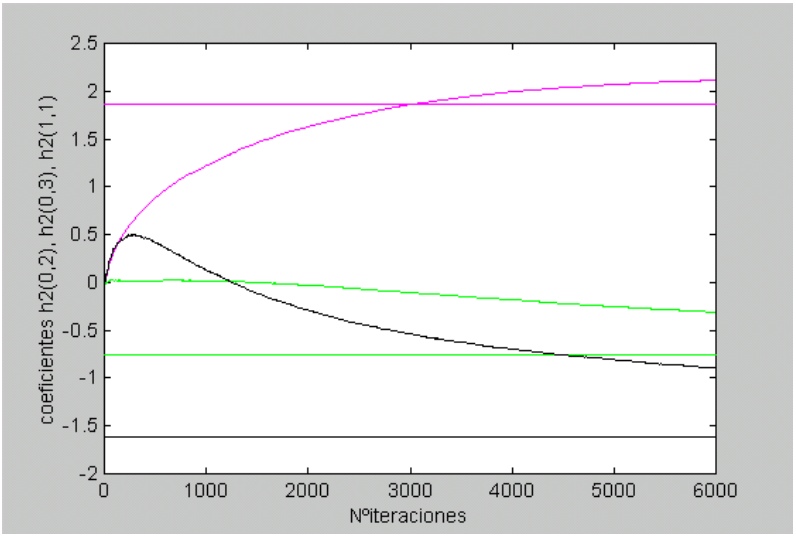
— $h_1(2)$ y $\hat{h}_1(2)$

— $h_1(3)$ y $\hat{h}_1(3)$

— $h_2(0,0)$ y $\hat{h}_2(0,0)$

— $h_2(0,1)$ y $\hat{h}_2(0,1)$

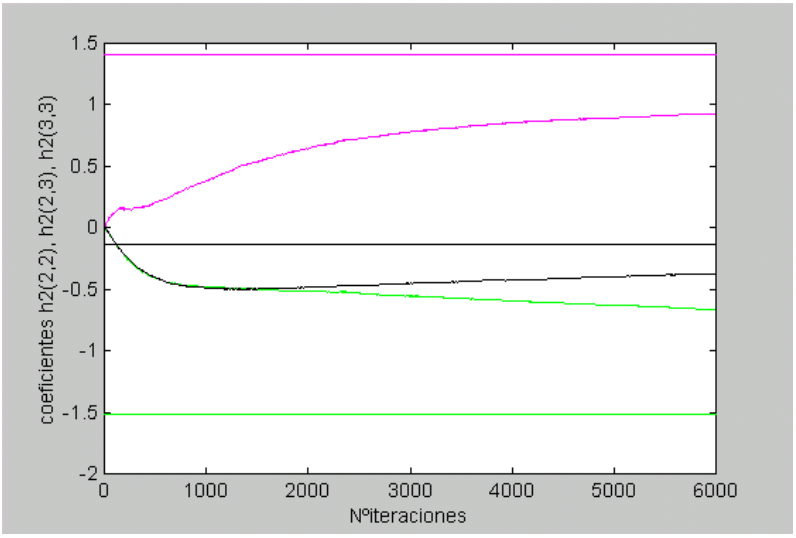



— $h_2(0,2)$ y $\hat{h}_2(0,2)$

— $h_2(0,3)$ y $\hat{h}_2(0,3)$

— $h_2(1,1)$ y $\hat{h}_2(1,1)$



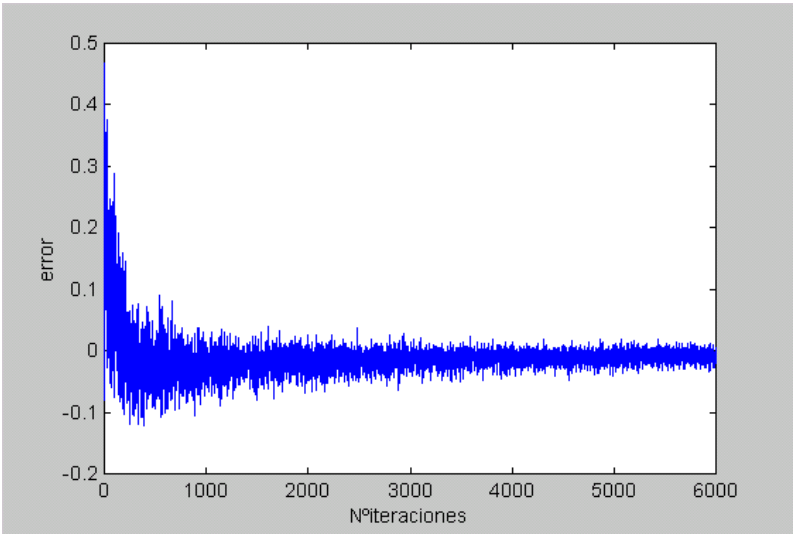

— $h_2(2,2)$ *y* $\hat{h}_2(2,2)$

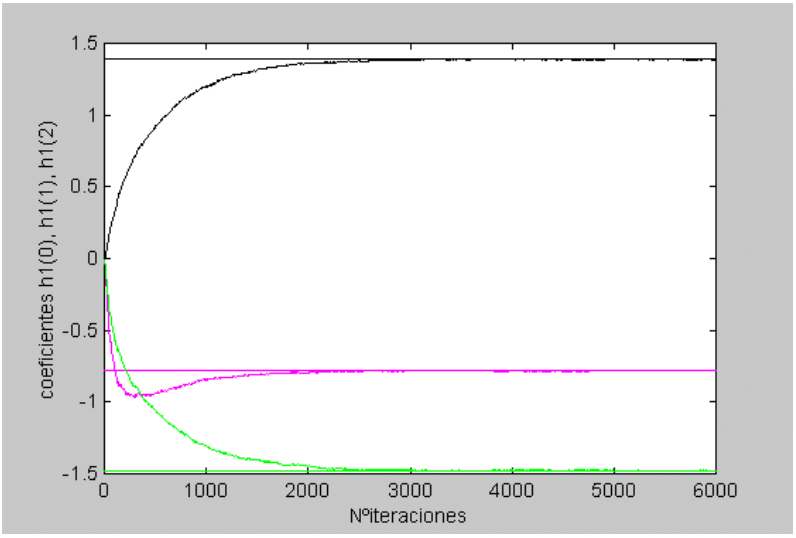
— $h_2(2,3)$ *y* $\hat{h}_2(2,3)$

— $h_2(3,3)$ *y* $\hat{h}_2(3,3)$

Se sigue mejorando en cuanto a velocidad de convergencia. Ésta se obtiene en las iteraciones 1141 y 152 según los criterios del 10% y del 15%. Para el criterio del 3% aún no se consigue.

Se simula a continuación con **m=0.09**. Las gráficas son:

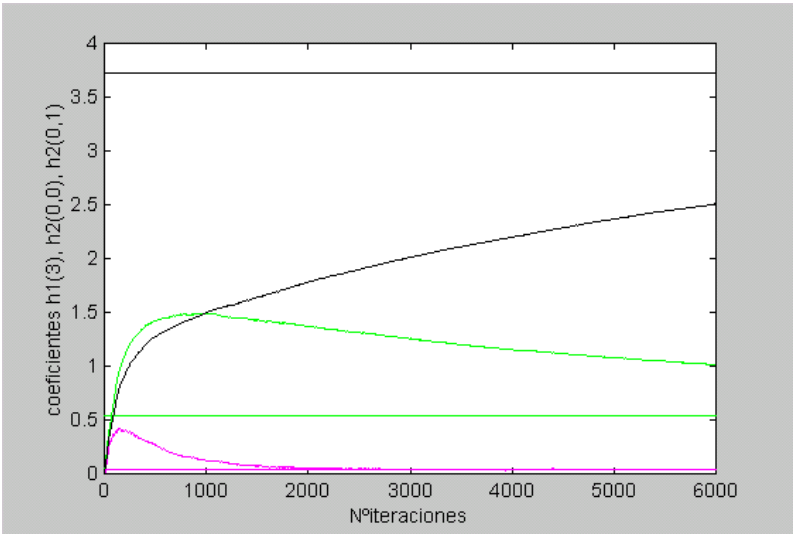



— $h_1(0)$ *y* $\hat{h}_1(0)$

— $h_1(1)$ *y* $\hat{h}_1(1)$

— $h_1(2)$ *y* $\hat{h}_1(2)$




— $h_1(3)$ *y* $\hat{h}_1(3)$

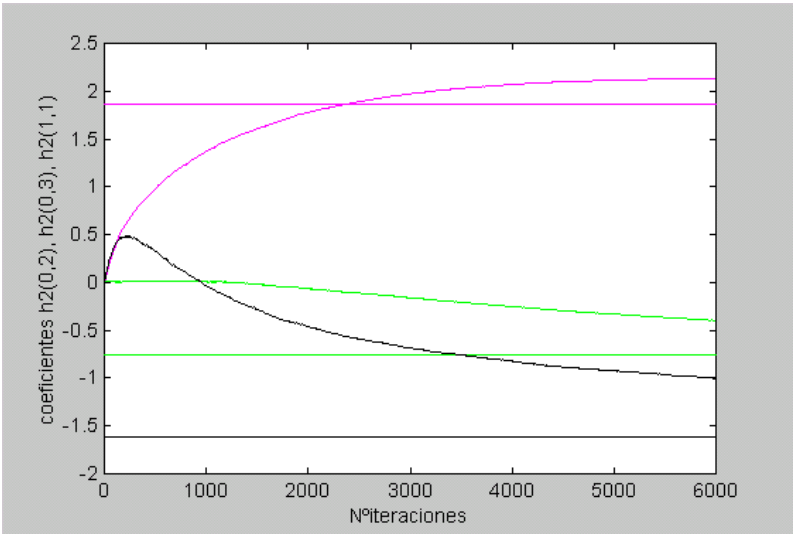
— $h_2(0,0)$ *y* $\hat{h}_2(0,0)$

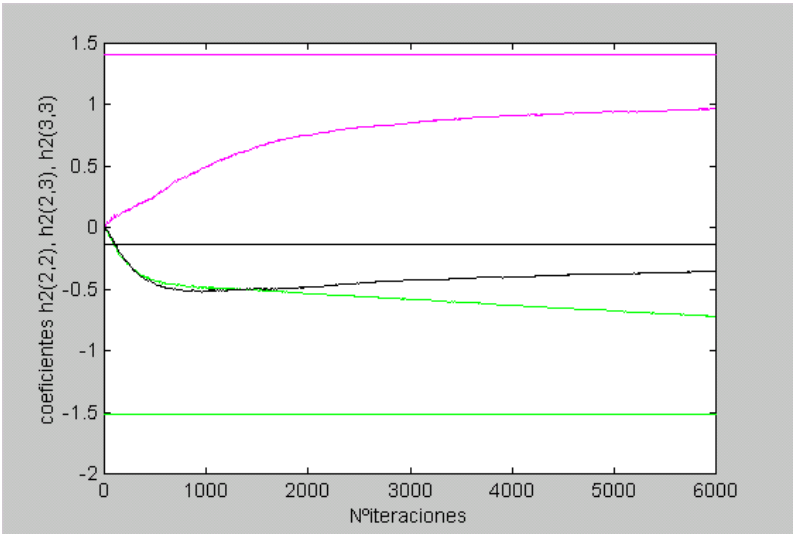
— $h_2(0,1)$ *y* $\hat{h}_2(0,1)$

— $h_2(0,2)$ *y* $\hat{h}_2(0,2)$

— $h_2(0,3)$ *y* $\hat{h}_2(0,3)$

— $h_2(1,1)$ *y* $\hat{h}_2(1,1)$



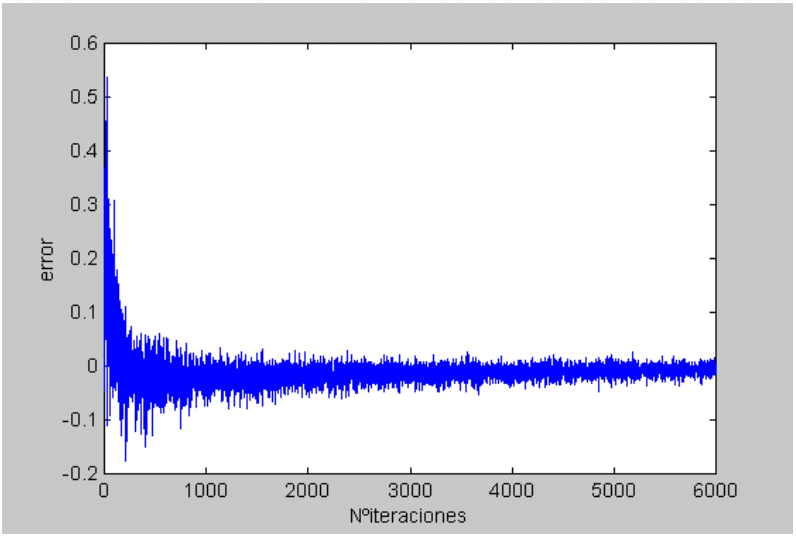

— $h_2(2,2)$ *y* $\hat{h}_2(2,2)$

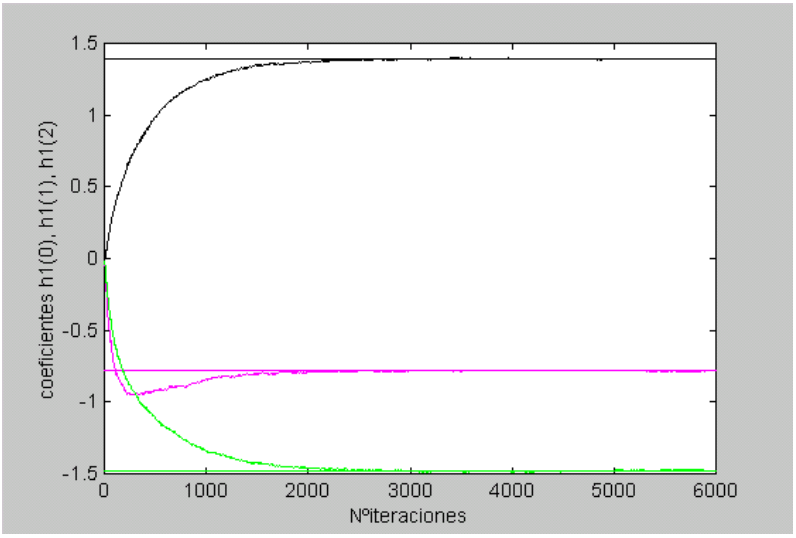
— $h_2(2,3)$ *y* $\hat{h}_2(2,3)$

— $h_2(3,3)$ *y* $\hat{h}_2(3,3)$

Para este valor se alcanza la convergencia en las iteraciones 5949, 891 y 177 según los criterios del 3%, 10% y 15% respectivamente. La convergencia es cada vez más rápida para los dos primeros criterios mientras que para el tercero no. Es decir, para una convergencia rápida y precisa el paso de adaptación óptimo no es el mismo que el que se necesita para una convergencia rápida pero no tan precisa.

Se sigue subiendo el **paso de adaptación** mientras se vaya mejorando en velocidad de convergencia con algún criterio. Así, se simula nuevamente con **m=0.1**. Los resultados son:

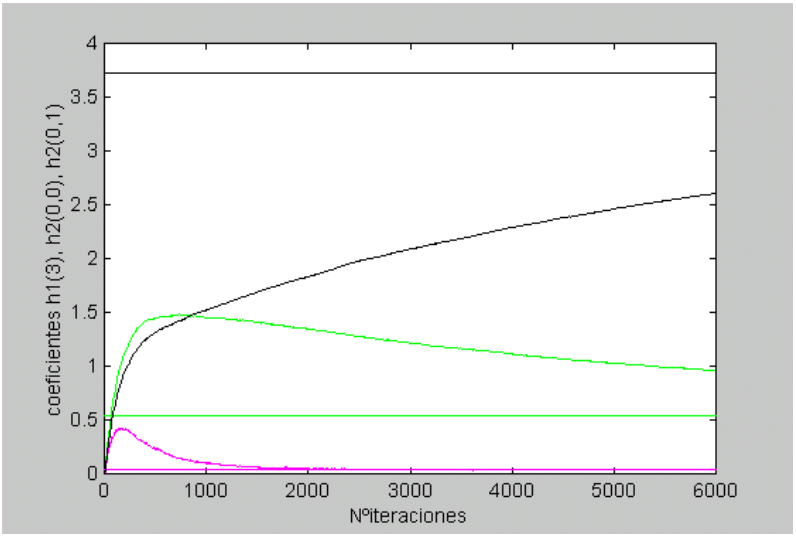



$h_1(0)$ y $\hat{h}_1(0)$

$h_1(1)$ y $\hat{h}_1(1)$

$h_1(2)$ y $\hat{h}_1(2)$




— $h_1(3)$ $y$ $\hat{h}_1(3)$

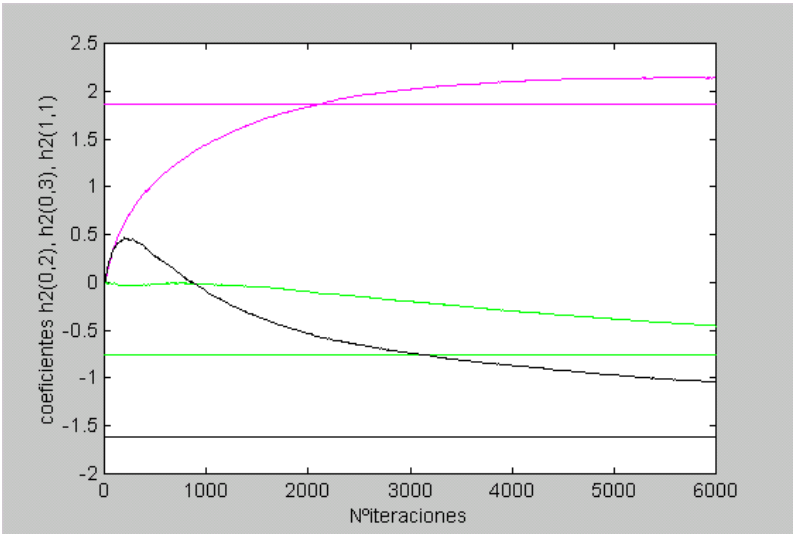
— $h_2(0,0)$ $y$ $\hat{h}_2(0,0)$

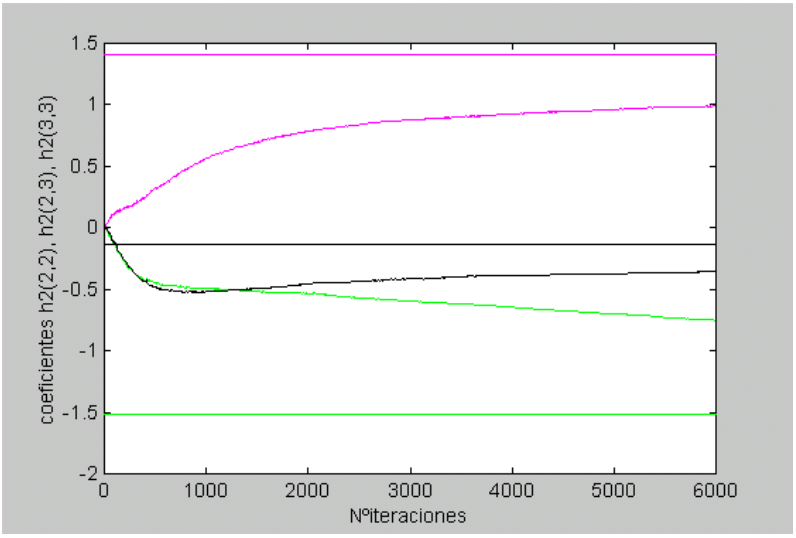
— $h_2(0,1)$ $y$ $\hat{h}_2(0,1)$

— $h_2(0,2)$ $y$ $\hat{h}_2(0,2)$

— $h_2(0,3)$ $y$ $\hat{h}_2(0,3)$

— $h_2(1,1)$ $y$ $\hat{h}_2(1,1)$




— $h_2(2,2)$ $y$ $\hat{h}_2(2,2)$

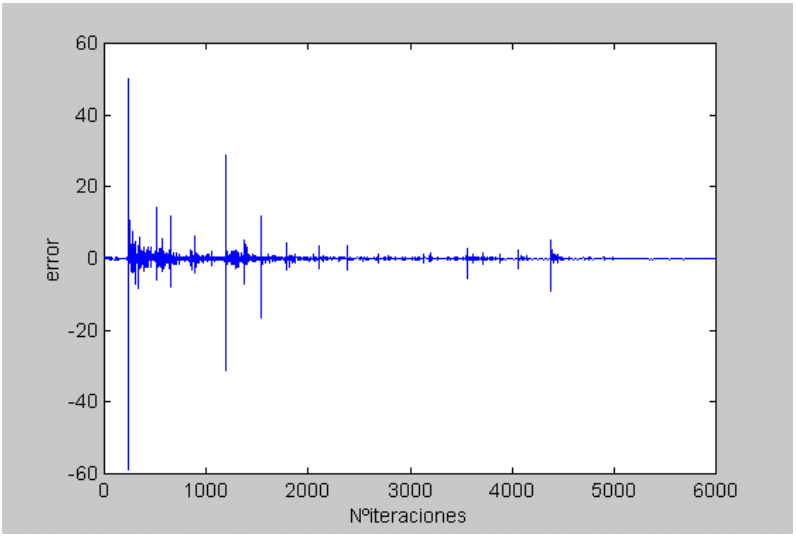
— $h_2(2,3)$ $y$ $\hat{h}_2(2,3)$

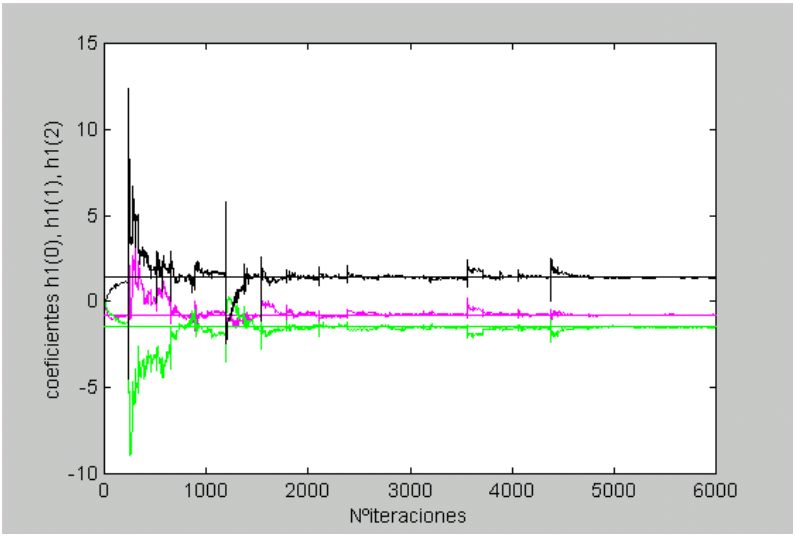
— $h_2(3,3)$ $y$ $\hat{h}_2(3,3)$

Se sigue mejorando la velocidad de convergencia para los dos primeros criterios. Ésta se alcanza en las iteraciones 5857, 755 y 204 según los criterios del 3%, 10% y 15% respectivamente.

Los coeficientes lineales convergen bastante bien. Los otros coeficientes tienen más dificultades para conseguirlo.

Se toma a continuación un valor para el **paso de adaptación** igual a **0.3**. Los resultados son:

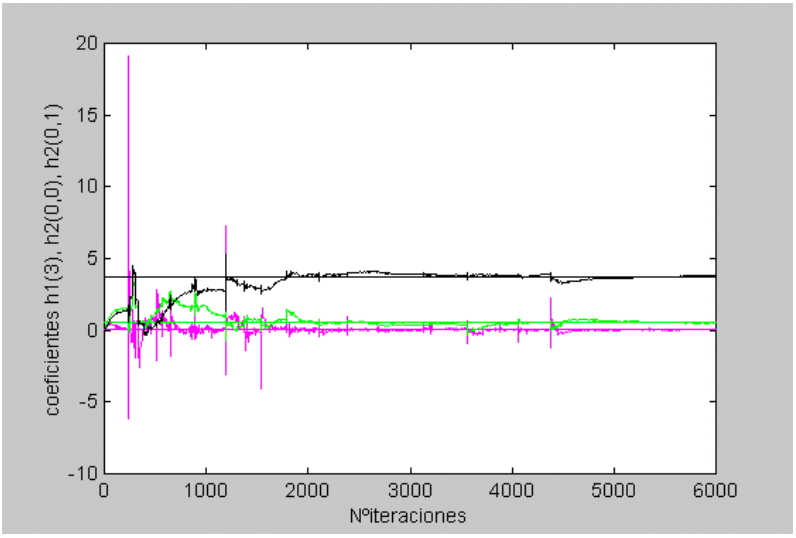



$h_1(0)$ y $\hat{h}_1(0)$

$h_1(1)$ y $\hat{h}_1(1)$

$h_1(2)$ y $\hat{h}_1(2)$




— $h_1(3)$ $y$ $\hat{h}_1(3)$

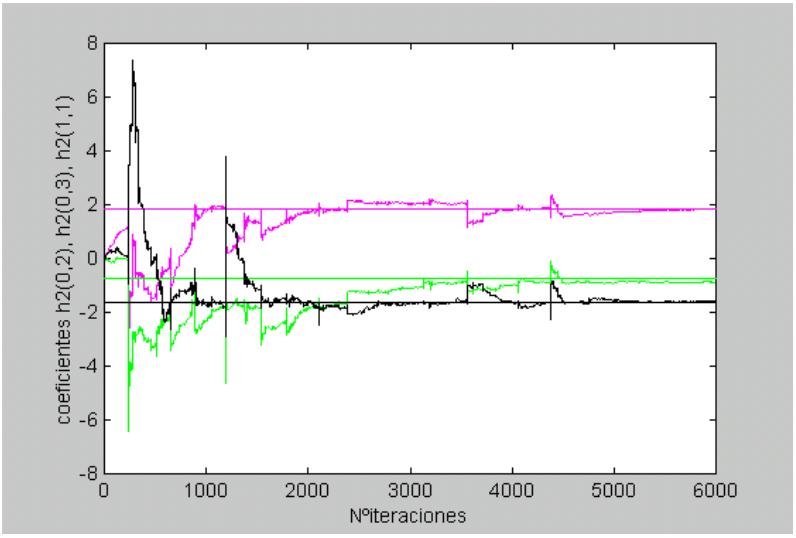
— $h_2(0,0)$ $y$ $\hat{h}_2(0,0)$

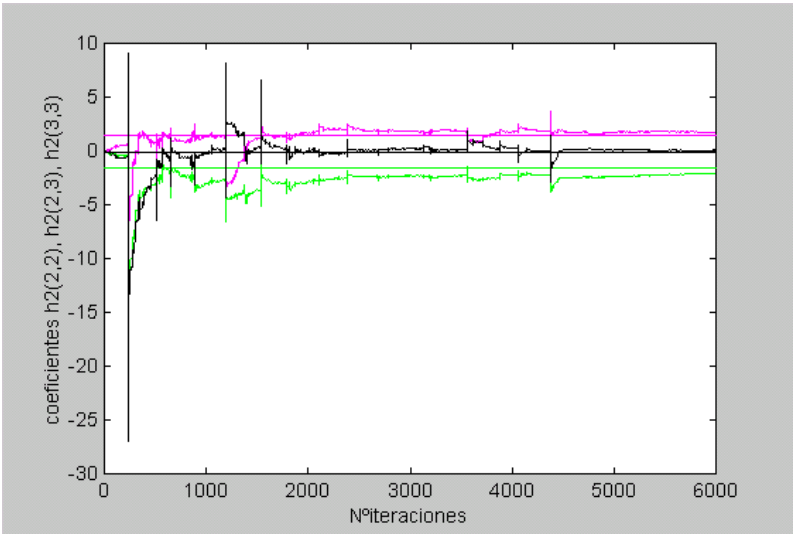
— $h_2(0,1)$ $y$ $\hat{h}_2(0,1)$

— $h_2(0,2)$ $y$ $\hat{h}_2(0,2)$

— $h_2(0,3)$ $y$ $\hat{h}_2(0,3)$

— $h_2(1,1)$ $y$ $\hat{h}_2(1,1)$



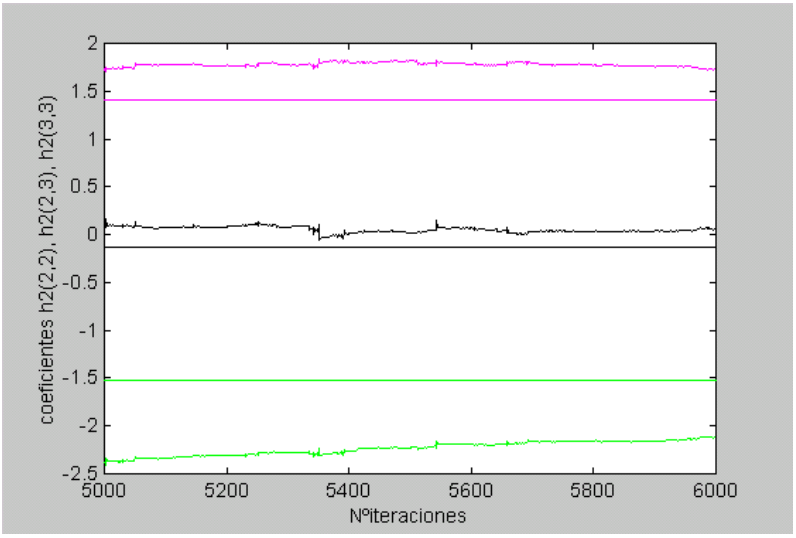

— $h_2(2,2)$ $y$ $\hat{h}_2(2,2)$

— $h_2(2,3)$ $y$ $\hat{h}_2(2,3)$

— $h_2(3,3)$ $y$ $\hat{h}_2(3,3)$

La velocidad de convergencia ha empeorado bastante. De hecho, ésta se alcanza sólo para los criterios del 10% y del 15% y para ambos en la iteración 5974.

Aunque parezca por las gráficas que los coeficientes alcanzan su valor final, hay que destacar que los valores del eje de ordenadas son muy elevados. Para verlo mejor se representan las últimas 1000 iteraciones para los tres últimos coeficientes:




Se ve que no se alcanza la convergencia.

Se resumen los resultados en una tabla:

| m | criterio/ convergencia | criterio/ convergencia | criterio/ convergencia |
|---|---|---|---|
| 0.01 | 3%/X | 10%/5326 | 15%/1937 |
| 0.03 | 3%/X | 10%/3751 | 15%/414 |
| 0.05 | 3%/X | 10%/1834 | 15%/307 |
| **0.07** | 3%/X | 10%/1141 | **15%/152** |
| 0.09 | 3%/5949 | 10%/891 | 15%/177 |
| **0.1** | **3%/5857** | **10%/755** | 15%/204 |
| 0.3 | 3%/X | 10%/5974 | 15%/5974 |

Resultados para el algoritmo LMS

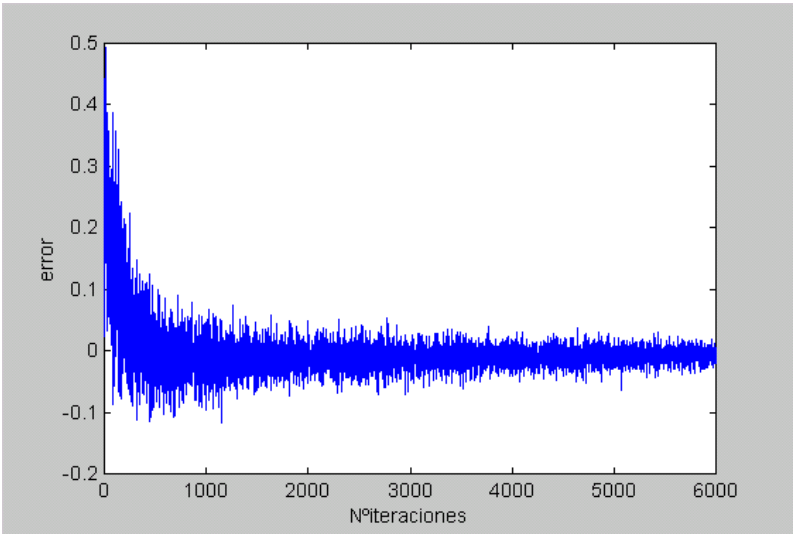
Se tienen dos pasos de adaptación óptimos dependiendo de la precisión que se exija. Evidentemente, cuantas más iteraciones, el número de operaciones necesarias es mayor.

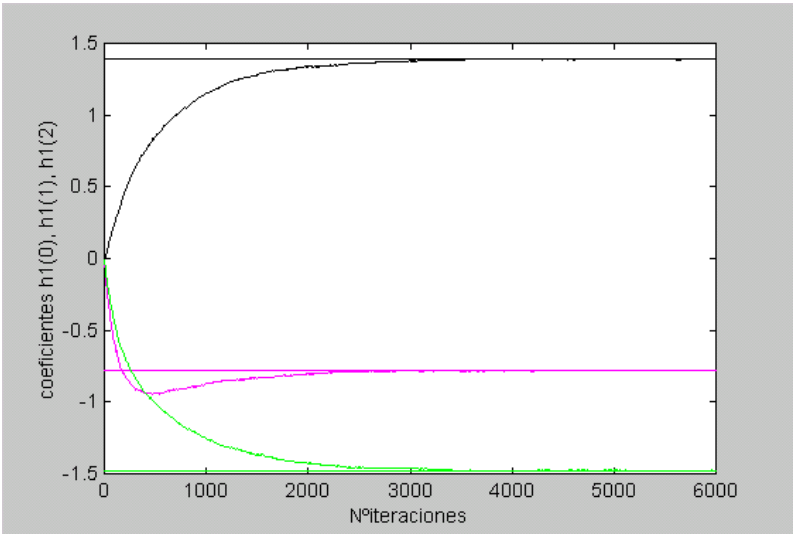
- **LMS normalizado:**

Se sigue la misma dinámica que con el algoritmo anterior, es decir, se utilizan los mismos tres criterios de convergencia y se simula para distintos valores del paso de adaptación.

Sabemos del ejemplo anterior que para este algoritmo el paso de adaptación es un poco mayor que para el LMS.

Se comienza con **m=0.05**. Los resultados son:

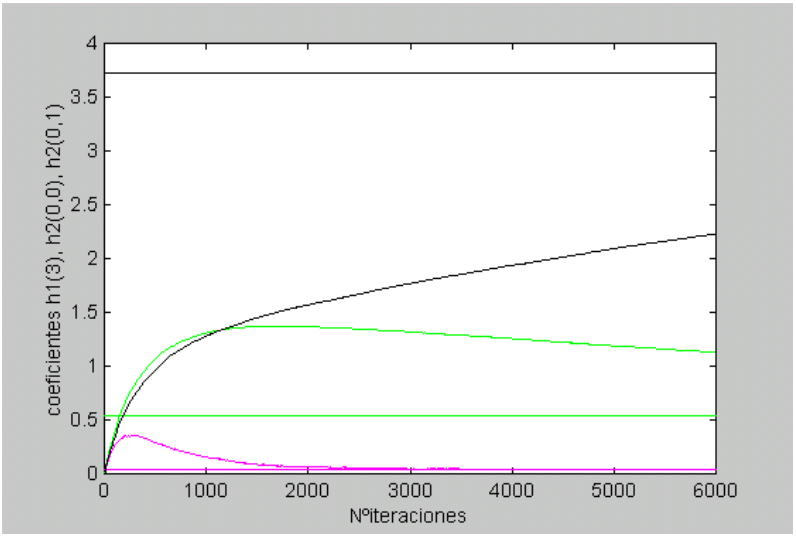



$h_1(0)$ y $\hat{h}_1(0)$

$h_1(1)$ y $\hat{h}_1(1)$

$h_1(2)$ y $\hat{h}_1(2)$




— $h_1(3)$ y $\hat{h}_1(3)$

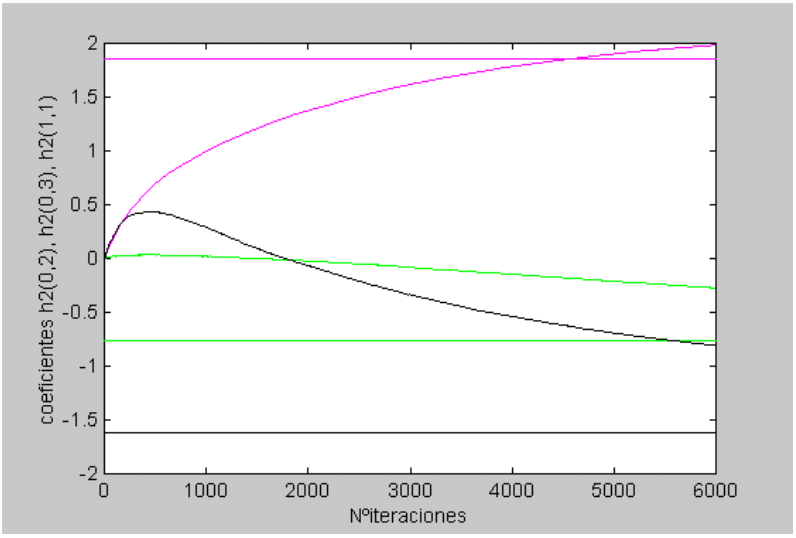
— $h_2(0,0)$ y $\hat{h}_2(0,0)$

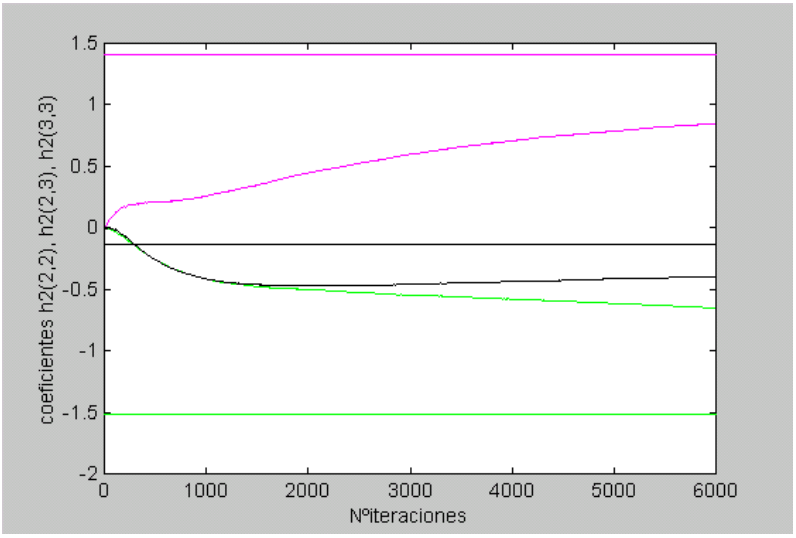
— $h_2(0,1)$ y $\hat{h}_2(0,1)$

— $h_2(0,2)$ y $\hat{h}_2(0,2)$

— $h_2(0,3)$ y $\hat{h}_2(0,3)$

— $h_2(1,1)$ y $\hat{h}_2(1,1)$




— $h_2(2,2)$ y $\hat{h}_2(2,2)$

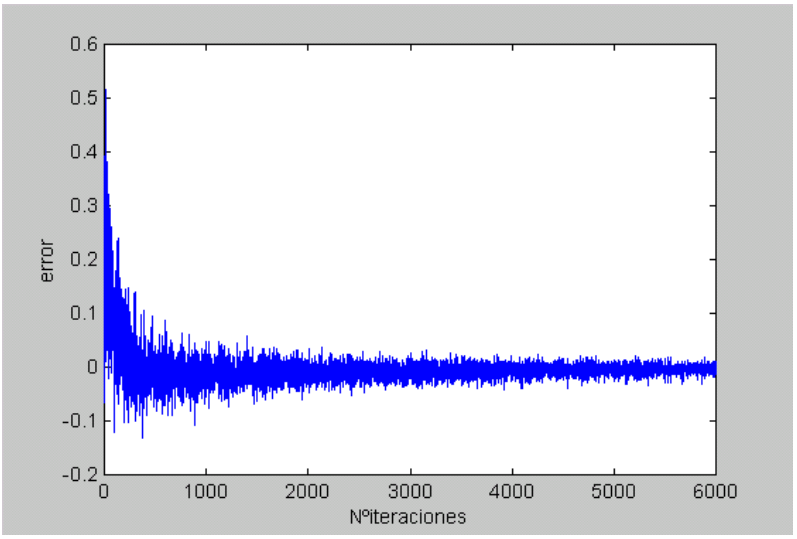
— $h_2(2,3)$ y $\hat{h}_2(2,3)$

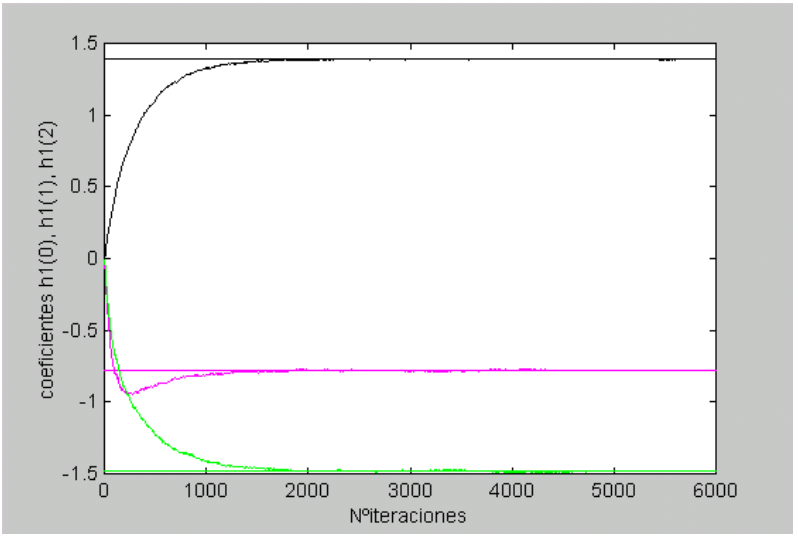
— $h_2(3,3)$ y $\hat{h}_2(3,3)$

La convergencia se produce en las iteraciones 5887, 1153 y 247 según los criterios del 3%, 10% y 15% respectivamente. Estos resultados son mejores que los del LMS. Lógicamente, el número de operaciones necesarias en este algoritmo es superior al LMS. En este último se realizaban 165 millones de operaciones en las 6000 iteraciones. Aquí se realizan 186 millones en las mismas 6000 iteraciones.

Dependerá de lo que se pueda utilizar en cada caso para elegir uno u otro.

A continuación se simula con un **paso de adaptación** igual a **0.09**. Las gráficas son:

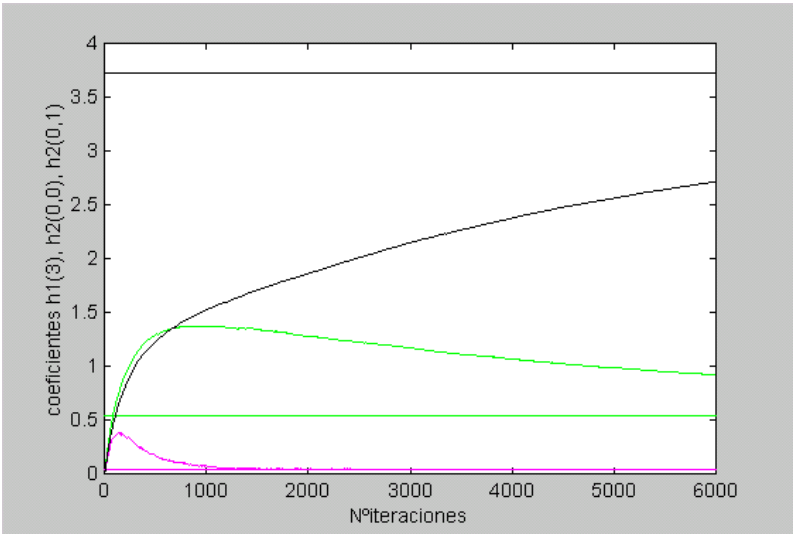



— $h_1(0)$ y $\hat{h}_1(0)$

— $h_1(1)$ y $\hat{h}_1(1)$

— $h_1(2)$ y $\hat{h}_1(2)$




— $h_1(3)$ y $\hat{h}_1(3)$

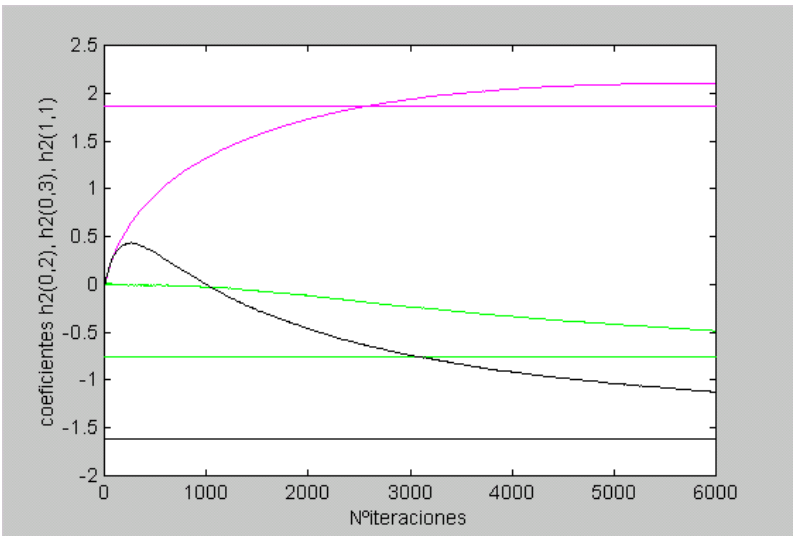
— $h_2(0,0)$ y $\hat{h}_2(0,0)$

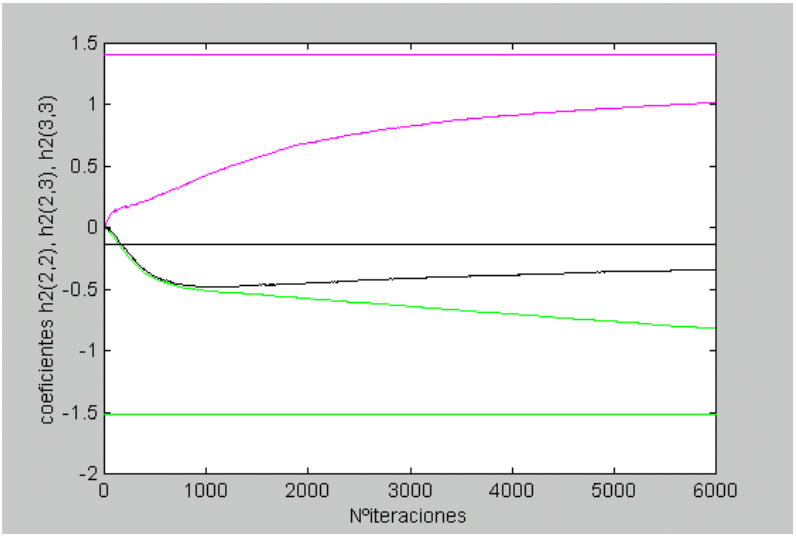
— $h_2(0,1)$ y $\hat{h}_2(0,1)$

— $h_2(0,2)$ y $\hat{h}_2(0,2)$

— $h_2(0,3)$ y $\hat{h}_2(0,3)$

— $h_2(1,1)$ y $\hat{h}_2(1,1)$



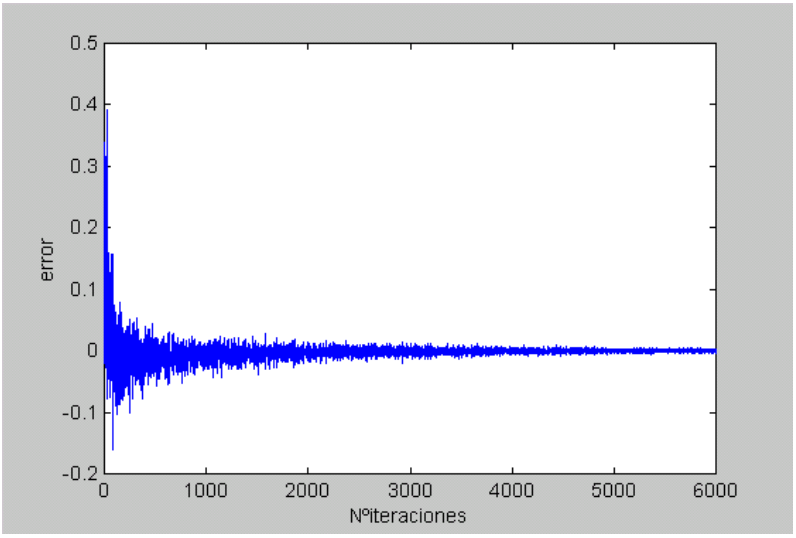

— $h_2(2,2)$ y $\hat{h}_2(2,2)$

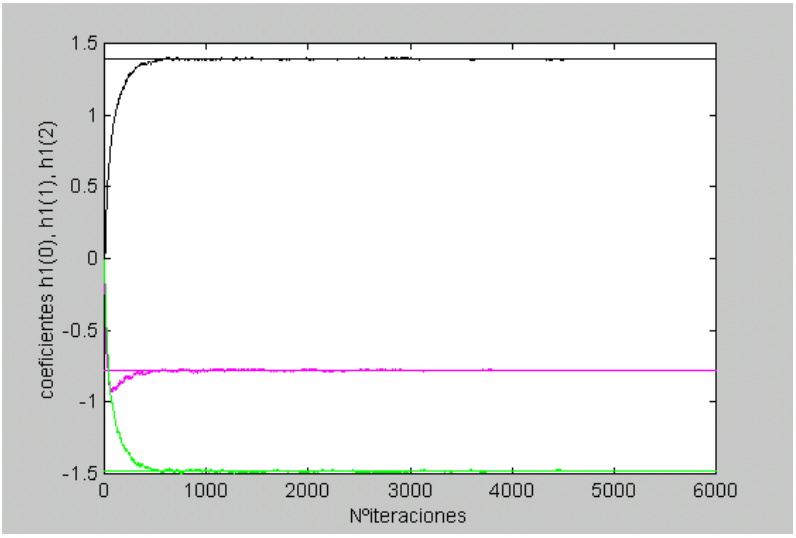
— $h_2(2,3)$ y $\hat{h}_2(2,3)$

— $h_2(3,3)$ y $\hat{h}_2(3,3)$

La velocidad de convergencia mejora respecto a la simulación anterior. Ésta se alcanza en las iteraciones 5860, 889 y 148 según los criterios del 3%, 10% y 15% respectivamente.

Se simula a continuación con **m=0.3**. Los resultados son:




— $h_1(0)$ y $\hat{h}_1(0)$

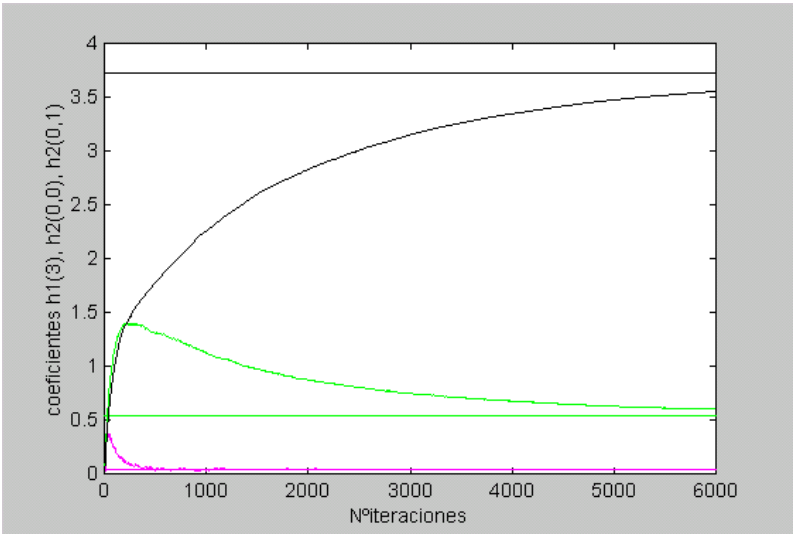
— $h_1(1)$ y $\hat{h}_1(1)$

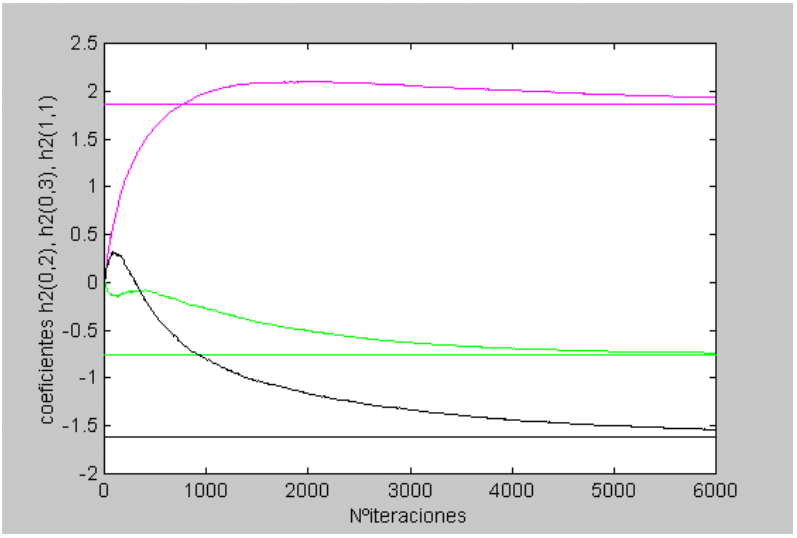
— $h_1(2)$ y $\hat{h}_1(2)$

— $h_1(3)$ y $\hat{h}_1(3)$

— $h_2(0,0)$ y $\hat{h}_2(0,0)$

— $h_2(0,1)$ y $\hat{h}_2(0,1)$






— $h_2(0,2)$ y $\hat{h}_2(0,2)$

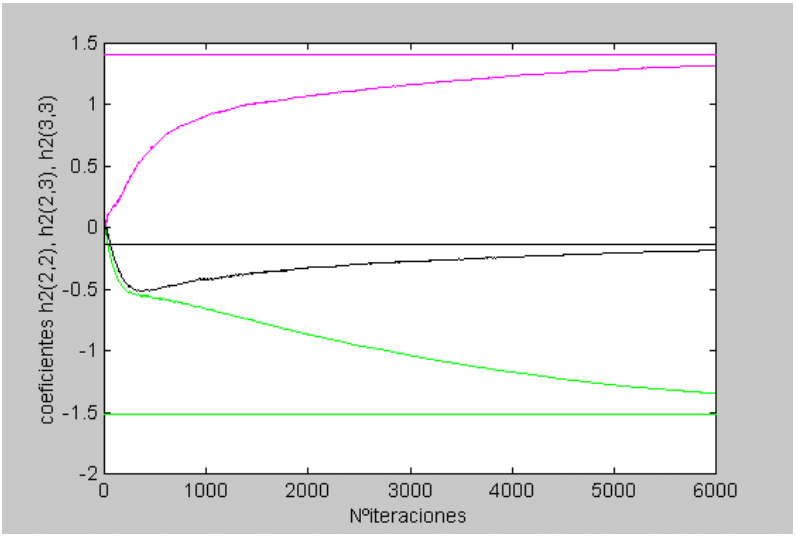
— $h_2(0,3)$ y $\hat{h}_2(0,3)$

— $h_2(1,1)$ y $\hat{h}_2(1,1)$

— $h_2(2,2)$ y $\hat{h}_2(2,2)$

— $h_2(2,3)$ y $\hat{h}_2(2,3)$

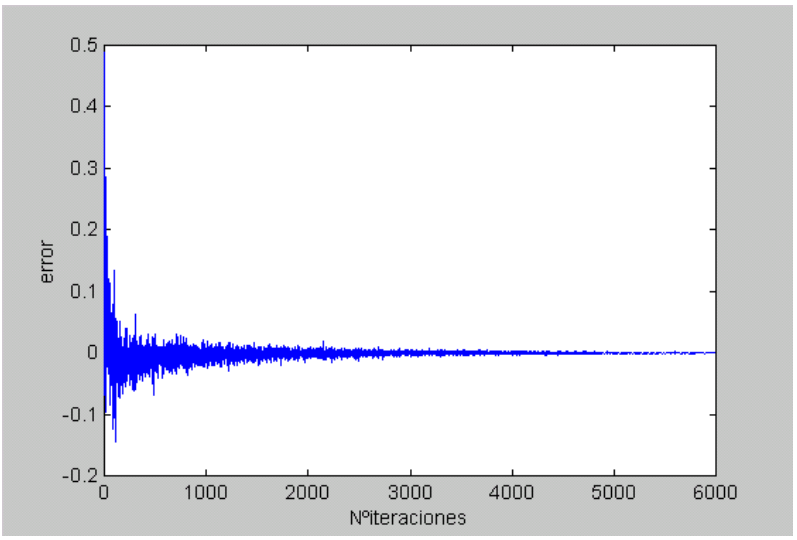
— $h_2(3,3)$ y $\hat{h}_2(3,3)$

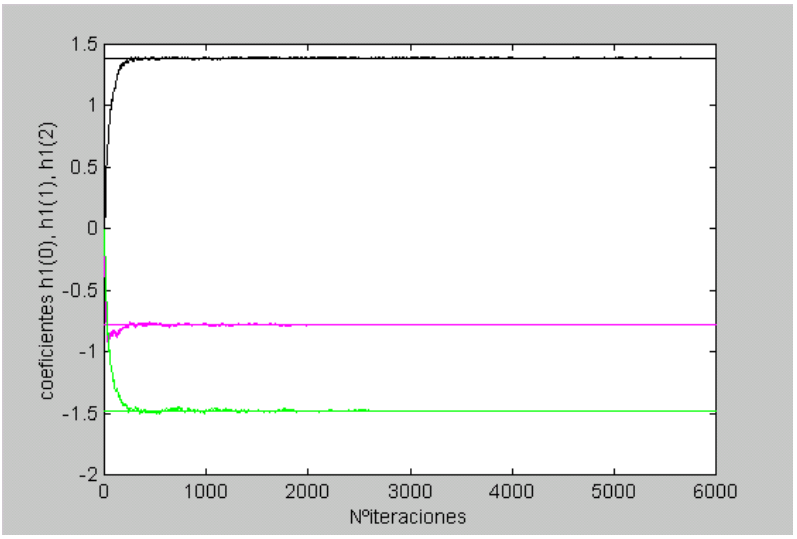



La convergencia se produce en las iteraciones 1515, 256 y 81 según los criterios del 3%, 10% y 15% respectivamente. Estos resultados son mucho mejores que los del LMS.

Además, la convergencia de los últimos coeficientes es mucho mejor aquí que en el LMS. Esto es lo que hace que la convergencia de la señal estimada sea más rápida en este algoritmo LMS normalizado.

Volvemos a subir el **paso de adaptación** y se toma un valor de **0.5**. Los resultados son:




— $h_1(0)$ y $\hat{h}_1(0)$

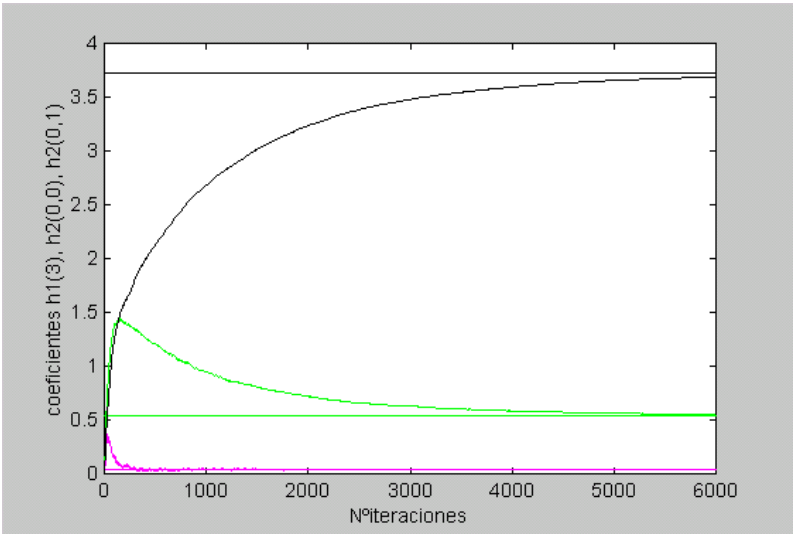
— $h_1(1)$ y $\hat{h}_1(1)$

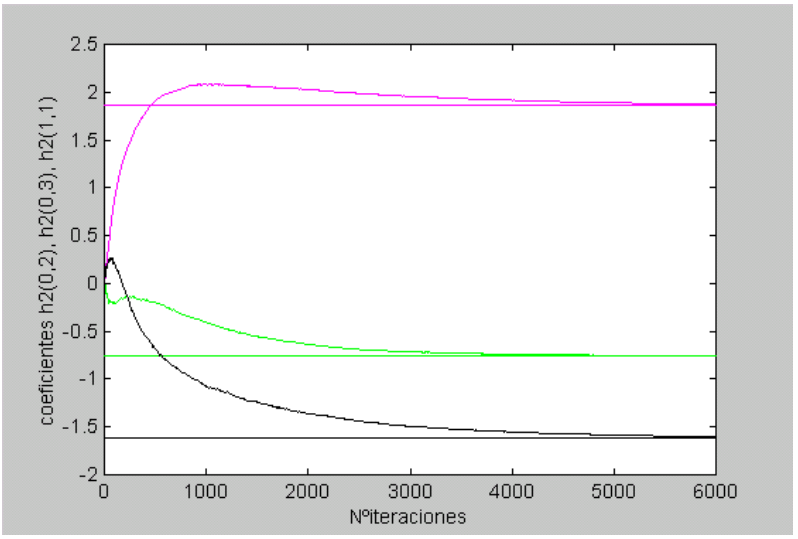
— $h_1(2)$ y $\hat{h}_1(2)$

— $h_1(3)$ y $\hat{h}_1(3)$

— $h_2(0,0)$ y $\hat{h}_2(0,0)$

— $h_2(0,1)$ y $\hat{h}_2(0,1)$






— $h_2(0,2)$ y $\hat{h}_2(0,2)$

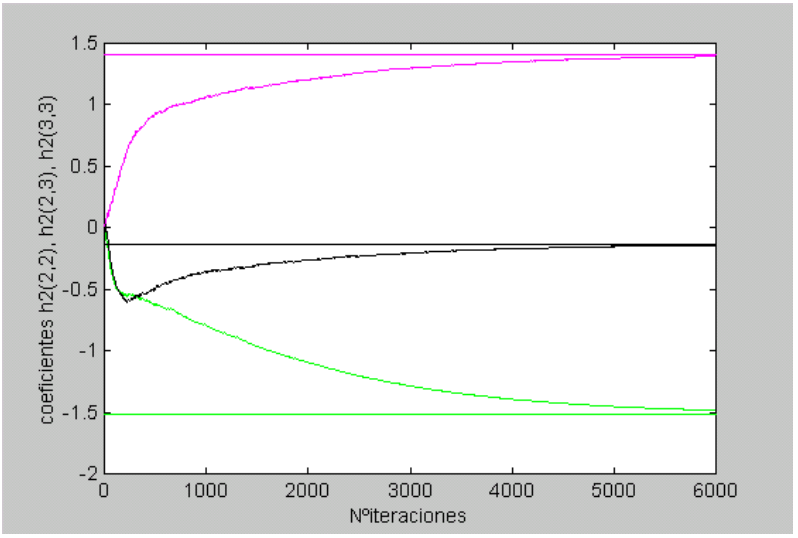
— $h_2(0,3)$ y $\hat{h}_2(0,3)$

— $h_2(1,1)$ y $\hat{h}_2(1,1)$

— $h_2(2,2)$ y $\hat{h}_2(2,2)$

— $h_2(2,3)$ y $\hat{h}_2(2,3)$

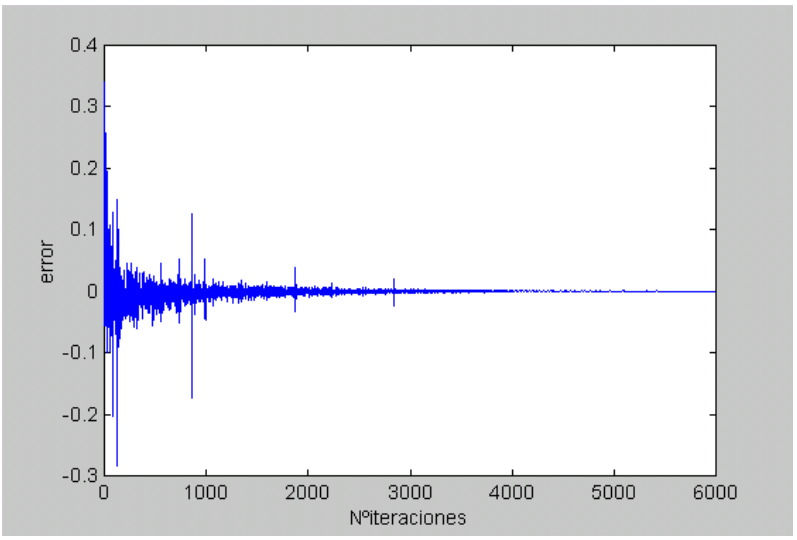
— $h_2(3,3)$ y $\hat{h}_2(3,3)$

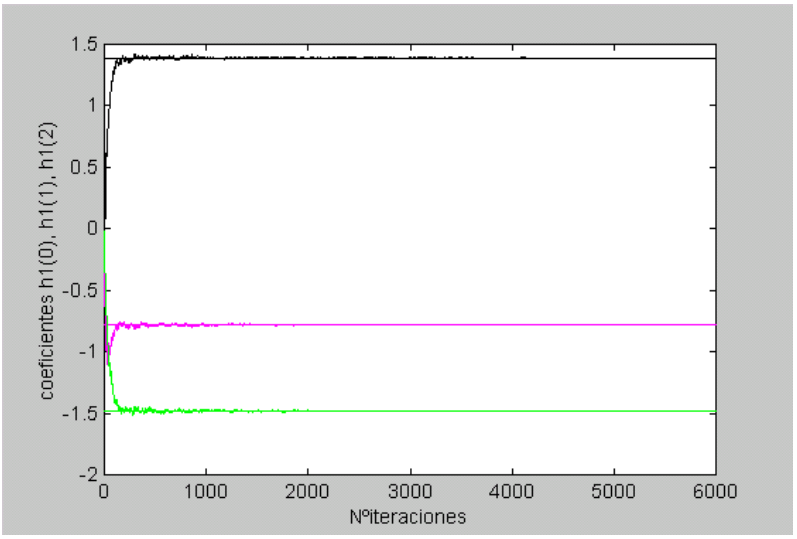



La convergencia se produce en las iteraciones 823, 117 y 32 de acuerdo con los criterios del 3%, 10% y 15% respectivamente. Esta velocidad de convergencia es muy superior a la que se obtuvo en el caso del LMS.

Es interesante destacar de nuevo que ahora todos los coeficientes se adaptan bastante bien.

Finalmente se simula con **m=0.7**. Las gráficas son:




— $h_1(0)$ y $\hat{h}_1(0)$

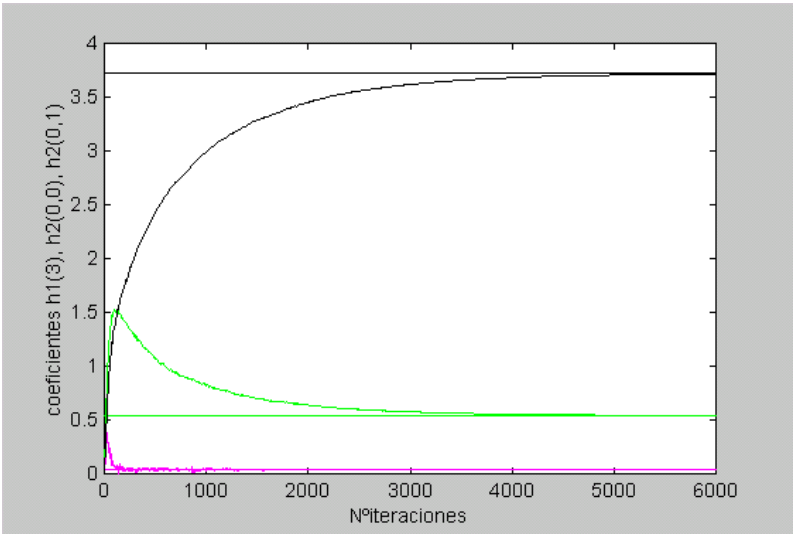
— $h_1(1)$ y $\hat{h}_1(1)$

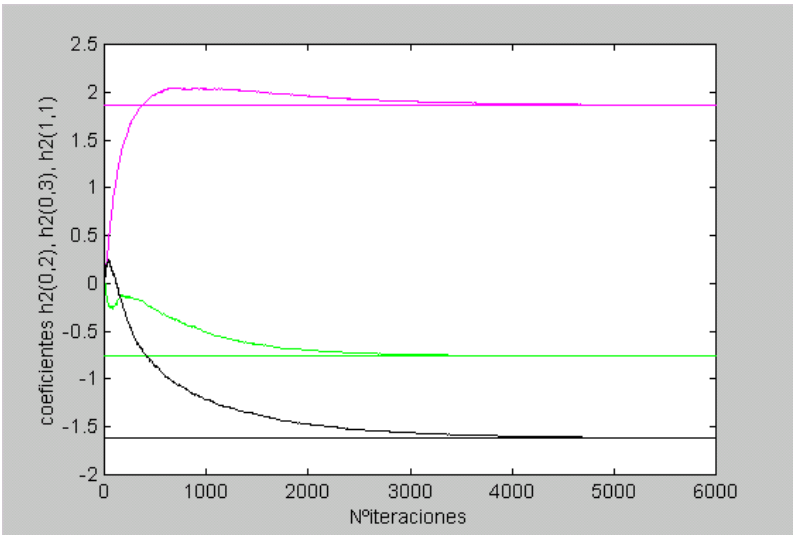
— $h_1(2)$ y $\hat{h}_1(2)$

— $h_1(3)$ y $\hat{h}_1(3)$

— $h_2(0,0)$ y $\hat{h}_2(0,0)$

— $h_2(0,1)$ y $\hat{h}_2(0,1)$






$h_2(0,2)$ y $\hat{h}_2(0,2)$

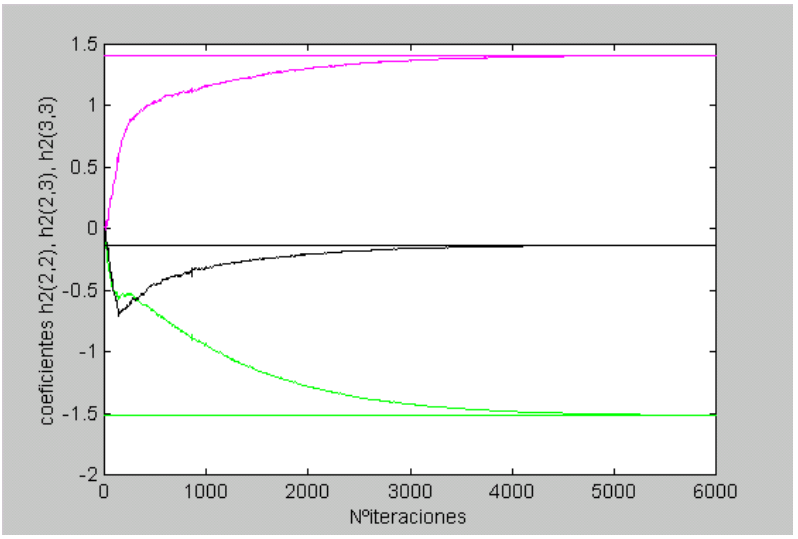
$h_2(0,3)$ y $\hat{h}_2(0,3)$

$h_2(1,1)$ y $\hat{h}_2(1,1)$

$h_2(2,2)$ y $\hat{h}_2(2,2)$

$h_2(2,3)$ y $\hat{h}_2(2,3)$

$h_2(3,3)$ y $\hat{h}_2(3,3)$




La convergencia se produce en las iteraciones 1870, 865 y 864 según los criterios del 3%, 10% y 15%. Este resultado ya es peor que el anterior. Se puede concluir que el valor óptimo del paso de adaptación es $\mu$=0.5.

La velocidad de convergencia es superior a la que se obtiene con el algoritmo anterior aunque el número de operaciones necesarias también es superior.

Vamos a resumir en una tabla todos los resultados:

| m | criterio/ convergencia | criterio/ convergencia | criterio/ convergencia |
|---|---|---|---|
| 0.05 | 3%/5887 | 10%/1153 | 15%/247 |
| 0.09 | 3%/5860 | 10%/889 | 15%/148 |
| 0.3 | 3%/1515 | 10%/256 | 15%/81 |
| **0.5** | **3%/823** | **10%/117** | **15%/32** |
| 0.7 | 3%/1870 | 10%/865 | 15%/864 |

Resultados con el algoritmo LMS normalizado

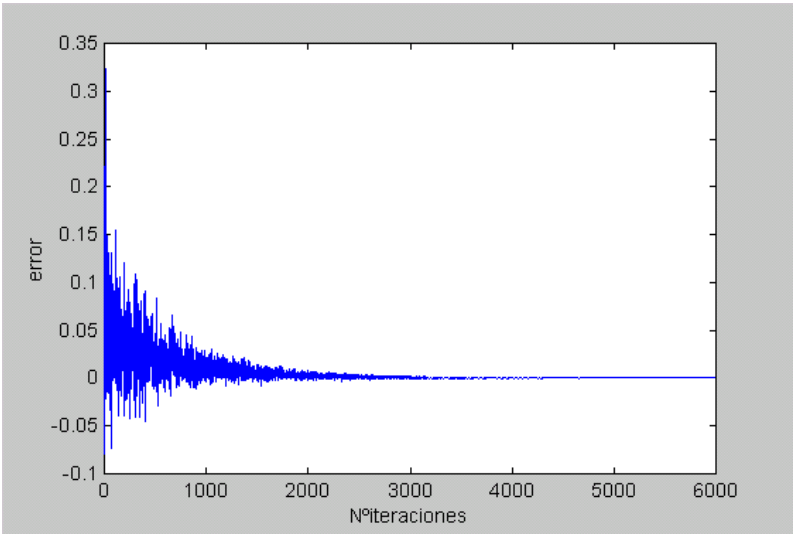
- **RLS:**

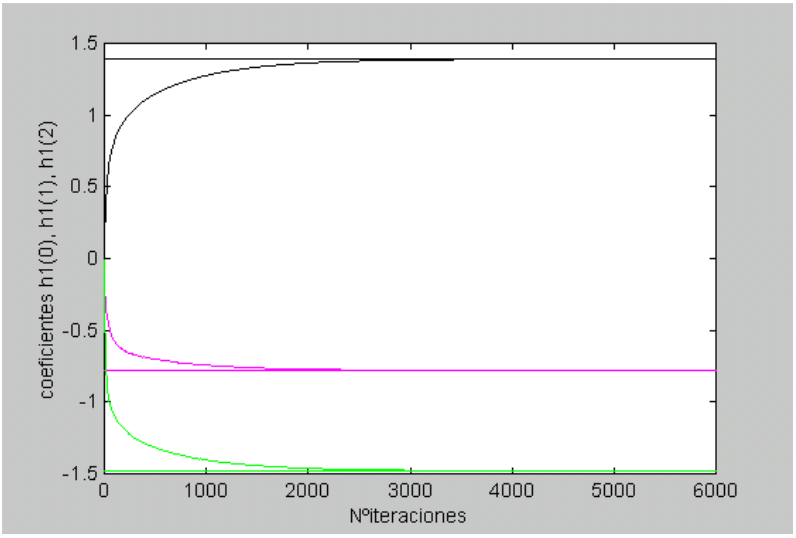
Por último se va a analizar este ejemplo con el algoritmo RLS. A priori es un algoritmo más eficaz en cuanto a velocidad de convergencia aunque requiere un mayor número de operaciones.

Se consideran los mismos criterios de convergencia que en los dos algoritmos anteriores, es decir, el del 3%, 10% y 15%.

Se analizan varios valores del paso de adaptación. Como ya se ha comentado, la mayoría de algoritmos RLS utilizan la unidad como valor por defecto para este parámetro. Por eso se empieza por un valor de 0.3 y se va subiendo en las siguientes simulaciones.

los resultados para un **paso de adaptación** de **0.3** son:

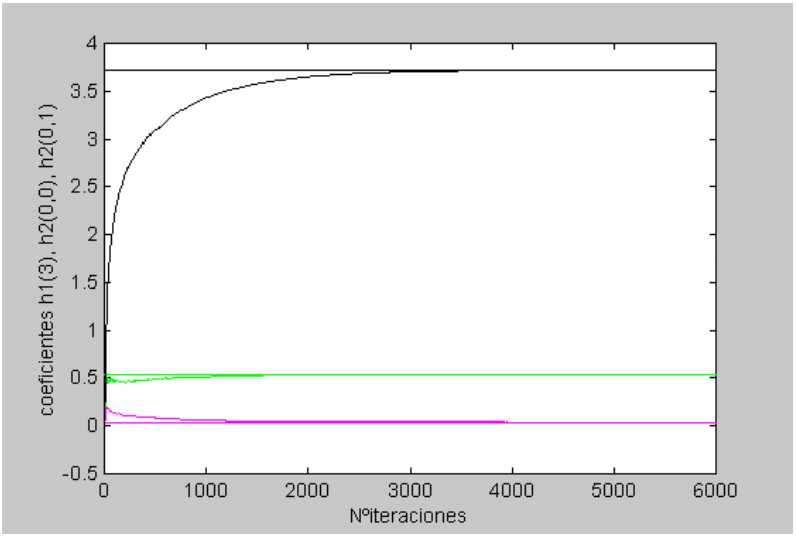



$h_1(0)$ y $\hat{h}_1(0)$

$h_1(1)$ y $\hat{h}_1(1)$

$h_1(2)$ y $\hat{h}_1(2)$




$h_1(3)$ y $\hat{h}_1(3)$

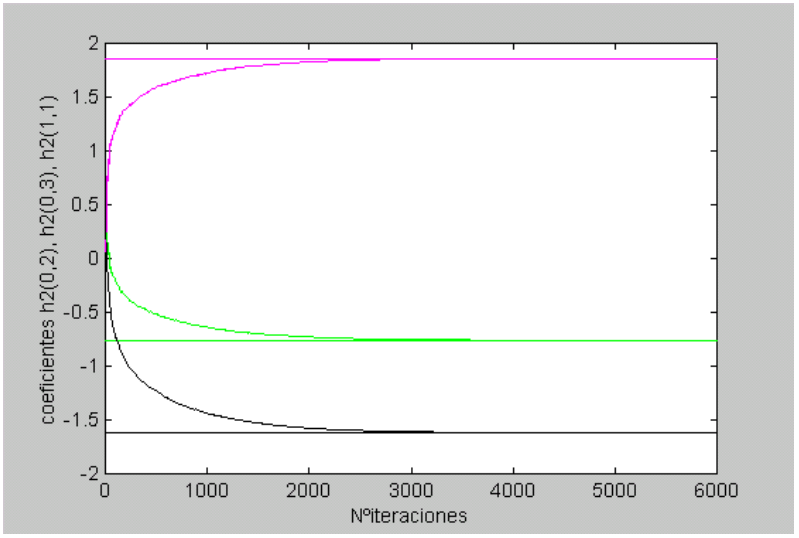
$h_2(0,0)$ y $\hat{h}_2(0,0)$

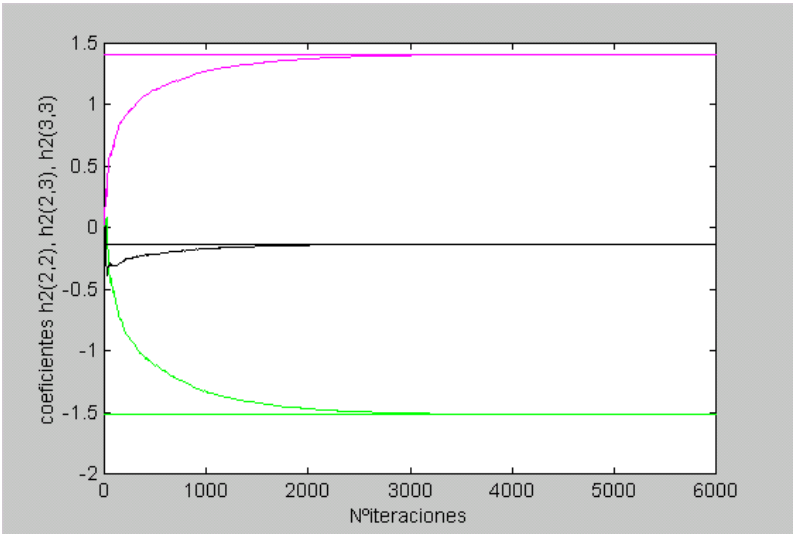
$h_2(0,1)$ y $\hat{h}_2(0,1)$

$h_2(0,2)$ y $\hat{h}_2(0,2)$

$h_2(0,3)$ y $\hat{h}_2(0,3)$

$h_2(1,1)$ y $\hat{h}_2(1,1)$



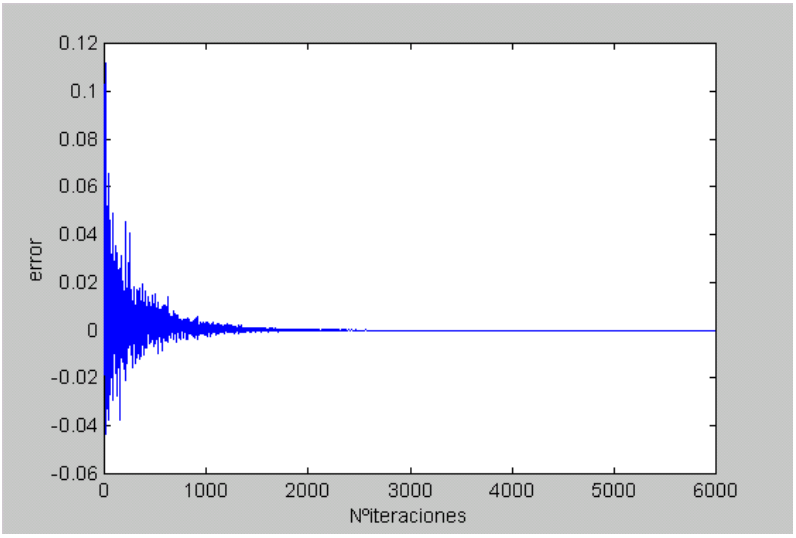

$h_2(2,2)$ y $\hat{h}_2(2,2)$

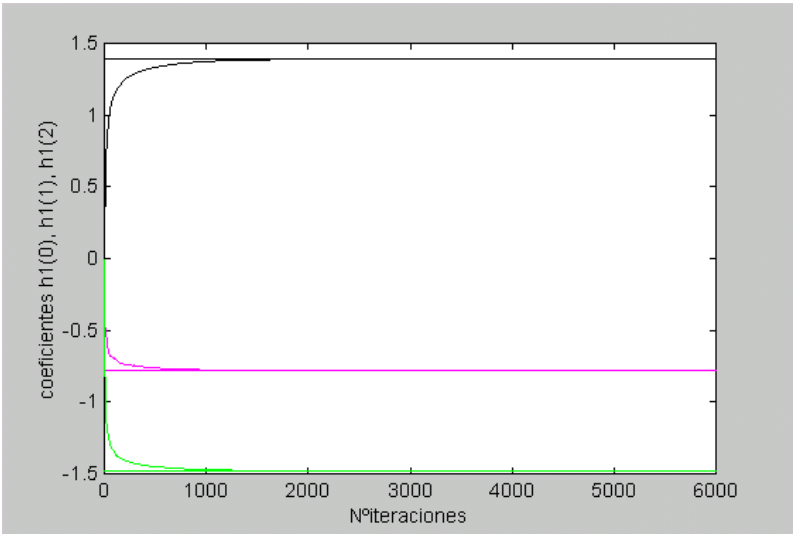
$h_2(2,3)$ y $\hat{h}_2(2,3)$

$h_2(3,3)$ y $\hat{h}_2(3,3)$

La convergencia se produce en las iteraciones 1056, 321 y 106 según los criterios del 3%, 10% y 15% respectivamente. Hay que destacar la buena adaptación de los coeficientes cuadráticos cosa que no ocurría en los dos algoritmos anteriores.

Se simula ahora con **m=0.5**. Las gráficas son:




— $h_1(0)$ y $\hat{h}_1(0)$

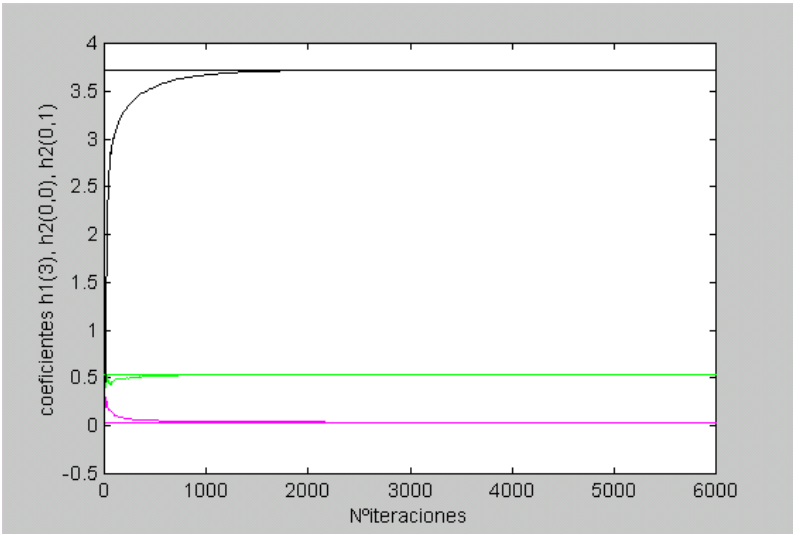
— $h_1(1)$ y $\hat{h}_1(1)$

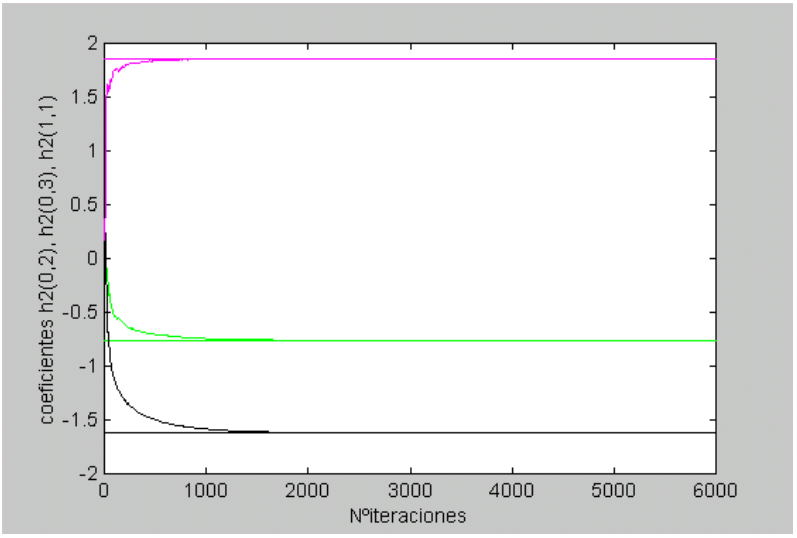
— $h_1(2)$ y $\hat{h}_1(2)$

— $h_1(3)$ y $\hat{h}_1(3)$

— $h_2(0,0)$ y $\hat{h}_2(0,0)$

— $h_2(0,1)$ y $\hat{h}_2(0,1)$






— $h_2(0,2)$ *y* $\hat{h}_2(0,2)$

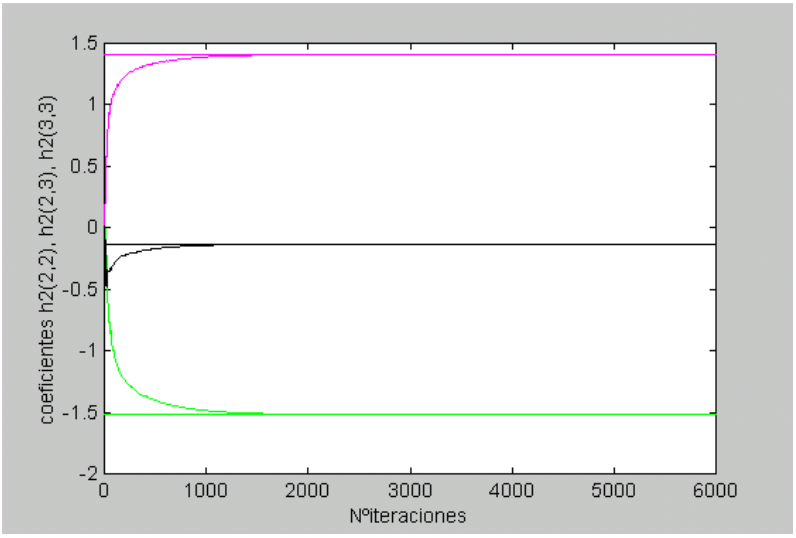
— $h_2(0,3)$ *y* $\hat{h}_2(0,3)$

— $h_2(1,1)$ *y* $\hat{h}_2(1,1)$

— $h_2(2,2)$ *y* $\hat{h}_2(2,2)$

— $h_2(2,3)$ *y* $\hat{h}_2(2,3)$

— $h_2(3,3)$ *y* $\hat{h}_2(3,3)$

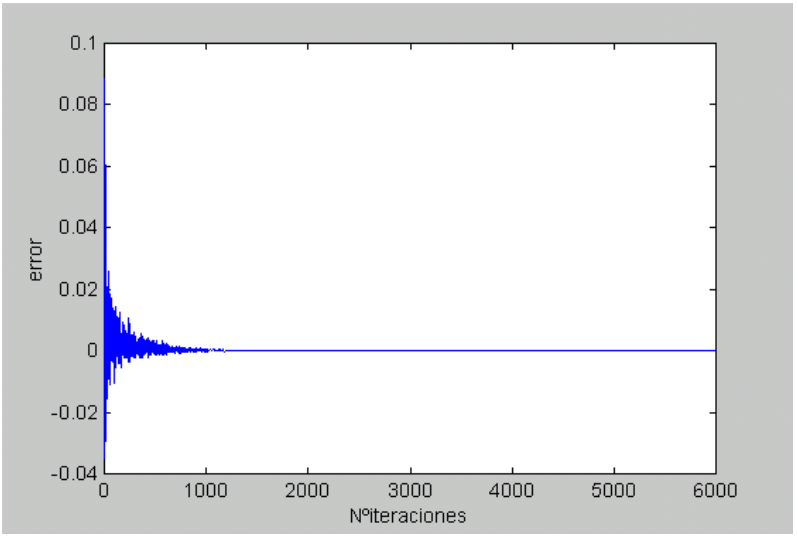


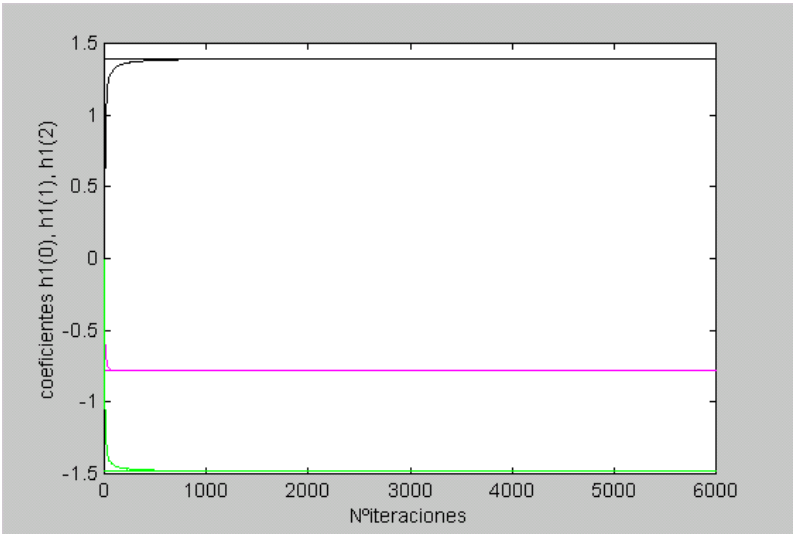

De acuerdo con los criterios del 3% y 10%, la convergencia se produce en las iteraciones 244 y 17 respectivamente. El criterio del 15% es demasiado flexible para este algoritmo, es decir, la señal de error siempre se mantiene por debajo de ese valor ($\pm 0.15$).

Estos resultados son muy superiores a los que se obtienen con los algoritmos anteriores. Además y como ya se ha dicho antes, en este algoritmo todos los coeficientes convergen bastante bien.

El inconveniente es el elevado número de operaciones que se necesitan.

Se aumenta un poco más el **paso de adaptación** hasta un valor igual a **0.7**. Los resultados son:




$h_1(0)$ y $\hat{h}_1(0)$

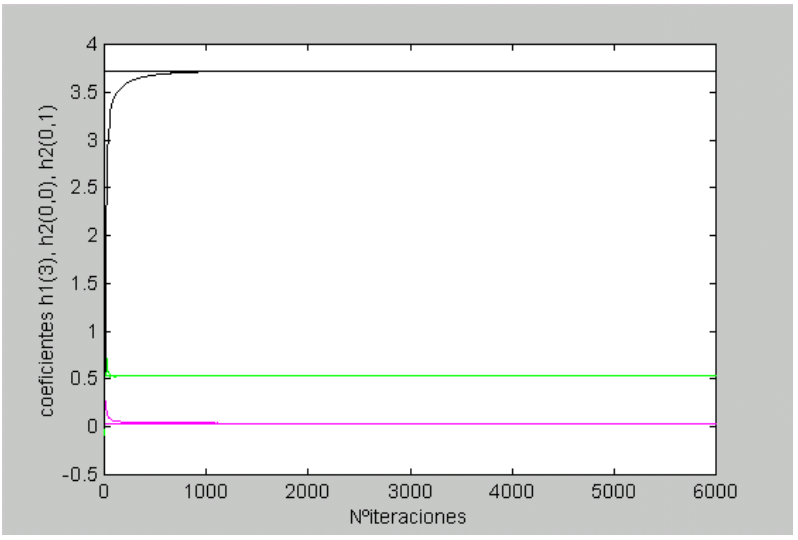
$h_1(1)$ y $\hat{h}_1(1)$

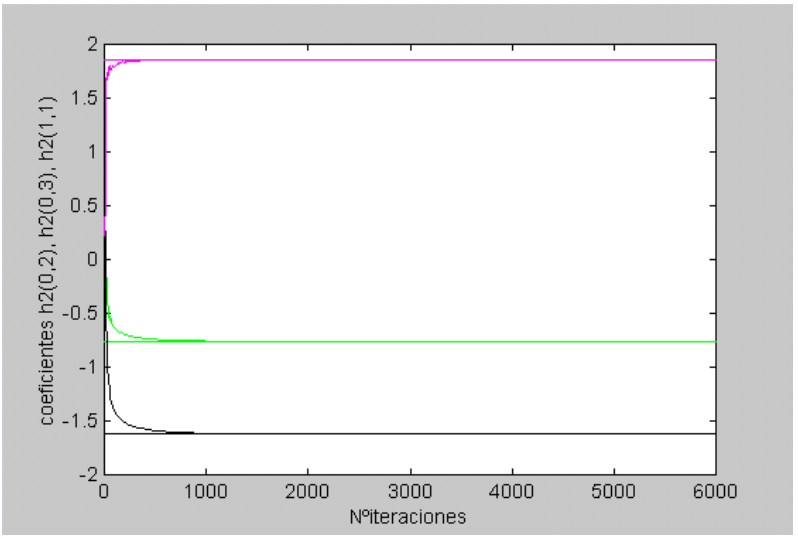
$h_1(2)$ y $\hat{h}_1(2)$

$h_1(3)$ y $\hat{h}_1(3)$

$h_2(0,0)$ y $\hat{h}_2(0,0)$

$h_2(0,1)$ y $\hat{h}_2(0,1)$






— $h_2(0,2)$ $y$ $\hat{h}_2(0,2)$

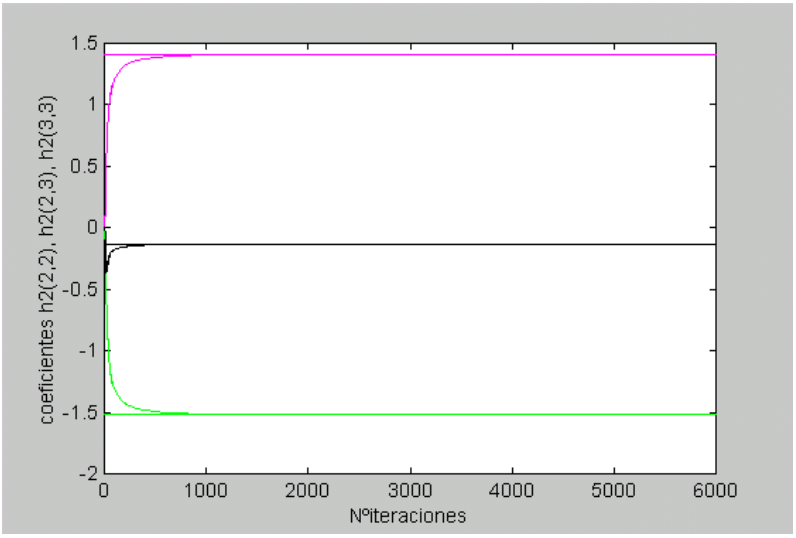
— $h_2(0,3)$ $y$ $\hat{h}_2(0,3)$

— $h_2(1,1)$ $y$ $\hat{h}_2(1,1)$

— $h_2(2,2)$ $y$ $\hat{h}_2(2,2)$

— $h_2(2,3)$ $y$ $\hat{h}_2(2,3)$

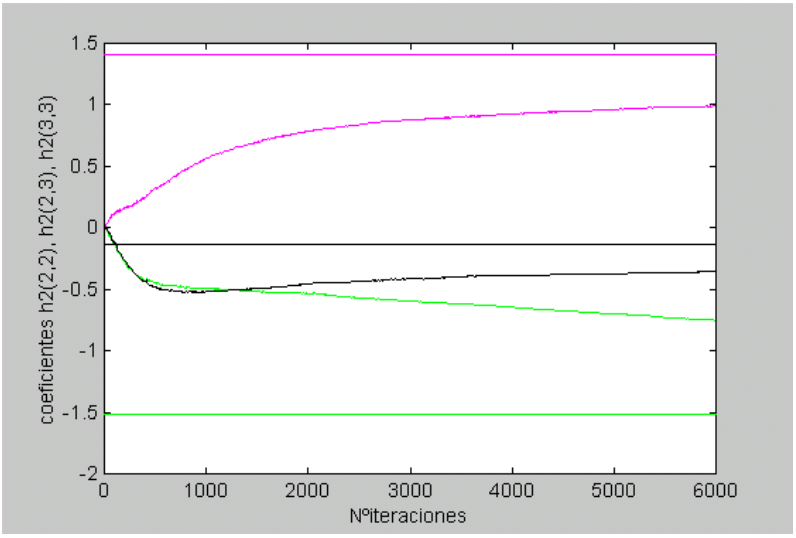
— $h_2(3,3)$ $y$ $\hat{h}_2(3,3)$




La convergencia se produce en la iteración 18 de acuerdo con el criterio del 3%. Para este valor del paso de adaptación, los otros dos criterios son demasiado flexibles, es decir, la señal de error siempre se mantiene por debajo de esos valores ($\pm 0.1$ $y$ $\pm 0.15$).

La convergencia es muy rápida y muy precisa superando con amplitud a los otros dos algoritmos.

Hay que volver a decir que todos los coeficientes convergen muy bien. Esto no ocurría en los algoritmos anteriores (sobre todo en el LMS). Para verlo mejor se muestra a continuación la convergencia de los tres últimos coeficientes en el caso óptimo del algoritmo LMS (μ=0.1) y en este caso del algoritmo RLS (μ=0.7):




— $h_2(2,2)$ *y* $\hat{h}_2(2,2)$

— $h_2(2,3)$ *y* $\hat{h}_2(2,3)$

— $h_2(3,3)$ *y* $\hat{h}_2(3,3)$

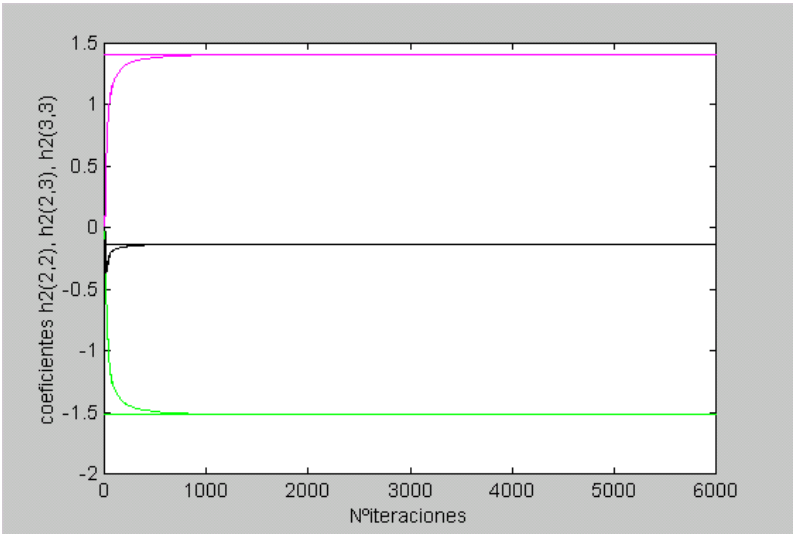
LMS

— $h_2(2,2)$ *y* $\hat{h}_2(2,2)$

— $h_2(2,3)$ *y* $\hat{h}_2(2,3)$

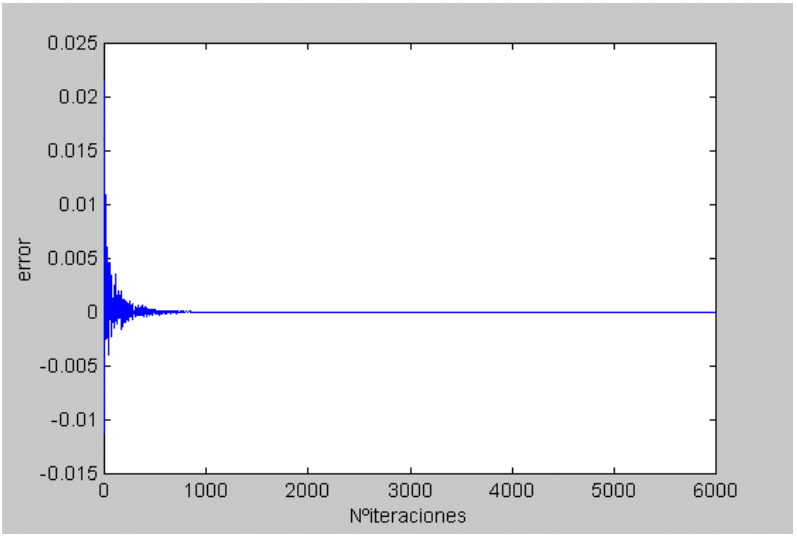
— $h_2(3,3)$ *y* $\hat{h}_2(3,3)$

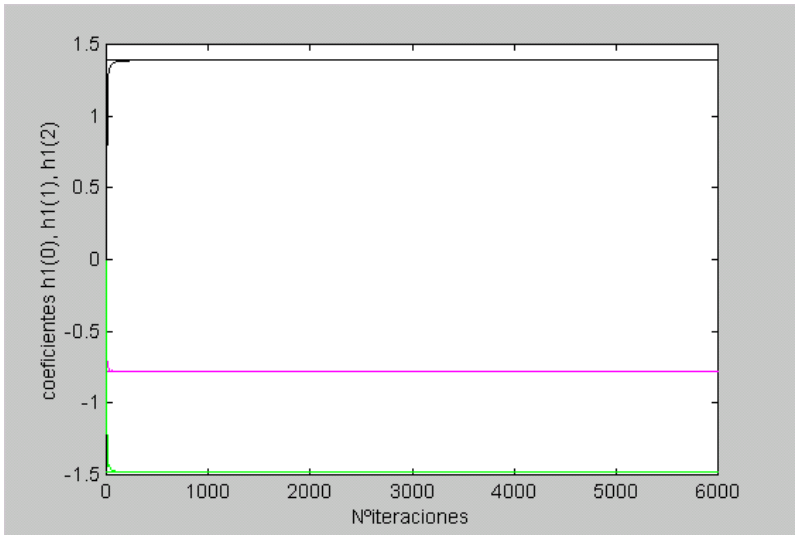
RLS




Se ve claramente la diferencia en la convergencia entre uno y otro algoritmo. Como se ha comentado antes, la velocidad de convergencia en el RLS es muy superior.

Se simula a continuación con un **paso de adaptación** igual a **0.9**. Los resultados son:




$h_1(0)$ y $\hat{h}_1(0)$

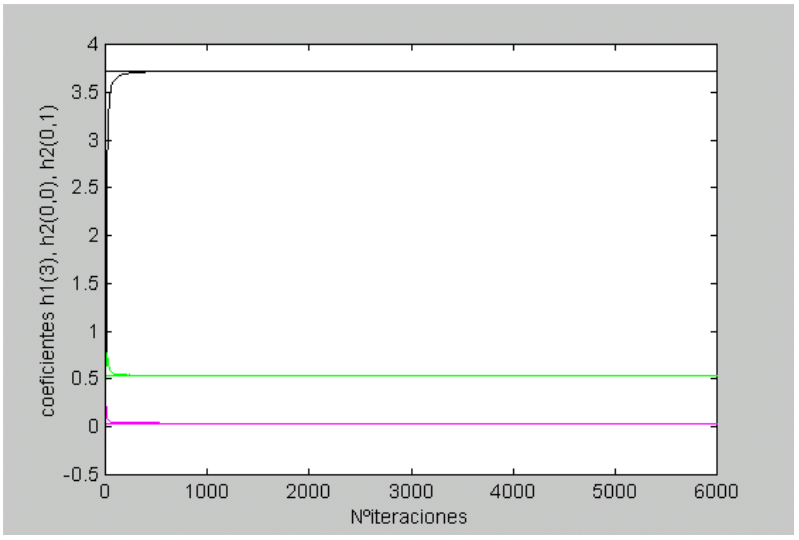
$h_1(1)$ y $\hat{h}_1(1)$

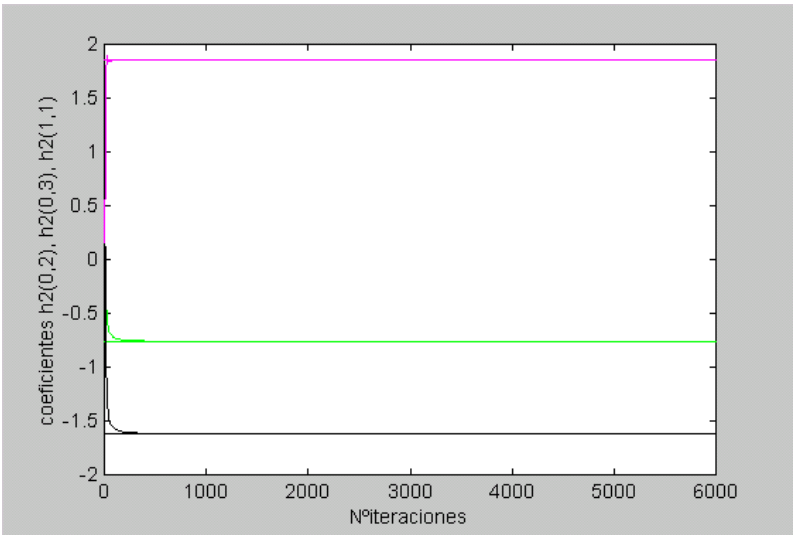
$h_1(2)$ y $\hat{h}_1(2)$

$h_1(3)$ y $\hat{h}_1(3)$

$h_2(0,0)$ y $\hat{h}_2(0,0)$

$h_2(0,1)$ y $\hat{h}_2(0,1)$






— $h_2(0,2)$ y $\hat{h}_2(0,2)$

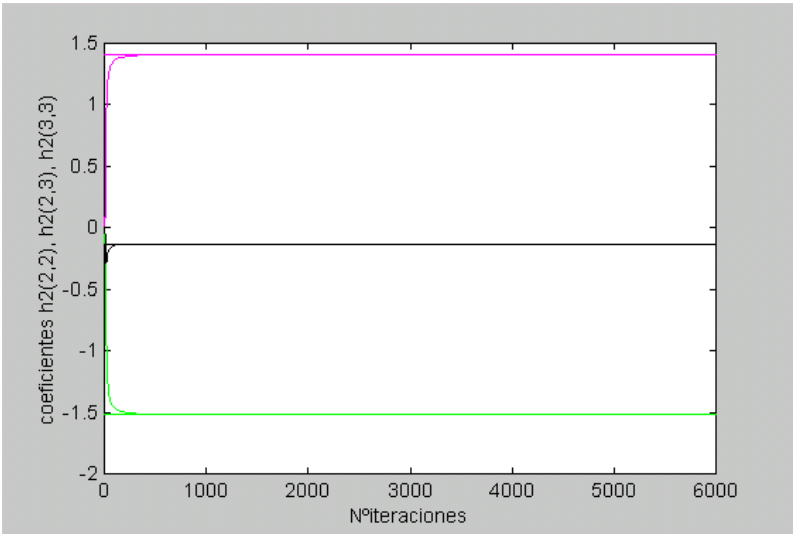
— $h_2(0,3)$ y $\hat{h}_2(0,3)$

— $h_2(1,1)$ y $\hat{h}_2(1,1)$

— $h_2(2,2)$ y $\hat{h}_2(2,2)$

— $h_2(2,3)$ y $\hat{h}_2(2,3)$

— $h_2(3,3)$ y $\hat{h}_2(3,3)$

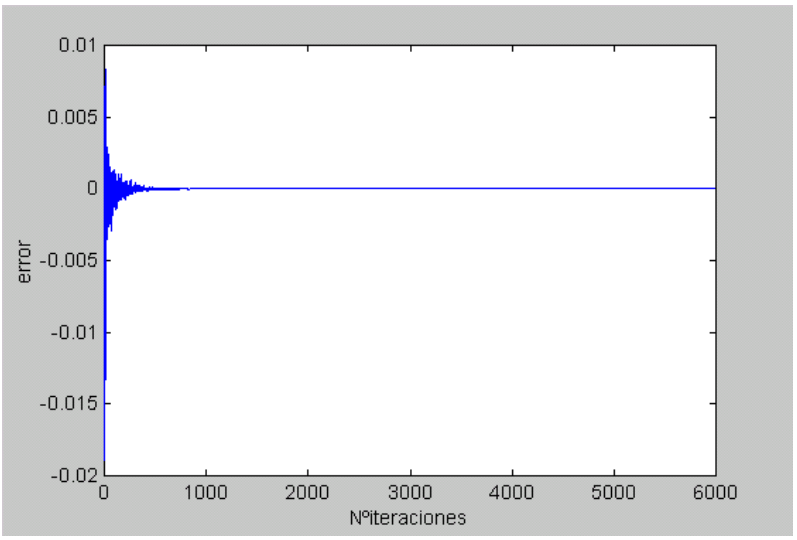


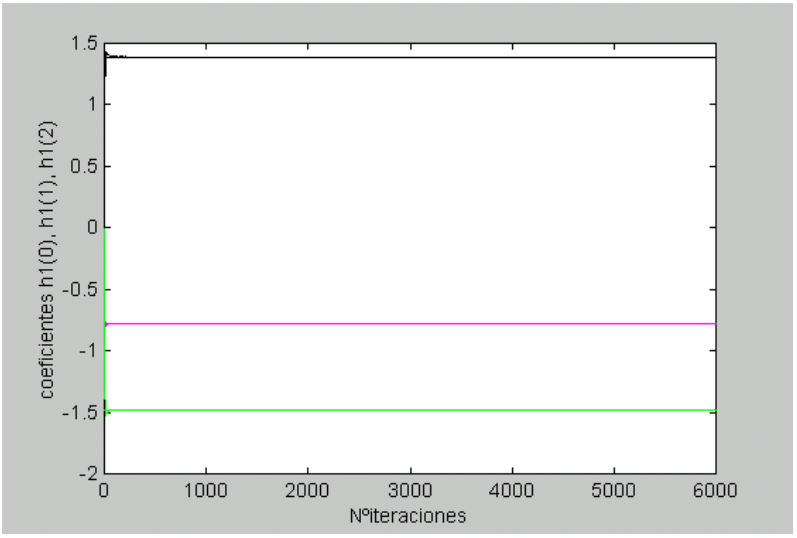

Los tres criterios son demasiado suaves para este caso. La señal de error siempre está por debajo de $\pm 0.03$.

Para este caso se han considerado tres criterios nuevos que sean más rigurosos. Así, se usan los criterios del 0.2% (señal de error por debajo de $\pm 0.002$), del 0.5% (señal de error por debajo de $\pm 0.005$) y del 1% (señal de error por debajo de $\pm 0.01$). Pues bien, la convergencia se produce en las iteraciones 106, 24 y 13 según los criterios del 0.2%, 0.5% y 1%.

Está claro que los resultados son mucho mejores que en los algoritmos anteriores.

Se simula de nuevo pero aumentando el **paso de adaptación**. Se considera un valor de **m=1.1**. Las gráficas son:




— $h_1(0)$ y $\hat{h}_1(0)$

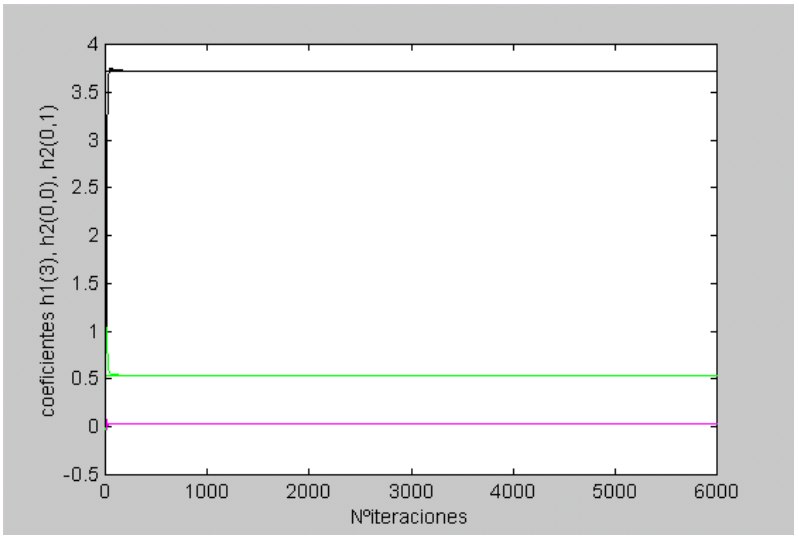
— $h_1(1)$ y $\hat{h}_1(1)$

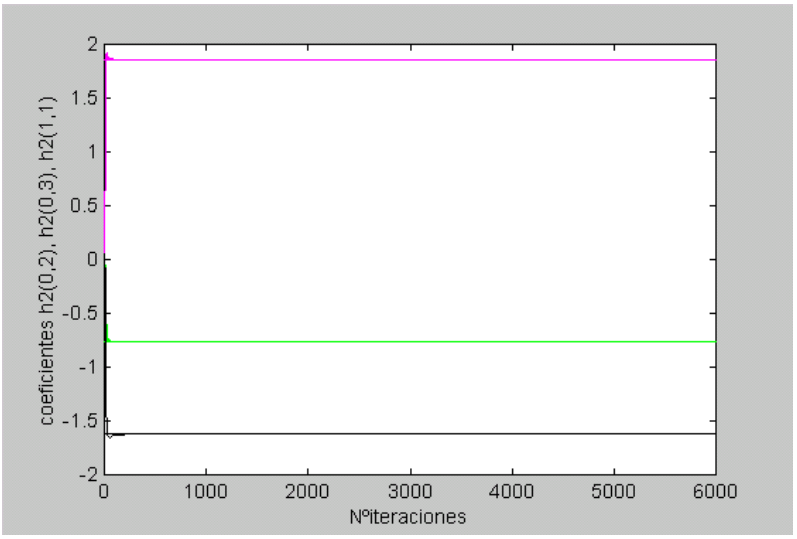
— $h_1(2)$ y $\hat{h}_1(2)$

— $h_1(3)$ y $\hat{h}_1(3)$

— $h_2(0,0)$ y $\hat{h}_2(0,0)$

— $h_2(0,1)$ y $\hat{h}_2(0,1)$






— $h_2(0,2)$ y $\hat{h}_2(0,2)$

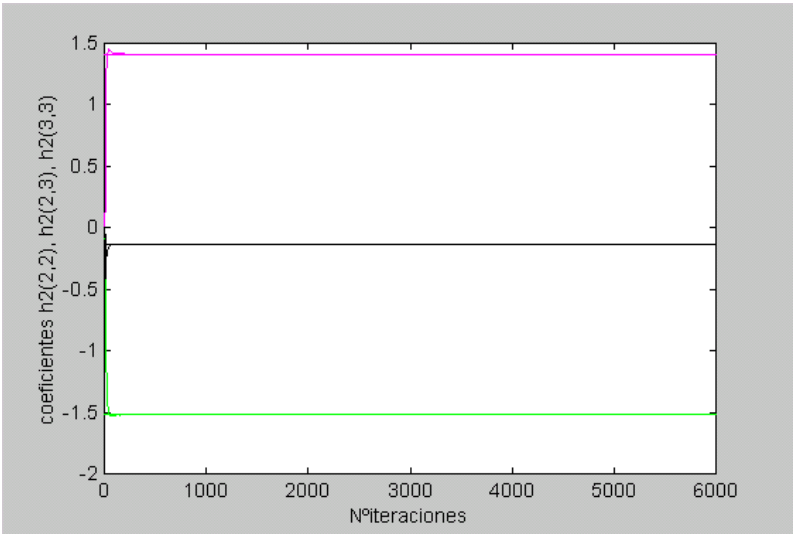
— $h_2(0,3)$ y $\hat{h}_2(0,3)$

— $h_2(1,1)$ y $\hat{h}_2(1,1)$

— $h_2(2,2)$ y $\hat{h}_2(2,2)$

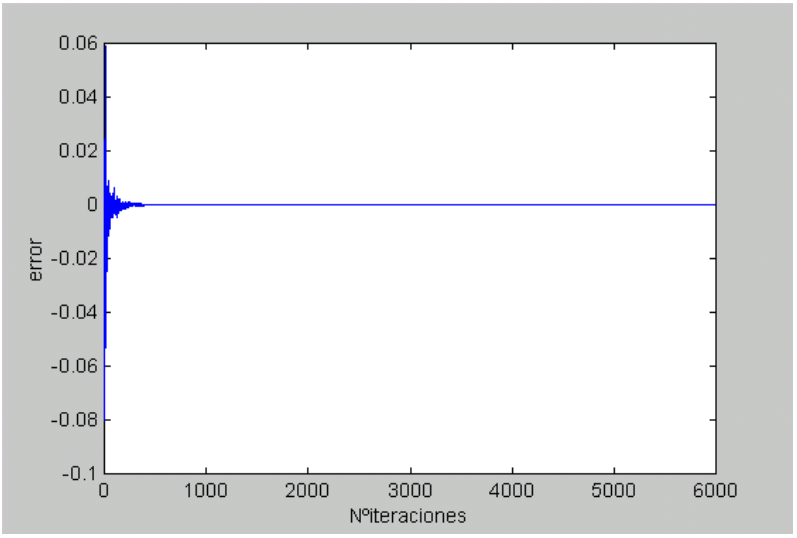
— $h_2(2,3)$ y $\hat{h}_2(2,3)$

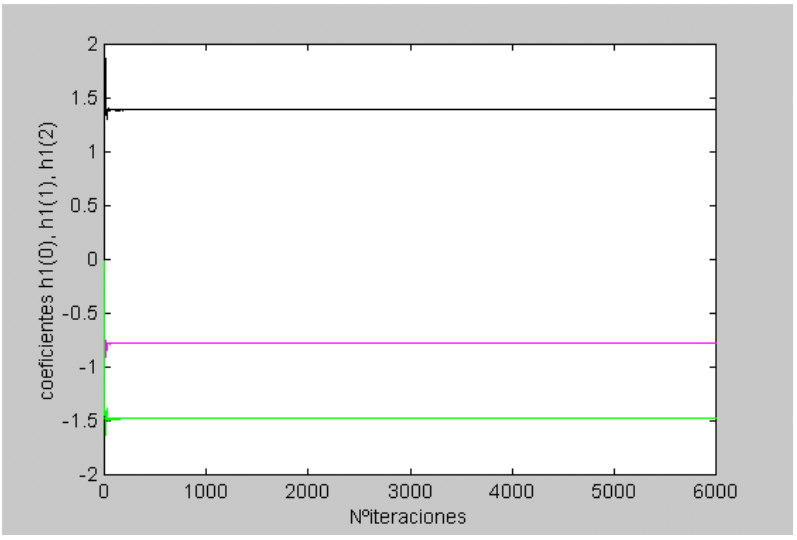
— $h_2(3,3)$ y $\hat{h}_2(3,3)$




Hay que utilizar los nuevos criterios de convergencia porque los anteriores son demasiados suaves. Así, la convergencia se produce en las iteraciones 65, 12 y 8 de acuerdo con los criterios del 0.2%, 0.5% y 1%. Los resultados continúan mejorando.

Se simula ahora con **m=1.3** y se obtiene:




— $h_1(0)$ y $\hat{h}_1(0)$

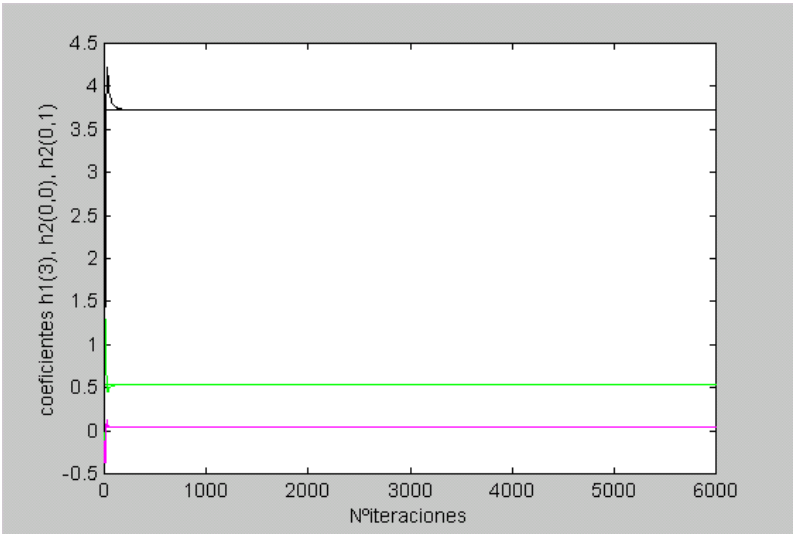
— $h_1(1)$ y $\hat{h}_1(1)$

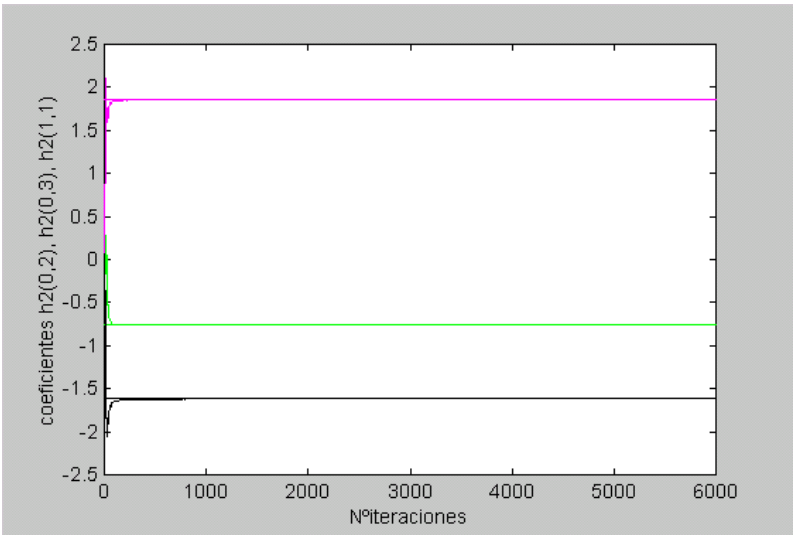
— $h_1(2)$ y $\hat{h}_1(2)$

— $h_1(3)$ y $\hat{h}_1(3)$

— $h_2(0,0)$ y $\hat{h}_2(0,0)$

— $h_2(0,1)$ y $\hat{h}_2(0,1)$






— $h_2(0,2)$ y $\hat{h}_2(0,2)$

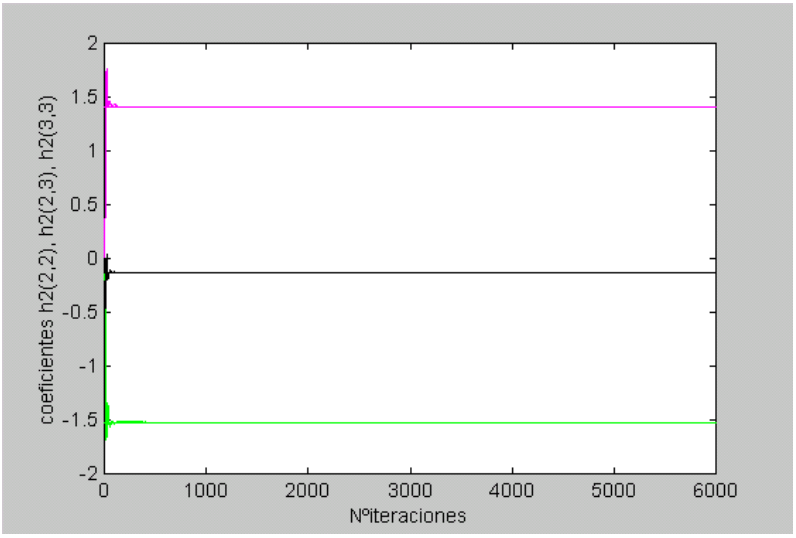
— $h_2(0,3)$ y $\hat{h}_2(0,3)$

— $h_2(1,1)$ y $\hat{h}_2(1,1)$

— $h_2(2,2)$ y $\hat{h}_2(2,2)$

— $h_2(2,3)$ y $\hat{h}_2(2,3)$

— $h_2(3,3)$ y $\hat{h}_2(3,3)$




Ahora la convergencia ya no es tan buena como en el caso anterior. Ésta se produce en la iteración 16 de acuerdo con el criterio del 3%.

Por tanto, se puede concluir que el valor óptimo para el paso de adaptación es de μ=1.1. Esto demuestra que es lógico que la mayoría de algoritmos RLS tomen el paso de adaptación igual a la unidad.

Vamos a resumir todos estos resultados en una tabla:

| m | criterio/ convergencia | criterio/ convergencia | criterio/ convergencia |
|---|---|---|---|
| 0.3 | 3%/1056 | 10%/321 | 15%/106 |
| 0.5 | 3%/244 | 10%/17 | 15%/0 |
| 0.7 | 3%/18 | 10%/0 | 15%/0 |
| 0.9 | 3%/0<br>0.2%/106 | 10%/0<br>0.5%/24 | 15%/0<br>1%/13 |
| **1.1** | 3%/0<br>**0.2%/65** | 10%/0<br>**0.5%/12** | 15%/0<br>**1%/8** |
| 1.3 | 3%/16<br>0.2%/167 | 10%/0<br>0.5%/92 | 15%/0<br>1%/47 |

Resultados con el algoritmo RLS

Vamos a hacer a continuación una comparativa entre los tres algoritmos respecto al número de operaciones. El LMS utiliza 552.3 operaciones/iteración/experimento, el LMS normalizado, 622.3 operaciones/iteración/experimento y el RLS, 7908.3 operaciones/iteración/experimento. A priori, cabe pensar que el RLS es bastante peor. Sin embargo, este análisis depende del número de iteraciones que sean necesarias en cada algoritmo. Así, vamos a resumir en una tabla los casos óptimos para cada algoritmo con el número de operaciones necesarias en cada caso:

| algoritmo | m | criterio | iteraciones convergencia | operaciones (millones) |
|---|---|---|---|---|
| LMS | 0.1 | 3% | 5857 | 161.7 |
| | | 10% | 755 | 20.8 |
| | 0.07 | 15% | 152 | 4.2 |
| LMS normalizado | 0.5 | 3% | 823 | 25.6 |
| | | 10% | 117 | 3.6 |
| | | 15% | 32 | 1 |
| RLS | 1.1 | 0.2% | 65 | 25.7 |
| | | 0.5% | 12 | 4.7 |
| | | 1% | 8 | 3.2 |

Claramente es preferible el algoritmo RLS puesto que al ser mucho más rápido, el número de operaciones necesarias para lograr la convergencia es menor que el que hace falta para lograrla con los otros dos algoritmos. Además, la convergencia es mucho más exacta.

## 5.3.- Conclusiones

Se han realizado distintas simulaciones con dos ejemplos de orden dos. El primero de ellos tenía sólo tres coeficientes (dos de ellos lineales) mientras que el segundo era un poco más complejo. Además, la memoria del sistema era de N=3 mientras que en el segundo era de N=4.

En el primer ejemplo se han estudiado los tres algoritmos considerando muchas variantes. A saber: presencia o no de ruido, consideración de distintas medias para el proceso de entrada y varios valores para el paso de adaptación.

Respecto a esta última variante, se ha observado que, en general, el LMS necesita un paso de adaptación menor que el LMS normalizado, que, a su vez, utiliza uno menor que el del RLS.

La presencia de ruido afecta bastante a los algoritmos LMS y LMS normalizado. De hecho, al aumentar el ruido, el criterio de convergencia se tiene que flexibilizar considerablemente. Además, el paso de adaptación debe irse reduciendo conforme aumenta el ruido, es decir, la adaptación se debe hacer más lenta para evitar que en el paso de una iteración a otra, el algoritmo diverja.

Con el algoritmo RLS esto no ocurre. No es necesario suavizar mucho los criterios de convergencia ni ir disminuyendo el paso de adaptación. Es decir, es bastante inmune al ruido.

La variación de la media del proceso de entrada perjudica considerablemente otra vez a los dos primeros algoritmos. En este caso, el algoritmo LMS normalizado se comporta un poco mejor que el LMS. Esto es lógico si se tiene en cuenta que en el primero se normaliza respecto a la señal de entrada, es decir, se le hace un poco más independiente respecto a dicho proceso de entrada. El algoritmo RLS es nuevamente el que mejor se comporta en cuanto a velocidad de convergencia ante un proceso de entrada con una media elevada. Esto se debe a que es un algoritmo más robusto que los otros dos.

No obstante, una posible solución para cuando la media del proceso de entrada sea distinta de cero sería: se identifica el sistema suponiendo un proceso de entrada de media cero, que sí funciona bien; una vez que se tiene el

sistema, se trabaja con éste y con el proceso de entrada que nos dieron.

En condiciones ideales (ausencia de ruido y proceso de entrada de media cero), el algoritmo LMS normalizado es el que mejor se comporta en cuanto a velocidad de convergencia.

Sin embargo, para este ejemplo tan sencillo, el número de operaciones necesarias para cada algoritmo es muy diferente siendo muy superior en el RLS. Esta diferencia será rentable en los casos en que haya ruido y/o el proceso de entrada tenga media distinta de cero. Esto es así porque el número de operaciones se ve compensada con el número de iteraciones necesarias para alcanzar la convergencia. Este número es muy elevado en condiciones desfavorables en los algoritmos LMS y LMS normalizado.

En el segundo ejemplo, se estudia un sistema más complejo, es decir, con más memoria y con más coeficientes. Se intenta buscar un paso de adaptación óptimo para cada algoritmo y compararlos en cuanto a velocidad de convergencia y número de operaciones.

El número de operaciones es mucho más elevado para el RLS que para los otros dos algoritmos aunque tanto la velocidad como la precisión de la convergencia es muy superior en el RLS que en los otros dos. Hay que combinar ambos factores para elegir el mejor algoritmo.

Tal y como se vio antes es mejor usar el algoritmo RLS puesto que además de converger con más rapidez y precisión, el número de operaciones necesarias para ello es menor que en los otros dos algoritmos.

Por tanto, cuando los sistemas se complican, aparece ruido y/o el proceso de entrada tiene media distinta de cero es conveniente usar el algoritmo RLS.

# CAPÍTULO 6.

# INVESTIGACIONES Y APLICACIONES FUTURAS

Los filtros adaptativos no lineales abarcan un campo muy amplio y aquí sólo se ha tratado una pequeña parte.

Se está trabajando mucho en los filtros adaptativos con ecuaciones en diferencias no lineales y recursivas. Se intentan buscar modelos basados en esto que no tengan tantos problemas con la realimentación.

También se trabaja mucho con sistemas basados en una estructura lattice. Este tipo de estructura tiene buenas propiedades y se estudian posibles implementaciones de filtros adaptativos usando dichas estructuras.

Existen algoritmos RLS rápidos que reducen el número de operaciones necesarias respecto al RLS.

Los filtros adaptativos se usan mucho en predicción de imágenes, tal y como ya se ha dicho. Vamos a ver con más detalle esta última aplicación y algunas de las técnicas que se usan basándose en estos filtros.

Se utilizan codificaciones DPCM y transformadores híbridos como métodos efectivos para reducir la tasa de bit en una transmisión digital de imágenes. Ambos métodos usan un predictor y su consiguiente código de predicción de error. Sin embargo, la eficiencia de los predictores lineales fijos es bastante limitada cuando la señal de imagen está bastante lejos de ser estacionaria. Para solventar este problema, se han propuesto predictores lineales adaptativos, aunque su habilidad para trabajar con variaciones temporales y espaciales abruptas es limitada a causa de sus características paso de baja. Un predictor no lineal que sea capaz de reunir información de momentos estadísticos de alto

orden y de aguantar efectos de paso alta será más eficiente. De hecho, una clase de predictores no lineales, basado en un filtro cuadrático, ofrece resultados interesantes sobre todo cuando hay cierta cantidad de movimiento presente en la imagen. En este caso, la parte cuadrática del operador compensa los efectos de alisado producidos por un predictor lineal convencional.

Una primera técnica consiste en subdividir cada imagen en bloques más pequeños (de 16 por 16 píxeles, por ejemplo) los cuales se clasifican como “activos” o “no activos” dependiendo del valor que de una función que mide la diferencia cuadrática media entre bloques homólogos. Para cada bloque activo, el predictor cuadrático óptimo se deriva y se aplica.

También se ha propuesto un algoritmo de adaptación píxel a píxel basado en el criterio LMS. Para adaptarse rápido a las variaciones de la señal de entrada y que no sea muy complejo, se utilizan sólo los tres píxeles previos en las direcciones espaciotemporales. Además, se ha demostrado que el mejor comportamiento se obtiene usando un filtro adaptativo lineal en las zonas más despejadas y un filtro cuadrático en las zonas más llenas. La selección de uno de los predictores se hace mediante la propuesta descrita por Prahbu. De acuerdo con esto, se elige el predictor que presente un menor valor absoluto para el error de predicción para una fracción concreta de píxeles.

También se están investigando mucho los filtros adaptativos polinomiales en el modelado de canales de comunicación. En muchos casos son necesarios ecualizadores no lineales para conseguir un buen comportamiento en canales de alta velocidad. Se utilizan también en los enlaces digitales por satélite ya que los amplificadores del satélite a menudo se llevan al punto de saturación y ahí existen muchas no linealidades.

# CAPÍTULO 7.

# PROGRAMAS

A continuación se muestran los seis programas en Matlab que se han utilizado (tres por cada ejemplo) para los tres algoritmos. En general son análogos los programas para ambos ejemplos. Sólo cambia la representación de la señal de entrada y el valor del parámetro N, que es la memoria del sistema.

## 6.1.- Ejemplo 1

- **LMS:**

```
%%%%%Identificar este sistema utilizando LMS-Volterra:
```

$$y(n) = 1 + x(n) + x^2(n)$$

```
%%%Se simulan 50 experimentos independientes
%%%Se representan las gráficas siguientes:
%%%       vector de error
%%%       convergencia de cada uno de los tres coeficientes
%%%Ocasionalmente se representa:
%%%       ||V_L(n)|| (coeficientes lineales)
%%%       ||V_Q(n)|| (coeficientes cuadráticos)

flops(0) %para saber el número de operaciones
Nexpe=50; %número de experimentos independientes para luego
          %obtener un valor medio
emean=0;
Niter=3000; %número de iteraciones en el algoritmo
N=3;
```

```
Hmean=zeros(Niter,N*(N+3)/2);
H0mean=zeros(Niter,1);
for expe=1:Nexpe

   %x[n] variable aleatoria con varianza 1 y media 0,1 ó 10
   Rx=1;
   media=[0,1,10];
   m=1;
    randn('state',sum(100*clock)); %para que cada vez tome
                                   %un valor distinto
   x=sqrt(Rx)*randn(Niter,1)+media(m);

   %vector de entrada
   %cada fila es para un instante de tiempo
   N=3; %N-1=2 elementos de retraso
   %primeras N-1 filas
   X(1,1:N*(N+3)/2)=[x(1),zeros(1,2),x(1)^2,zeros(1,5)];
   X(2,1:N*(N+3)/2)=[x(2),x(1),0,x(2)^2,x(2)*x(1),0,x(1)^2,
    zeros(1,2)];

   %resto de filas
   for k=3:Niter
      X(k,1:N*(N+3)/2)=[x(k),x(k-1),x(k-2),x(k)^2,x(k)*x(k-
       1),x(k)*x(k-2),x(k-1)^2,x(k-1)*x(k-2),x(k-2)^2];
   end


   %vector de coeficientes
   H=[1,0,0,1,zeros(1,5)];
   H0=1;

   %obtención de la salida del sistema:
   %y[n]=1+x[n]+(x[n]^2)
   y(1)=H0+H(1)*x(1)+H(4)*x(1)^2;
   y(2)=H0+H(1)*x(2)+H(2)*x(1)+H(4)*x(2)^2+H(5)*x(2)*x(1)+
    H(7)*x(1)^2;
   for k=3:Niter
      y(k)=H0+H(1)*x(k)+H(2)*x(k-1)+H(3)*x(k-
       2)+H(4)*x(k)^2+H(5)*x(k)*x(k-1)+H(6)*x(k)*x(k-
       2)+H(7)*x(k-1)^2+H(8)*x(k-1)*x(k-2)+H(9)*x(k-2)^2;
   end

   %obtención de la señal deseada
   Rw=[0,0.05,1];
   i=3; %para elegir la varianza del ruido
    randn('state',sum(100*clock)); %para que cada vez tome
                                   %un valor distinto
   w=sqrt(Rw(i))*randn(Niter,1);
   for k=1:Niter
      d(k)=y(k)+w(k);
   end
```

```
   %algoritmo LMS
   u=0.03; %paso de adaptación
   Hest=zeros(Niter,N*(N+3)/2);
   H0est=zeros(Niter,1);
   for j=1:N*(N+3)/2
      udiag(j)=u;
   end
   umatr=diag(udiag); %matriz diagonal con u en la diagonal
   for n=1:Niter
      e(n)=d(n)-H0est(n)-Hest(n,:)*X(n,:)';
      Hest(n+1,:)=Hest(n,:)+(umatr*X(n,:)'*e(n))';
      H0est(n+1)=H0est(n)+u*e(n);
   end
   emean=emean+e;
   Hmean=Hmean+Hest(1:Niter,:);
   H0mean=H0mean+H0est(1:Niter);
end

e=emean./Nexpe;
Hest=Hmean./Nexpe;
H0est=H0mean./Nexpe;

%ecuaciones para las gráficas ocasionales
%por un lado los coeficientes lineales y por otro los
%cuadráticos
%for n=1:Niter
%   for j=1:N
%      aux(j)=(Hest(n,j)-H(j))^2;
%      temp(j)=H(j)^2;
%   end
%   lin(n)=10*log(aux/temp);
%   t=1;
%   for j=N+1:N*(N+3)/2
%      aux(t)=(Hest(n,j)-H(j))^2;
%      temp(t)=H(j)^2;
%      t=t+1;
%   end
%   cua(n)=10*log(aux/temp);
%end

%gráficas
n=1:Niter;
plot(e),xlabel('Nºiteraciones'),ylabel('error con varianza
  1')
pause
plot(n,H0est(n),'b',n,H0,'r'),xlabel('Nº    iteraciones'),
  ylabel('coeficiente h0')
pause
plot(n,Hest(n,1),'b',n,H(1),'r'),xlabel('Nº  iteraciones'),
  ylabel('coeficiente h1')
```

```
pause
plot(n,Hest(n,4),'b',n,H(4),'r'),xlabel('Nº  iteraciones'),
  ylabel('coeficiente h2')
pause
%plot(lin),title('coeficientes lineales'),xlabel('Nº
  iteraciones'),ylabel('||VL(n)||')
%pause
%plot(cua),title('coeficientes cuadráticos'),xlabel('Nº
%  iteraciones'),ylabel('||VQ(n)||')
%pause

%número de operaciones
oper=flops


%convergencia
z=1;
t=1;
for g=1:Niter
  if(abs(e(g))>0.5)
     result(z)=g;
     z=z+1;
  end
  if(abs(e(g))>0.55)
     res(t)=g;
     t=t+1;
  end
end
convergencia50=max(result)
convergencia55=max(res)
```

- **LMS normalizado:**

```
%%%%%Identificar este sistema utilizando LMS-Volterra
%%%%%normalizado:
```

$$y(n) = 1 + x(n) + x^2(n)$$

```
%%%Se simulan 50 experimentos independientes
%%%Se representan las gráficas siguientes:
%%%       vector de error
%%%       convergencia de cada uno de los tres coeficientes

flops(0) %sirve para contar el número de operaciones en
         %punto flotante
Nexpe=50; %número de experimentos independientes para luego
          %obtener un valor medio
emean=0;
Niter=3000; %número de iteraciones en el algoritmo
N=3;
Hmean=zeros(Niter,N*(N+3)/2);
H0mean=zeros(Niter,1);
for expe=1:Nexpe

   %x[n] variable aleatoria con varianza 1 y media 0,1 ó 10
   Rx=1;
    randn('state',sum(100*clock)); %para que cada vez tome
                                   %un valor distinto
   x=sqrt(Rx)*randn(Niter,1)+media(m);

   %vector de entrada
   %cada fila es para un instante de tiempo
   N=3; %N-1=2 elementos de retraso
   %primeras N-1 filas
   X(1,1:N*(N+3)/2)=[x(1),zeros(1,2),x(1)^2,zeros(1,5)];
   X(2,1:N*(N+3)/2)=[x(2),x(1),0,x(2)^2,x(2)*x(1),0,x(1)^2,
     zeros(1,2)];

   %resto de filas
   for k=3:Niter
      X(k,1:N*(N+3)/2)=[x(k),x(k-1),x(k-2),x(k)^2,x(k)*x(k-
        1),x(k)*x(k-2),x(k-1)^2,x(k-1)*x(k-2),x(k-2)^2];
   end


   %vector de coeficientes
   H=[1,0,0,1,zeros(1,5)];
   H0=1;
```

```
   %obtención de la salida del sistema:
   %y[n]=1+x[n]+(x[n]^2)
   y(1)=H0+H(1)*x(1)+H(4)*x(1)^2;
   y(2)=H0+H(1)*x(2)+H(2)*x(1)+H(4)*x(2)^2+H(5)*x(2)*x(1)+
      H(7)*x(1)^2;
   for k=3:Niter
      y(k)=H0+H(1)*x(k)+H(2)*x(k-1)+H(3)*x(k-
        2)+H(4)*x(k)^2+H(5)*x(k)*x(k-1)+H(6)*x(k)*x(k-
        2)+H(7)*x(k-1)^2+H(8)*x(k-1)*x(k-2)+H(9)*x(k-2)^2;
   end

   %obtención de la señal deseada
   Rw=[0,0.05,1];
   i=3; %para elegir la varianza del ruido
    randn('state',sum(100*clock)); %para que cada vez tome
                                   %un valor distinto
   w=sqrt(Rw(i))*randn(Niter,1);
   for k=1:Niter
      d(k)=y(k)+w(k);
   end

   %algoritmo LMS normalizado
   u=0.5; %paso de adaptación
   Hest=zeros(Niter,N*(N+3)/2);
   H0est=zeros(Niter,1);
   for j=1:N*(N+3)/2
      udiag(j)=u;
   end
   umatr=diag(udiag); %matriz diagonal con u en la diagonal
   for n=1:Niter
      e(n)=d(n)-H0est(n)-Hest(n,:)*X(n,:)';
      Hest(n+1,:)=Hest(n,:)+(umatr*X(n,:)'*e(n))'/
            (norm(X(n,:)));
      H0est(n+1)=H0est(n)+u*e(n);
   end
   emean=emean+e;
   Hmean=Hmean+Hest(1:Niter,:);
   H0mean=H0mean+H0est(1:Niter);
end

e=emean./Nexpe;
Hest=Hmean./Nexpe;
H0est=H0mean./Nexpe;
```

```
%gráficas
n=1:Niter;
plot(e),xlabel('Nºiteraciones'),ylabel('error con varianza
  1')
pause
plot(n,H0est(n),'b',n,H0,'r'),xlabel('Nº    iteraciones'),
  ylabel('coeficiente h0')
pause
plot(n,Hest(n,1),'b',n,H(1),'r'),xlabel('Nº  iteraciones'),
  ylabel('coeficiente h1')
pause
plot(n,Hest(n,4),'b',n,H(4),'r'),xlabel('Nº  iteraciones'),
  ylabel('coeficiente h2')
pause

%número de operaciones
oper=flops


%convergencia
z=1;
t=1;
for g=1:Niter
  if(abs(e(g))>0.5)
     result(z)=g;
     z=z+1;
  end
  if(abs(e(g))>0.55)
     res(t)=g;
     t=t+1;
  end
end
convergencia50=max(result)
convergencia55=max(res)
```

- **RLS:**

```
%%%%%Identificar este sistema utilizando RLS-Volterra:
```

$$y(n) = 1 + x(n) + x^2(n)$$

```
%%%Se simulan 50 experimentos independientes
%%%Se representan las gráficas siguientes:
%%%       vector de error
%%%       convergencia de cada uno de los tres coeficientes

flops(0) %para saber el número de operaciones
Nexpe=50; %número de experimentos independientes para luego
          %obtener un valor medio
emean=0;
N=3;
Niter=3000; %número de iteraciones en el algoritmo
N=3;
Hmean=zeros(Niter,N*(N+3)/2);
H0mean=zeros(Niter,1);
for expe=1:Nexpe

   %x[n] variable aleatoria con varianza 1 y media 0,1 ó 10
   Rx=1;
   media=[0,1,10];
   m=1;
    randn('state',sum(100*clock)); %para que cada vez tome
                                   %un valor distinto
   x=sqrt(Rx)*randn(Niter,1)+media(m);

   %vector de entrada
   %cada fila es para un instante de tiempo
   N=3; %N-1=2 elementos de retraso
   %primeras N-1 filas
   X(1,1:N*(N+3)/2)=[x(1),zeros(1,2),x(1)^2,zeros(1,5)];
   X(2,1:N*(N+3)/2)=[x(2),x(1),0,x(2)^2,x(2)*x(1),0,x(1)^2,
     zeros(1,2)];

   %resto de filas
   for k=3:Niter
      X(k,1:N*(N+3)/2)=[x(k),x(k-1),x(k-2),x(k)^2,x(k)*x(k-
         1),x(k)*x(k-2),x(k-1)^2,x(k-1)*x(k-2),x(k-2)^2];
   end


   %vector de coeficientes
   H=[1,0,0,1,zeros(1,5)];
   H0=1;

   %obtención de la salida del sistema:
   %y[n]=1+x[n]+(x[n]^2)
```

```
y(1)=H0+H(1)*x(1)+H(4)*x(1)^2;
y(2)=H0+H(1)*x(2)+H(2)*x(1)+H(4)*x(2)^2+H(5)*x(2)*x(1)+
     H(7)*x(1)^2;
for k=3:Niter
   y(k)=H0+H(1)*x(k)+H(2)*x(k-1)+H(3)*x(k-
      2)+H(4)*x(k)^2+H(5)*x(k)*x(k-1)+H(6)*x(k)*x(k-
      2)+H(7)*x(k-1)^2+H(8)*x(k-1)*x(k-2)+H(9)*x(k-2)^2;
end

%obtención de la señal deseada
Rw=[0,0.05,1];
i=3; %para elegir la varianza del ruido
randn('state',sum(100*clock)); %para que cada vez tome
                               %un valor distinto
w=sqrt(Rw(i))*randn(Niter,1);
for k=1:Niter
   d(k)=y(k)+w(k);
end

%algoritmo RLS
u=0.7; %paso de adaptación
for j=1:N*(N+3)/2
   udiag(j)=u;
end
umatr=diag(udiag); %matriz diagonal con u en la diagonal
lambda=0.995;
invlam=1/lambda;
Hest=zeros(Niter,N*(N+3)/2);
H0est=zeros(Niter,1);
delta=0.001;
invC=(1/delta)*eye(N*(N+3)/2);

%primera iteración
K(1,1:N*(N+3)/2)=((invlam*invC*X(1,:)')/
    (1+invlam*X(1,:)* invC*X(1,:)'))';
eps(1)=d(1)-0-zeros(1,N*(N+3)/2)*X(1,:)';
Hest(1,:)=zeros(1,N*(N+3)/2)+(umatr*K(1,:)'*eps(1))';
H0est(1)=0+u*eps(1);
invC=invlam*invC-invlam*K(1,:)'*X(1,:)*invC;
e(1)=d(1)-H0est(1)-Hest(1,:)*X(1,:)';

%resto de iteraciones
for n=2:Niter
   K(n,1:N*(N+3)/2)=((invlam*invC*X(n,:)')/
       (1+invlam*X(n,:)*invC*X(n,:)'))';
   eps(n)=d(n)-H0est(n-1)-Hest(n-1,:)*X(n,:)';
   Hest(n,:)=Hest(n-1,:)+(umatr*K(n,:)'*eps(n))';
   H0est(n)=H0est(n-1)+u*eps(n);
   invC=invlam*invC-invlam*K(n,:)'*X(n,:)*invC;
   e(n)=d(n)-H0est(n)-Hest(n,:)*X(n,:)';
end
```

```
   emean=emean+e;
   Hmean=Hmean+Hest(1:Niter,:);
   H0mean=H0mean+H0est(1:Niter);
end

e=emean./Nexpe;
Hest=Hmean./Nexpe;
H0est=H0mean./Nexpe;



%gráficas
n=1:Niter;
plot(n,e(n)),xlabel('Nºiteraciones'),ylabel('error con
  varianza 1')
pause
plot(n,H0est(n),'b',n,H0,'r'),xlabel('Nº    iteraciones'),
  ylabel('coeficiente h0')
pause
plot(n,Hest(n,1),'b',n,H(1),'r'),xlabel('Nº  iteraciones'),
  ylabel('coeficiente h1')
pause
plot(n,Hest(n,4),'b',n,H(4),'r'),xlabel('Nº  iteraciones'),
  ylabel('coeficiente h2')
pause

%número de operaciones
oper=flops


%convergencia
z=1;
t=1;
for g=1:Niter
  if(abs(e(g))>0.5)
     result(z)=g;
     z=z+1;
  end
  if(abs(e(g))>0.55)
     res(t)=g;
     t=t+1;
  end
end
convergencia50=max(result)
convergencia55=max(res)
```

## 6.2.- Ejemplo 2

- **LMS:**

%%%%%Identificar este sistema utilizando LMS-Volterra:

$$y(n) = -0.78x(n) - 1.48x(n-1) + 1.39x(n-2) + 0.04x(n-3) +$$

$$+ 0.54x^2(n) + 3.72x(n)x(n-1) + 1.86x(n)x(n-2) - 0.76x(n)x(n-3) -$$

$$- 1.62x^2(n-1) + 1.41x^2(n-2) - 1.52x(n-2)x(n-3) - 0.13x^2(n-3)$$

```
%%%Se simulan 50 experimentos independientes
%%%Se han representado las siguientes gráficas
%%%        vector de error
%%%        convergencia de los coeficientes


flops(0) %para saber el número de operaciones
Nexpe=50; %número de experimentos independientes para luego
          %obtener un valor medio
emean=0;
Niter=5000; %número de iteraciones en el algoritmo
N=4;
Hmean=zeros(Niter,N*(N+3)/2);
for expe=1:Nexpe

   %obtención de x[n]
   Rw=0.05;
   randn('state',sum(100*clock)); %para que cada vez tome
                                  %un valor distinto
   w=sqrt(Rw)*randn(Niter,1); %w[n] ruido gausiano con
                              %varianza 0.05 y media 0
   h=[0.9045,1,0.9045];
   x=filter(h,1,w);

   %vector de entrada
   %cada fila es para un instante de tiempo
   N=4; %N-1=3 elementos de retraso
   %primeras N-1 filas
   X(1,1:14)=[x(1),zeros(1,3),x(1)^2,zeros(1,9)];
   X(2,1:14)=[x(2),x(1),zeros(1,2),x(2)^2,x(2)*x(1),
     zeros(1,2),x(1)^2,zeros(1,5)];
   X(3,1:14)=[x(3),x(2),x(1),0,x(3)^2,x(3)*x(2),x(3)*x(1),
      0,x(2)^2,x(2)*x(1),0,x(1)^2,0,0];
```

```
   %resto de filas
   for k=4:Niter
      X(k,1:N*(N+3)/2)=[x(k),x(k-1),x(k-2),x(k-
        3),x(k)^2,x(k)*x(k-1),x(k)*x(k-2),x(k)*x(k-3),x(k-
        1)^2,x(k-1)*x(k-2),x(k-1)*x(k-3),x(k-2)^2,x(k-
        2)*x(k-3),x(k-3)^2];
   end


   %vector de coeficientes
   H=[-0.78,-1.48,1.39,0.04,0.54,3.72,1.86,-0.76,-
      1.62,0,0,1.41,-1.52,-0.13];


   %obtención de la salida del sistema
   y(1)=H(1)*x(1)+H(5)*x(1)^2;
   y(2)=H(1)*x(2)+H(2)*x(1)+H(5)*x(2)^2+H(6)*x(2)*x(1)+
       H(9)*x(1)^2;
   y(3)=H(1)*x(3)+H(2)*x(2)+H(3)*x(1)+H(5)*x(3)^2+
        H(6)*x(3)*x(2)+H(7)*x(3)*x(1)+H(9)*x(2)^2+
        H(10)*x(2)*x(1)+H(12)*x(1)^2;
   for k=4:Niter
      y(k)=H(1)*x(k)+H(2)*x(k-1)+H(3)*x(k-2)+H(4)*x(k-
          3)+H(5)*x(k)^2+H(6)*x(k)*x(k-1)+H(7)*x(k)*x(k-
          2)+H(8)*x(k)*x(k-3)+H(9)*x(k-1)^2+H(10)*x(k-
          1)*x(k-2)+H(11)*x(k-1)*x(k-3)+H(12)*x(k-
          2)^2+H(13)*x(k-2)*x(k-3)+H(14)*x(k-3)^2;
   end

   %obtención de la señal deseada
   d=y; %en este caso no se le suma ningún ruido a la
        %salida del sistema

   %algoritmo LMS
   u=0.4; %paso de adaptación
   Hest=zeros(Niter,N*(N+3)/2);
   for j=1:N*(N+3)/2
      udiag(j)=u;
   end
   umatr=diag(udiag); %matriz diagonal con u en la diagonal
   for n=1:Niter
      e(n)=d(n)-Hest(n,:)*X(n,:)';
      Hest(n+1,:)=Hest(n,:)+(umatr*X(n,:)'*e(n))';
   end
   emean=emean+e;
   Hmean=Hmean+Hest(1:Niter,:);
end

e=emean./Nexpe;
Hest=Hmean./Nexpe;
```

```
%gráficas
n=1:Niter;
plot(n,e(n)),xlabel('Nºiteraciones'),ylabel('error')
pause
plot(n,Hest(n,1),'b',n,H(1),'r',Hest(n,2),'g',H(2),'r',Hest(n
  ,3),'y',H(3),'r'),xlabel('Nº              iteraciones'),
  ylabel('coeficientes h1(0),h1(1),h1(2)')
pause
plot(n,Hest(n,4),'b',n,H(4),'r',Hest(n,5),'g',H(5),'r',Hest(n
  ,6),'y',H(6),'r'),xlabel('Nº              iteraciones'),
  ylabel('coeficientes h1(3),h2(0,0),h2(0,1)')
pause
plot(n,Hest(n,7),'b',n,H(7),'r',Hest(n,8),'g',H(8),'r',Hest(n
  ,9),'y',H(9),'r'),xlabel('Nº              iteraciones'),
  ylabel('coeficientes h2(0,2),h2(0,3),h2(1,1)')
pause
plot(n,Hest(n,12),'b',n,H(12),'r',Hest(n,13),'g',H(13),'r',He
  st(n,14),'y',H(14),'r'),xlabel('Nº         iteraciones'),
  ylabel('coeficientes h2(2,2),h2(2,3),h2(3,3)')
pause


%número de operaciones
oper=flops


%convergencia
z=1;
t=1;
s=1;
for g=1:Niter
  if(abs(e(g))>0.03)
     resultado(z)=g;
     z=z+1;
  end
  if(abs(e(g))>0.1)
     resul(t)=g;
     t=t+1;
  end
  if(abs(e(g))>0.15)
     res(s)=g;
     s=s+1;
  end
end
convergencia3=max(resultado)
convergencia10=max(resul)
convergencia15=max(res)
```

- **LMS normalizado:**

```
%%%%%Identificar este sistema utilizando LMS normalizado
%%%%%Volterra:
```

$$y(n) = -0.78x(n) - 1.48x(n-1) + 1.39x(n-2) + 0.04x(n-3) +$$

$$+ 0.54x^2(n) + 3.72x(n)x(n-1) + 1.86x(n)x(n-2) - 0.76x(n)x(n-3) -$$

$$- 1.62x^2(n-1) + 1.41x^2(n-2) - 1.52x(n-2)x(n-3) - 0.13x^2(n-3)$$

```
%%%Se simulan 50 experimentos independientes
%%%Se representan las siguientes gráficas
%%%        vector de error
%%%        convergencia de los coeficientes


flops(0)
Nexpe=50; %número de experimentos independientes para luego
          %obtener un valor medio
emean=0;
Niter=6000; %número de iteraciones en el algoritmo
N=4;
Hmean=zeros(Niter,N*(N+3)/2);
for expe=1:Nexpe


   %obtención de x[n]
   Rw=0.05;
   randn('state',sum(100*clock)); %para que cada vez tome
                                  %un valor distinto
   w=sqrt(Rw)*randn(Niter,1); %w[n] ruido gausiano con
                              %varianza 0.05 y media 0
   h=[0.9045,1,0.9045];
   x=filter(h,1,w);

   %vector de entrada
   %cada fila es para un instante de tiempo
   N=4; %N-1=3 elementos de retraso
   %primeras N-1 filas
   X(1,1:14)=[x(1),zeros(1,3),x(1)^2,zeros(1,9)];
   X(2,1:14)=[x(2),x(1),zeros(1,2),x(2)^2,x(2)*x(1),
       zeros(1,2),x(1)^2,zeros(1,5)];
   X(3,1:14)=[x(3),x(2),x(1),0,x(3)^2,x(3)*x(2),
       x(3)*x(1),0,x(2)^2,x(2)*x(1),0,x(1)^2,0,0];

   %resto de filas
   for k=4:Niter
```

```
      X(k,1:N*(N+3)/2)=[x(k),x(k-1),x(k-2),x(k-
        3),x(k)^2,x(k)*x(k-1),x(k)*x(k-2),x(k)*x(k-3),x(k-
        1)^2,x(k-1)*x(k-2),x(k-1)*x(k-3),x(k-2)^2,x(k-
        2)*x(k-3),x(k-3)^2];
   end


   %vector de coeficientes
   H=[-0.78,-1.48,1.39,0.04,0.54,3.72,1.86,-0.76,-
      1.62,0,0,1.41,-1.52,-0.13];


   %obtención de la salida del sistema
   y(1)=H(1)*x(1)+H(5)*x(1)^2;
   y(2)=H(1)*x(2)+H(2)*x(1)+H(5)*x(2)^2+H(6)*x(2)*x(1)+
       H(9)*x(1)^2;
   y(3)=H(1)*x(3)+H(2)*x(2)+H(3)*x(1)+H(5)*x(3)^2+
        H(6)*x(3)*x(2)+H(7)*x(3)*x(1)+H(9)*x(2)^2+
        H(10)*x(2)*x(1)+H(12)*x(1)^2;
   for k=4:Niter
      y(k)=H(1)*x(k)+H(2)*x(k-1)+H(3)*x(k-2)+H(4)*x(k-
          3)+H(5)*x(k)^2+H(6)*x(k)*x(k-1)+H(7)*x(k)*x(k-
          2)+H(8)*x(k)*x(k-3)+H(9)*x(k-1)^2+H(10)*x(k-
          1)*x(k-2)+H(11)*x(k-1)*x(k-3)+H(12)*x(k-
          2)^2+H(13)*x(k-2)*x(k-3)+H(14)*x(k-3)^2;
   end

   %obtención de la señal deseada
   d=y; %en este caso no se le suma ningún ruido a la
        %salida del sistema

   %algoritmo LMS
   u=1.5; %paso de adaptación
   Hest=zeros(Niter,N*(N+3)/2);
   for j=1:N*(N+3)/2
      udiag(j)=u;
   end
   umatr=diag(udiag); %matriz diagonal con u en la diagonal
   for n=1:Niter
      e(n)=d(n)-Hest(n,:)*X(n,:)';
      Hest(n+1,:)=Hest(n,:)+(umatr*X(n,:)'*e(n))'/
                  (norm(X(n,:)));
   end
   emean=emean+e;
   Hmean=Hmean+Hest(1:Niter,:);
end


e=emean./Nexpe;
Hest=Hmean./Nexpe;
```

```
%gráficas
n=1:Niter;
plot(n,e(n)),xlabel('Nºiteraciones'),ylabel('error')
pause
plot(n,Hest(n,1),'b',n,H(1),'r',Hest(n,2),'g',H(2),'r',Hest(n
  ,3),'y',H(3),'r'),xlabel('Nº              iteraciones'),
  ylabel('coeficientes h1(0),h1(1),h1(2)')
pause
plot(n,Hest(n,4),'b',n,H(4),'r',Hest(n,5),'g',H(5),'r',Hest(n
  ,6),'y',H(6),'r'),xlabel('Nº              iteraciones'),
  ylabel('coeficientes h1(3),h2(0,0),h2(0,1)')
pause
plot(n,Hest(n,7),'b',n,H(7),'r',Hest(n,8),'g',H(8),'r',Hest(n
  ,9),'y',H(9),'r'),xlabel('Nº              iteraciones'),
  ylabel('coeficientes h2(0,2),h2(0,3),h2(1,1)')
pause
plot(n,Hest(n,12),'b',n,H(12),'r',Hest(n,13),'g',H(13),'r',He
  st(n,14),'y',H(14),'r'),xlabel('Nº        iteraciones'),
  ylabel('coeficientes h2(2,2),h2(2,3),h2(3,3)')
pause


%número de operaciones
oper=flops


%convergencia
z=1;
t=1;
s=1;
for g=1:Niter
  if(abs(e(g))>0.03)
     resultado(z)=g;
     z=z+1;
  end
  if(abs(e(g))>0.1)
     resul(t)=g;
     t=t+1;
  end
  if(abs(e(g))>0.15)
     res(s)=g;
     s=s+1;
  end
end
convergencia3=max(resultado)
convergencia10=max(resul)
convergencia15=max(res)
```

- **RLS:**

%%%%%Identificar este sistema utilizando RLS-Volterra:

$$y(n) = -0.78x(n) - 1.48x(n-1) + 1.39x(n-2) + 0.04x(n-3) +$$

$$+ 0.54x^2(n) + 3.72x(n)x(n-1) + 1.86x(n)x(n-2) - 0.76x(n)x(n-3) -$$

$$- 1.62x^2(n-1) + 1.41x^2(n-2) - 1.52x(n-2)x(n-3) - 0.13x^2(n-3)$$

```
%%%Se simulan 50 experimentos independientes
%%%Se han representado las siguientes gráficas:
%%%        vector de error
%%%        convergencia de los coeficientes


flops(0)
Nexpe=50; %número de experimentos independientes para luego
          %obtener un valor medio
emean=0;
Niter=6000; %número de iteraciones en el algoritmo
N=4;
Hmean=zeros(Niter,N*(N+3)/2);

for expe=1:Nexpe


   %obtención de x[n]
   Rw=0.05;
   randn('state',sum(100*clock)); %para que cada vez tome
                                  %un valor distinto
   w=sqrt(Rw)*randn(Niter,1); %w[n] ruido gausiano con
                              %varianza 0.05 y media 0
   h=[0.9045,1,0.9045];
   x=filter(h,1,w);

   %vector de entrada
   %cada fila es para un instante de tiempo
   N=4; %N-1=3 elementos de retraso
   %primeras N-1 filas
   X(1,1:N*(N+3)/2)=[x(1),zeros(1,3),x(1)^2,zeros(1,9)];
   X(2,1:N*(N+3)/2)=[x(2),x(1),zeros(1,2),x(2)^2,x(2)*x(1),
       zeros(1,2),x(1)^2,zeros(1,5)];
   X(3,1:N*(N+3)/2)=[x(3),x(2),x(1),0,x(3)^2,x(3)*x(2),
       x(3)*x(1),0,x(2)^2,x(2)*x(1),0,x(1)^2,0,0];
```

```
%resto de filas
for k=4:Niter
   X(k,1:N*(N+3)/2)=[x(k),x(k-1),x(k-2),x(k-
       3),x(k)^2,x(k)*x(k-1),x(k)*x(k-2),x(k)*x(k-
       3),x(k-1)^2,x(k-1)*x(k-2),x(k-1)*x(k-3),x(k-
       2)^2,x(k-2)*x(k-3),x(k-3)^2];
end


%vector de coeficientes
H=[-0.78,-1.48,1.39,0.04,0.54,3.72,1.86,-0.76,-
   1.62,0,0,1.41,-1.52,-0.13];


%obtención de la salida del sistema
y(1)=H(1)*x(1)+H(5)*x(1)^2;
y(2)=H(1)*x(2)+H(2)*x(1)+H(5)*x(2)^2+H(6)*x(2)*x(1)+
     H(9)*x(1)^2;
y(3)=H(1)*x(3)+H(2)*x(2)+H(3)*x(1)+H(5)*x(3)^2+
     H(6)*x(3)*x(2)+H(7)*x(3)*x(1)+H(9)*x(2)^2+
     H(10)*x(2)*x(1)+H(12)*x(1)^2;
for k=4:Niter
   y(k)=H(1)*x(k)+H(2)*x(k-1)+H(3)*x(k-2)+H(4)*x(k-
        3)+H(5)*x(k)^2+H(6)*x(k)*x(k-1)+H(7)*x(k)*x(k-
        2)+H(8)*x(k)*x(k-3)+H(9)*x(k-1)^2+H(10)*x(k-
        1)*x(k-2)+H(11)*x(k-1)*x(k-3)+H(12)*x(k-
        2)^2+H(13)*x(k-2)*x(k-3)+H(14)*x(k-3)^2;
end

%obtención de la señal deseada
d=y; %en este caso no se le suma ningún ruido a la
     %salida del sistema

%algoritmo RLS
u=0.7; %paso de adaptación
for j=1:N*(N+3)/2
   udiag(j)=u;
end
umatr=diag(udiag); %matriz diagonal con u en la diagonal
lambda=0.995;
invlam=1/lambda;
Hest=zeros(Niter,N*(N+3)/2);
delta=0.001;
invC=(1/delta)*eye(N*(N+3)/2);

%primera iteración
K(1,1:N*(N+3)/2)=((invlam*invC*X(1,:)')/
        (1+invlam*X(1,:)*invC*X(1,:)'))';
eps(1)=d(1)-zeros(1,N*(N+3)/2)*X(1,:)';
Hest(1,:)=zeros(1,N*(N+3)/2)+(umatr*K(1,:)'*eps(1))';
invC=invlam*invC-invlam*K(1,:)'*X(1,:)*invC;
```

```
   e(1)=d(1)-Hest(1,:)*X(1,:)';

   %resto de iteraciones
   for n=2:Niter
      K(n,1:N*(N+3)/2)=((invlam*invC*X(n,:)')/
              (1+invlam*X(n,:)*invC*X(n,:)'))';
      eps(n)=d(n)-Hest(n-1,:)*X(n,:)';
      Hest(n,:)=Hest(n-1,:)+(umatr*K(n,:)'*eps(n))';
      invC=invlam*invC-invlam*K(n,:)'*X(n,:)*invC;
      e(n)=d(n)-Hest(n,:)*X(n,:)';
   end
   emean=emean+e;
   Hmean=Hmean+Hest(1:Niter,:);
end

e=emean./Nexpe;
Hest=Hmean./Nexpe;


%gráficas
n=1:Niter;
plot(n,e(n)),xlabel('Nºiteraciones'),ylabel('error')
pause
plot(n,Hest(n,1),'b',n,H(1),'r',Hest(n,2),'g',H(2),'r',Hest(n
  ,3),'y',H(3),'r'),xlabel('Nº              iteraciones'),
  ylabel('coeficientes h1(0),h1(1),h1(2)')
pause
plot(n,Hest(n,4),'b',n,H(4),'r',Hest(n,5),'g',H(5),'r',Hest(n
  ,6),'y',H(6),'r'),xlabel('Nº              iteraciones'),
  ylabel('coeficientes h1(3),h2(0,0),h2(0,1)')
pause
plot(n,Hest(n,7),'b',n,H(7),'r',Hest(n,8),'g',H(8),'r',Hest(n
  ,9),'y',H(9),'r'),xlabel('Nº              iteraciones'),
  ylabel('coeficientes h2(0,2),h2(0,3),h2(1,1)')
pause
plot(n,Hest(n,12),'b',n,H(12),'r',Hest(n,13),'g',H(13),'r',He
  st(n,14),'y',H(14),'r'),xlabel('Nº        iteraciones'),
  ylabel('coeficientes h2(2,2),h2(2,3),h2(3,3)')
pause


%número de operaciones
oper=flops


%convergencia
z=1;
t=1;
s=1;
for g=1:Niter
  if(abs(e(g))>0.03)
```

```
      resultado(z)=g;
      z=z+1;
   end
   if(abs(e(g))>0.1)
      resul(t)=g;
      t=t+1;
   end
   if(abs(e(g))>0.15)
      res(s)=g;
      s=s+1;
   end
end
convergencia3=max(resultado)
convergencia10=max(resul)
convergencia15=max(res)
```

# CAPÍTULO 8.

# BIBLIOGRAFÍA


- [1] V.J. Mathews, “Adaptive polynomial filters”, IEEE Signal Processing Magazine, pp 10-26, Julio 1991.

- [2] Giovanni L. Sicuranza, “Quadratic filters for signal processing”, Proceedings of the IEEE, Vol. 80, NO. 8, pp 1263-1285, Agosto 1992.

- [3] Simon Haykin, “Adaptive filter theory”, Prentice-Hall International, Englewood Cliff, N.J., 1996, 3ª edición.

- [4] Maurice G. Bellenger, “Adaptive digital filters and signal analysis”, Marcel Dekker, New York, 1987.