View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by UPCommons. Portal del coneixement obert de la UPC

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# TRABAJO FINAL DE GRADO

**TÍTULO DEL TFG: Programación eficiente de algoritmos de estimación de parámetros para modelado de amplificadores de RF**

**TITULACIÓN: Grado en Ingeniería de Sistemas de Telecomunicación**

**AUTOR: Rahul Bhambi Blanco**

**DIRECTOR: Gabriel Montoro López**

**CODIRECTOR: Pere Lluís Gilabert Pinal**

**FECHA: 25/07/2017**

**Título:** Programación eficiente de algoritmos de estimación de parámetros para modelado de amplificadores de RF

**Autor:** Rahul Bhambi Blanco

**Director:** Gabriel Montoro López

**Codirector:** Pere Lluís Gilabert Pinal

**Fecha:** 25 de julio del 2017

**Resumen**

**Los filtros adaptativos es un dispositivo que intenta modelar la relación entre señales en tiempo real de forma iterativa.**

**Entre todos los filtros adaptativos, el que más destaca entre ellos es el Least Mean Square (LMS). El Least Mean Square fue creado en el año 1960 por el profesor Bernard Widrow y por su estudiante Ted Hoff.**

**El Least Mean Square lo utilizamos para calcular los coeficientes a partir de la señal deseada con la señal de salida calculada a partir de los coeficientes obtenidos. El LMS tiene varias formas para obtener sus coeficientes: Standard LMS, NLMS, Sign-LMS, etc. El NLMS, normalizamos la señal de entrada, a la hora de calcular los coeficientes y el Sign-LMS a partir de la señal de entrada, error o ambas a la vez, donde si el número es positivo le daremos valor igual a 1 y si el numero negativo le daremos un valor equivalente a -1.**

**Con cada forma de LMS, podemos ver quién es el más rápido a la hora de obtener de coeficientes o podemos ver también cuál es el que tiene menos error.**

**Títle:** Efficient programming of parameter estimation algorithms for modeling and digital predistorsion of RF amplifiers

**Author:** Rahul Bhambi Blanco

**Advisor:** Gabriel Montoro López

**Coadvisor: Pere Lluís Gilabert Pinal**

**Date:** July 25, 2017

**Overview**

**An adaptive filter is a device that attempts to model the relationship between signals in real time in an iterative way.**

**Among all the adaptive filters, the most outstanding among them is the Least Mean Square (LMS). Least Mean Square was created in 1960 by Professor Bernand Widrow and his student Ted Hoff.**

**Least Mean Square is used to calculate the coefficients from the desired output signal with the output signal that we obtain from the coefficients that we have calculated. The LMS has several ways to get its coefficients: LMS, NLMS, Sign-LMS, etc. NLMS, normalize the input signal when we calculate coefficients and the Sign-LMS from the input signal, when calculating the coefficients and Sign-LMS from the input signal, error or both at once, where if the number is positive we will give value equal to 1 and if we obtain a negative number, the value is -1.**

**With each form of LMS, we can see who is the fastest when it comes to obtaining coefficients or we can also which one has the least error.**

# ÍNDICE

**INTRODUCCIÓN** ........ 7
**CAPITULO 1. FILTROS ADAPTATIVOS** ........ 8
1.1. Definición ........ 8
1.2. Características ........ 9
1.3. Aplicaciones ........ 9
1.3.1. Identificación del sistema ........ 9
1.3.2. Modelo inverso del sistema ........ 10
1.3.3. Predicción de la señal ........ 11
1.3.4. Cancelación de interferencias ........ 11
**CAPITULO 2: LMS** ........ 12
2.1. Introducción ........ 12
2.2. Tipos de Least Mean Square ........ 13
2.2.1 Standard LMS ........ 13
2.2.2. NLMS ........ 15
2.2.3. Sign LMS ........ 17
**CAPITULO 3: PLATAFORMAS DEL PROYECTO** ........ 20
3.1. Plataformas ........ 20
3.2. Matlab ........ 21
3.3. Función Mex ........ 21
3.3.1. Introducción ........ 21
3.3.2. Razones ........ 22
3.3.3. Implementación ........ 23
3.3.4. Función ........ 24
3.4. C ........ 25
**CAPÍTULO 4: PARAMETROS** ........ 26
4.1. NMSE ........ 26
4.2. Least Square ........ 26
**CAPÍTULO 5: RESULTADOS** ........ 28
**CONCLUSIONES** ........ 38
**BIBLIOGRAFIA** ........ 39
**ANEXOS** ........ 40
A.1. Código en MATLAB de datos ........ 40

A.2. Código en Matlab para ejecutar lostipos LMS ........ 42
A.2.1. Standard LMS ........ 42
A.2.2. Normalised LMS ........ 42
A.2.3. Sign LMS ........ 42
A.3. Código en C para ejecutar los tipos LMS ........ 43
A.3.1. Standard LMS ........ 43
A.3.2. Normalised LMS ........ 44
A.3.3. Sign LMS ........ 45

# ÍNDICE DE FIGURAS

Fig 1. 1. Identificación del sistema ..... 10
Fig 1. 2. Modelo inverso del sistema ..... 10
Fig 1. 3. Predicción de la señal ..... 11
Fig 1. 4. Cancelación de interferencias ..... 11

Fig 2. 1. Filtro LMS ..... 12
Fig 2. 2. Función sign(x) ..... 17

Fig 3. 1. Plataformas del proyecto ..... 20
Fig 3. 2. Función mexFunction ..... 22
Fig 3. 3. Proceso que sigue MEX ..... 24
Fig 3. 4. La función Mex que utilizaremos para cada tipo de LMS ..... 25

Fig 4. 1. Predicción entre la salida deseada y la calculada con los coeficientes obtenidos ..... 27

Fig 5. 1. Gráfica del LMS con datos absolutos y complejos ..... 28
Fig 5. 2. Gráfica del NLMS con datos absolutos y complejos ..... 29
Fig 5. 3. Gráfica del Sign LMS con datos absolutos y complejos ..... 29
Fig 5. 4. Gráfica del LMS con datos en Matlab y C ..... 30
Fig 5. 5. Gráfica del LMS con datos en Matlab y C ..... 31
Fig 5. 6. Gráfica del LMS con datos en Matlab y C ..... 31
Fig 5. 7. Velocidad de LMS obtenidas a partir del speed ..... 33
Fig 5. 8. Predicción y error del LMS en absoluto y complejo ..... 34
Fig 5. 9. Predicción y error del NLMS en absoluto y complejo ..... 34
Fig 5. 10. Predicción y error del Sign LMS en absoluto y complejo ..... 35
Fig 5. 11. Tabla de la NMSE, con valor de $\mu$=0.015, $\mu$=0,05 y $\mu$=0,1 ..... 36
Fig 5. 12. Gráfica del NMSE, con valor de $\mu$=0.015, $\mu$=0.05 y $\mu$=0.1 ..... 37

# INTRODUCCIÓN

El Least Mean Square es un tipo de filtro adaptativo que se usa para calcular los coeficientes y minimizar la señal de salida deseada y la señal de salida que obtenemos a partir de calcular los coeficientes.

Lo primero de todo, es estudiar que es un filtro adaptativo y porque escogemos el LMS. También miraremos las características que hay en un filtro adaptativo y las aplicaciones que tiene.

Tras conocer las nociones del filtro adaptativo, profundizaremos en que es LMS y las razones por lo cual lo vamos utilizar. A parte de conocer el LMS conceptualmente, iremos a conocer que tipos de LMS vamos a conocer y las características de cada uno para saber cual es mejor en cada caso y las variaciones que sufrirán a la hora de calcularlo.

Hablaremos también de las plataformas que hemos usado para realizar los cálculos de los coeficientes y la razón por la cual vamos a utilizarlos.

Una vez conocida la plataforma que vamos a utilizar, hablaremos sobre los parámetros en el que nos hemos basado para hacer las comparaciones.

# CAPITULO 1. FILTROS ADAPTATIVOS

El Least Mean Square es un tipo de filtro adaptativo y es por ello, que en este capítulo, vamos a detallar que es un filtro adaptativo, sus características y los diferentes usos que podemos hacer de él.

## 1.1. Definición

Un filtro adaptativo es un dispositivo computacional que intenta modelar la relación entre dos señales en tiempo real de manera iterativa. Los filtros adaptativos se realizan a menudo como un conjunto de instrucciones de programa que se ejecutan en un dispositivo de procesado aritmético tal como un microprocesador o un chip DSP, o como un conjunto de operaciones lógicas implementadas en una FPGA o en un VLSI [1].

Un filtro adaptativo está definido por 4 aspectos:

- Las señales son procesadas por el filtro.
- La estructura que define cómo se calcula la señal de salida del filtro es a partir de su señal de entrada.
- Los parámetros dentro de esta estructura que se pueden cambiar iterativamente para alterar la relación entrada-salida del filtro.
- El algoritmo adaptativo que describe cómo se ajustan los parámetros de un instante al siguiente.

El filtro adaptativo se aproxima a los algoritmos iterativos debidos a que:

- No requieren conocimientos previos de las estadísticas de la señal.
- Tiene una pequeña complejidad computacional por iteraciones.
- Converge a una solución óptima cercana.

Los filtros adaptativos son utilizados normalmente en:

- Aplicaciones en tiempo real, cuando no hay tiempo para hacer las estimaciones de estadística.
- Aplicaciones con señales y/o sistemas no estacionarios.

## 1.2. Características

Las características del filtro adaptativo son las siguientes:

- Buscar la solución óptima en la superficie de rendimiento.
- Seguir los principios de las técnicas de optimización.
- Implementar una solución de optimización recursiva.
- La velocidad de convergencia puede depender de la inicialización.
- Tener regiones de estabilidad.
- La solución en estado estacionario fluctúa alrededor de la óptima.
- Puede controlar el tiempo que varía los entornos de operación de la señal mejor que los filtros óptimos.
- El rendimiento depende de la superficie de rendimiento.

## 1.3. Aplicaciones

Los filtros adaptativos lo podemos implementar en 4 aplicaciones posibles [2]:

1. Identificación del sistema.
2. Modelo inverso del sistema.
3. Predicción de la señal.
4. Cancelación de interferencias.

### 1.3.1. Identificación del sistema

Un filtro adaptativo lo utilizamos para proporcionar un modelo lineal que representa el mejor ajuste a una planta desconocida. El filtro adaptativo y la planta utiliza la misma señal de entrada. En la salida de la planta obtendremos la señal deseada para el filtro adaptativo.

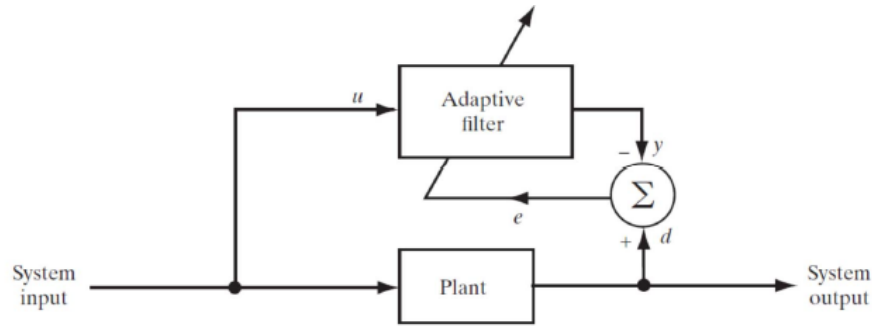


**Fig 1. 1.** Identificación del sistema

### 1.3.2. Modelo inverso del sistema

El filtro adaptativo proporciona un modelo inverso que representa el mejor ajuste a una planta ruidosa desconocida. Idealmente, en el caso de un sistema lineal, el modelo inverso tiene una función de transferencia igual a la reciproca (inversa) de la función de transferencia de la planta, de tal manera que la combinación de los dos constituye un medio de transmisión ideal.

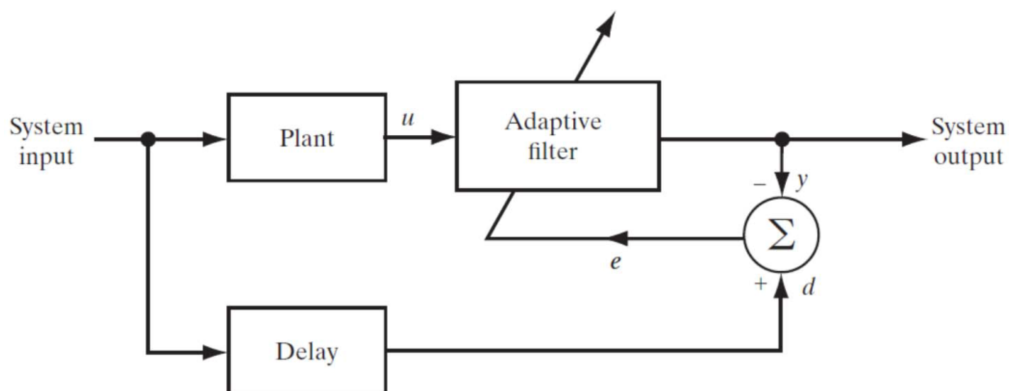


**Fig 1. 2.** Modelo inverso del sistema

### 1.3.3. Predicción de la señal

El filtro adaptativo proporciona la mejor predicción del valor actual de una señal aleatoria. El valor presente de la señal sirve, por lo tanto, al propósito de una respuesta deseada para el filtro adaptativo. Los valores pasados de la señal suministran la entrada aplicada al filtro.

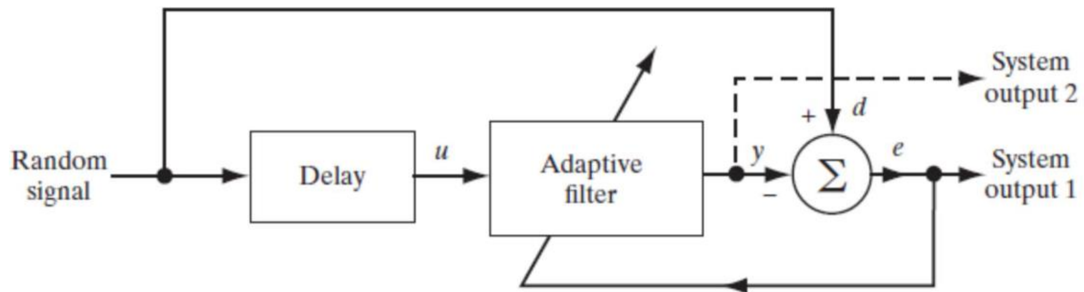


**Fig 1. 3.** Predicción de la señal

### 1.3.4. Cancelación de interferencias

El filtro adaptativo se usa para cancelar la interferencia desconocida contenida en una señal primaria, con la optimización de la cancelación en algún sentido. La señal primaria sirve como la respuesta deseada para el filtro adaptativo. Se emplea una señal de referencia (auxiliar) como entrada al filtro.

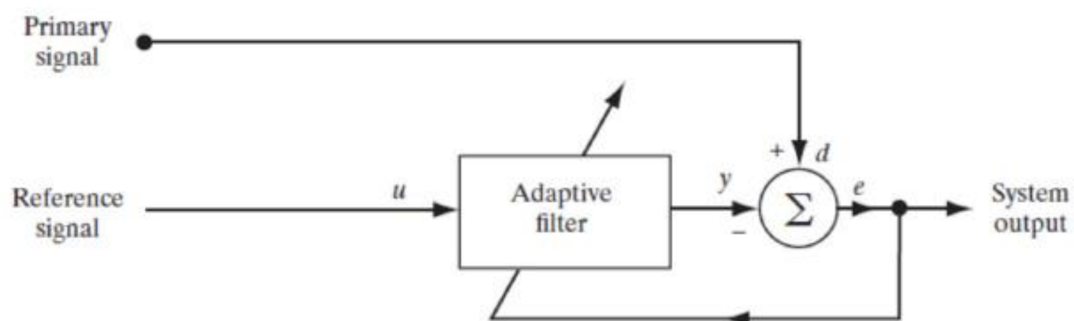


**Fig 1. 4.** Cancelación de interferencias

# CAPITULO 2: LMS

En este capítulo, vamos a explicar el funcionamiento del Least Mean Square y los problemas que tenemos que resolver. También, explicaremos los diferentes tipos de Least Mean Square.

## 2.1. Introducción

El Least Mean Square (LMS) es un algoritmo de búsqueda en el que una simplificación de la computación de vectores de gradiente es posible mediante la modificación apropiada de la función objetivo.

El algoritmo LMS es ampliamente utilizado en diversas aplicaciones de filtrado adaptativo debido a su simplicidad computacional. Por ejemplo, los usamos para encontrar los coeficientes para obtener el valor de la señal del error, que está definida como la diferencia entre la señal que nosotros deseamos d(n) y la señal que tenemos a la salida del filtro y(n).

El algoritmo LMS es el algoritmo más utilizado en el filtrado adaptativo por varias razones. Las principales características de su uso son por su baja complejidad computacional, la prueba de convergencia en el entorno estacionario y el comportamiento estable con implementación finita [3].

El esquema de la figura 2.1. [4], es la manera en la que podemos representar el filtro LMS.

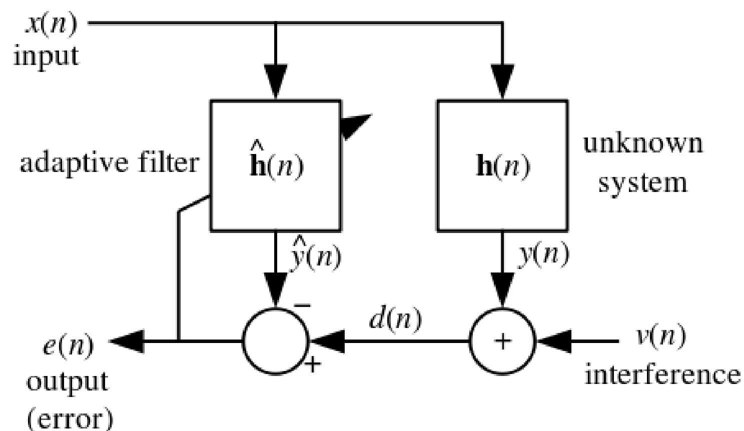


**Fig 2. 1.** Filtro LMS

El algoritmo LMS es un algoritmo de filtrado lineal adaptativo que, en general, consiste de dos procesos básicos:

- Un proceso de filtrado, que involucra: el cómputo de la salida de un filtro lineal en respuesta a una señal de entrada y la generación de una estimación del error mediante la comparación de esta salida con la señal deseada.
- Un proceso adaptativo, que involucra el ajuste automático de los parámetros del filtro de acuerdo al error estimado.

## 2.2. Tipos de Least Mean Square

Hay diferentes tipos de Least Mean Square. En este trabajo, nos hemos centrado en 3 de ellos:

- Standard Least Mean Square
- Normalised Least Mean Square
- Sign-Least Mean Square

### 2.2.1 Standard LMS

El Standard Least Mean Square es el primero del que vamos a hablar. El número de iteraciones que tiene es igual a la dimensionalidad de la entrada.

Las ventajas son las siguientes:

- La implementación que tiene es muy simple.
- Rendimiento estable y robusto frente a las diferentes condiciones de la señal.

La única desventaja que podemos encontrarnos es su lenta convergencia.

A continuación, vemos como podemos calcular el Standard Least Mean Square:

Parámetros de entrada:

- Inicialización del vector: $w(n)=0$
- Vector de entrada: $x(n)$
- Vector de salida deseado: $d(n)$
- Vector error: $e(n)$
- Orden del filtro: $M$
- Parámetro mu: $\mu$

Parámetros de salida:

- Filtro de salida: y(n)
- Coeficientes del vector: w(n+1)

Procedimiento:

- Valores reales:
  1. Calcular la señal de salida y(n) desde el filtro adaptativo.

$$y(n) = x(n) * w^T(n) \quad \textbf{(2.1)}$$

  2. Calcular la señal error e(n) usando la siguiente fórmula.

$$e(n) = d(n) - y(n) \quad \textbf{(2.2)}$$

  3. Calcular los coeficientes del vector utilizando la siguiente fórmula.

$$w(n+1) = w(n) + \mu * e(n) * x^T(n) \quad \textbf{(2.3)}$$

*Nota: El superíndice T denota Transposición.*

- Valores imaginarios:
  1. Calcular la señal de salida y(n) desde el filtro adaptativo.

$$y(n) = x(n) * w^H(n) \quad \textbf{(2.4)}$$

  2. Calcular la señal error e(n) usando la siguiente fórmula.

$$e(n) = d(n) - y(n) \quad \textbf{(2.5)}$$

  3. Calcular los coeficientes del vector utilizando la siguiente fórmula.

$$w(n+1) = w(n) + \mu * e(n) * x^H(n) \quad \textbf{(2.6)}$$

*Nota: El superíndice H denota una matriz Hermítica, equivalente a una conjugada transpuesta.*

El valor del parámetro μ se calcula de la siguiente manera:

$$0 < \mu < 2 \quad \textbf{(2.7)}$$

### 2.2.2. NLMS

En términos estructurales, tanto el filtro NLMS como el filtro Standard LMS, son iguales. En realidad, el NLMS es una variante del LMS donde introduce una normalización en el vector de x(n).

Las ventajas del NLMS son las siguientes:

- Como aquí la ecuación está normalizada, este algoritmo converge más rápido que el LMS.
- Aquí el valor de error estimado entre la señal deseada y la salida del filtro es menor que LMS.

Las desventajas del NLMS que podemos que encontrarnos es que es más complejo y menos estable que el LMS.

Como hicimos en el Standard LMS, vamos a explicar cómo calculamos el NLMS:

Parámetros de entrada:

- Inicialización del vector: w(n)=0
- Vector de entrada: x(n)
- Vector de salida deseado: d(n)
- Vector error: e(n)
- Orden del filtro: M
- Parámetro mu: μ

Parámetros de salida:

- Filtro de salida: y(n)
- Coeficientes del vector: w(n+1)

Procedimiento:

- Valores reales:
  1. Calcular la señal de salida y(n) desde el filtro adaptativo.

$$y(n) = x(n) * w^T(n) \quad \textbf{(2.8)}$$

  2. Calcular la señal error e(n) usando la siguiente fórmula.

$$e(n) = d(n) - y(n) \quad \textbf{(2.9)}$$

  3. Calcular los coeficientes del vector utilizando la siguiente fórmula.

$$w(n+1) = w(n) + \mu * e(n) * \frac{x(n)}{|x^T(n) * x(n)|} \quad \textbf{(2.10)}$$

- Valores imaginarios:

1. Calcular la señal de salida y(n) desde el filtro adaptativo.

$$y(n) = x(n) * w^H(n) \quad \textbf{(2.11)}$$

2. Calcular la señal error e(n) usando la siguiente fórmula.

$$e(n) = d(n) - y(n) \quad \textbf{(2.12)}$$

3. Calcular los coeficientes del vector utilizando la siguiente fórmula.

$$w(n+1) = w(n) + \mu * e(n) * \frac{x(n)}{|x^H(n) * x(n)|} \quad \textbf{(2.13)}$$

El valor μ para NLMS está entre en los siguientes valores:

$$0 < \mu < 2 \qquad \textbf{(2.14)}$$

### 2.2.3. Sign LMS

Para entender primero que es el Sign LMS, primero explicaremos que hace la función Sign.

Como indica la figura 2.2., si el valor de x es mayor a 0 tendrá valor a 1, si el valor de x es igual a 0 tendrá valor a 0 y los valores inferiores a 0 tendrán valor igual a -1.

$$\text{sign}(x) = \begin{cases} 1, & x > 0; \\ 0, & x = 0; \\ -1, & x < 0. \end{cases}$$

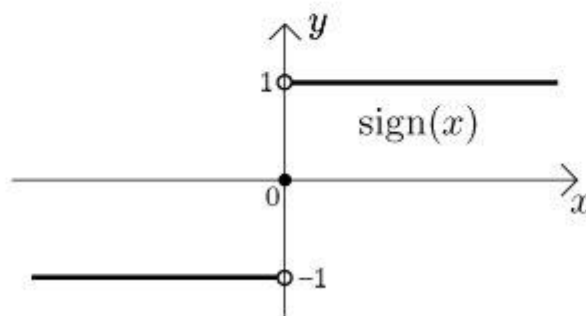


**Fig 2. 2.** Función sign(x)

Cuando x tiene valor imaginario también, separamos la parte real y la parte imaginaria. Con la parte imaginaria, si su valor es mayor a 0 tendrá valor igual a j, si es igual a 0 tendrá valor a 0 y con valor menos a 0 tendrá valor a –j.

Para expresar la fórmula matemática del Sign-LMS, hay 3 formas de expresar para calcular los coeficientes del vector:

- El Sign Algorithm(Pilot LMS o Sign error):

$$w(n+1) = w(n) + \mu * sign(e(n)) * x^T(n) \quad \textbf{(2.15)}$$

- Clipped LMS o Signed Regressor:

$$w(n+1) = w(n) + \mu * e(n) * sign(x^T(n)) \quad \textbf{(2.16)}$$

- Zero forcing LMS o Sign Sign:

$$w(n+1) = w(n) + \mu * sign(e(n)) * sign(x^T(n)) \quad \textbf{(2.17)}$$

De las 3 formas de Sign-LMS, escogemos el Sign error para calcular los coeficientes.

Parámetros de entrada:

- Inicialización del vector: w(n)=0
- Vector de entrada: x(n)
- Vector de salida deseado: d(n)
- Vector error: e(n)
- Orden del filtro: M
- Parámetro mu: μ

Parámetros de salida:

- Filtro de salida: y(n)
- Coeficientes del vector: w(n+1)

Procedimiento:

- Valores reales:
  1. Calcular la señal de salida y(n) desde el filtro adaptativo.

$$y(n) = x(n) * w^T(n) \quad \textbf{(2.18)}$$

  2. Calcular la señal error e(n) usando la siguiente fórmula.

$$e(n) = d(n) - y(n) \quad \textbf{(2.19)}$$

  3. Calcular los coeficientes del vector utilizando la siguiente fórmula.

$$w(n+1) = w(n) + \mu * sign\big(e(n)\big) * x^T(n) \quad \textbf{(2.20)}$$

- Valores imaginarios:

  1. Calcular la señal de salida y(n) desde el filtro adaptativo.

$$y(n) = x(n) * w^H(n) \quad \textbf{(2.21)}$$

  2. Calcular la señal error e(n) usando la siguiente fórmula.

$$e(n) = d(n) - y(n) \quad \textbf{(2.22)}$$

  3. Calcular los coeficientes del vector utilizando la siguiente fórmula.

$$w(n+1) = w(n) + \mu * sign\big(e(n)\big) * x^H(n) \quad \textbf{(2.23)}$$

# CAPITULO 3: PLATAFORMAS DEL PROYECTO

En el siguiente capítulo, hablaremos sobre las plataformas que vamos a usar y una vez que conozcamos las plataformas y profundizaremos que hace cada una.

## 3.1. Plataformas

Las principales plataformas que utilizaremos en este trabajo serán Matlab y C.

- Matlab: En Matlab, nos encargaremos básicamente de cargar los datos para poder enviarlos en el código C y recibiremos desde C los datos donde calcularemos los parámetros. Como haremos una comparación entre los parámetros de Matlab y C, también calcularemos los coeficientes en Matlab para comprobar que dan lo mismo.
- C: En C, nos encargaremos de ejecutar las operaciones para poder obtener los resultados de los coeficientes. La comparación que haremos, es para ver quien calcula más rápidos los coeficientes entre los dos lenguajes.

A parte de estos dos lenguajes, hay que destacar el Mex function que es la función que se encargará de pasar los datos entre Matlab y C, y una vez tengamos obtenidos los datos de C, volveremos a pasar estos datos en Matlab para obtener el valor de los parámetros.

**Fig 3. 1.** Plataformas del proyecto

## 3.2. Matlab

En Matlab, como hemos explicado anteriormente, lo hemos utilizado para cargar los datos. A continuación, explicaremos que hacemos en el código (Nota: el código tanto en Matlab como en C lo adjuntaremos en el anexo).

Cargaremos los datos donde obtendremos dos matrices, una será con los datos de la señal de entrada (xBB) y otro será con la señal de salida (yBB). La matriz que viene definida en los datos, es de 12288 filas y 1 columna. Como el número de columnas es muy pequeño, fijaremos el número de filas que viene predeterminada y el número de columnas lo iremos variando mediante un número de coeficientes y un vector que lo llamaremos delay. Las posiciones que tenga este vector multiplicado por el número de coeficientes será el total de columnas que tengamos y crearemos una nueva señal de entrada.

Una vez tengamos creadas la señal de entrada, procederemos a calcular el vector w para obtener los coeficientes de cada tipo de LMS, que debería de ser igual a lo que obtengamos en C.

El vector w tendrá el mismo tamaño que columnas hayamos obtenido en la señal de entrada. Una vez definido mediremos tanto el tiempo en C como en Matlab, calcularemos los coeficientes en una función de Matlab y C y que seguirá el procedimiento que hemos explicado en el capítulo anterior.

Una vez obtenidos estos coeficientes, obtendremos la señal salida y la compararemos con la señal deseada (yBB). Con estos dos datos, calcularemos el valor de NMSE y predeciremos si es ideal la señal salida calculada.

## 3.3. Función Mex

Los archivos MEX son subrutinas que hace de mediador entre MATLAB y C. La información la cargamos en MATLAB y ejecutamos la función en C. Explicaremos las razones porque utilizamos los MEX files, como funcionan y la estructura que tienen.

### 3.3.1. Introducción

MEX(Matlab EXecutable) Files están escritos en C, C++ o Fortran en un determinado estándar y luego se compila utilizando la función incorporada de MEX de Matlab y uno de los compiladores externos compatibles.

El código Matlab puede llamar directamente a funciones MEX en la ruta MATLAB, al igual que otro archivo m o función. Cuando el código m de Matlab invoca una función MEX, la mexFunction llama los argumentos de entrada específicos y los argumentos de salidas pasan de nuevo a Matlab [5].

La estructura del mexFunction es la siguiente:

```
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
  // C Code
}
```

**Fig 3. 2.** Función mexFunction

Los argumentos del Mex Function tiene el siguiente significado:

- Nlhs: es el número de salidas que tiene la función que enviamos.
- Plhs: Puntero de una matriz que mantendrá los datos de salida.
- Nrhs: es el número de entradas que tiene la función que enviamos.
- Prhs: puntero de una matriz que mantendrá los datos de entrada.

El código que tenemos que escribir en el MEX Function, tenemos que inicializar los inputs y outputs que declaramos en la función que ejecutamos en Matlab. También tenemos que saber cosas importantes que tendríamos que hacer, para que MEX Function nos funcione correctamente:

- Los parámetros prhs, plhs, nrhs y nlhs son necesarios en el MEX Function, ya que de lo contrario no funcionaría.
- También se requiere el archivo de cabecera, mex.h, que declara el punto de entrada y las rutinas de interfaz.
- El nombre del archivo del Gateway será el mismo nombre que le damos al comando en Matlab.[6]

### 3.3.2. Razones

A continuación, enumeramos una serie de razones porque utilizamos Mex files:

- Los archivos MEX pueden ser muy importantes para el rendimiento de MATLAB, ya que se ejecuta casi a la velocidad del código C. La única razón por la que se ejecuta un poco más lento es la pequeña sobrecarga asociada con la interfaz de MATLAB y las llamadas a la API de MEX.
- MEX permite a los programadores sofisticados la aceleración adicional debido a la manipulación de MATLAB y la memoria nativa in situ, pasando por alto el mecanismo de copia a escribir (COW)* de MATLAB y la necesidad de asignar y desasignar memoria al manipular matrices de datos.
- Los archivos MEX también permiten a los programadores utilizar con facilidad la interfaz externa del código C y estáticamente enlace con bibliotecas externas. Esto evita la necesidad de utilizar bibliotecas compartidas/dinámicas que necesiten un manejo especial para ser cargadas e invocadas en MATLAB.

### 3.3.3. Implementación

En la figura 3.3. [7], se explica cómo implementar los datos de Matlab a C y después devolver la respuesta de nuevo a Matlab.

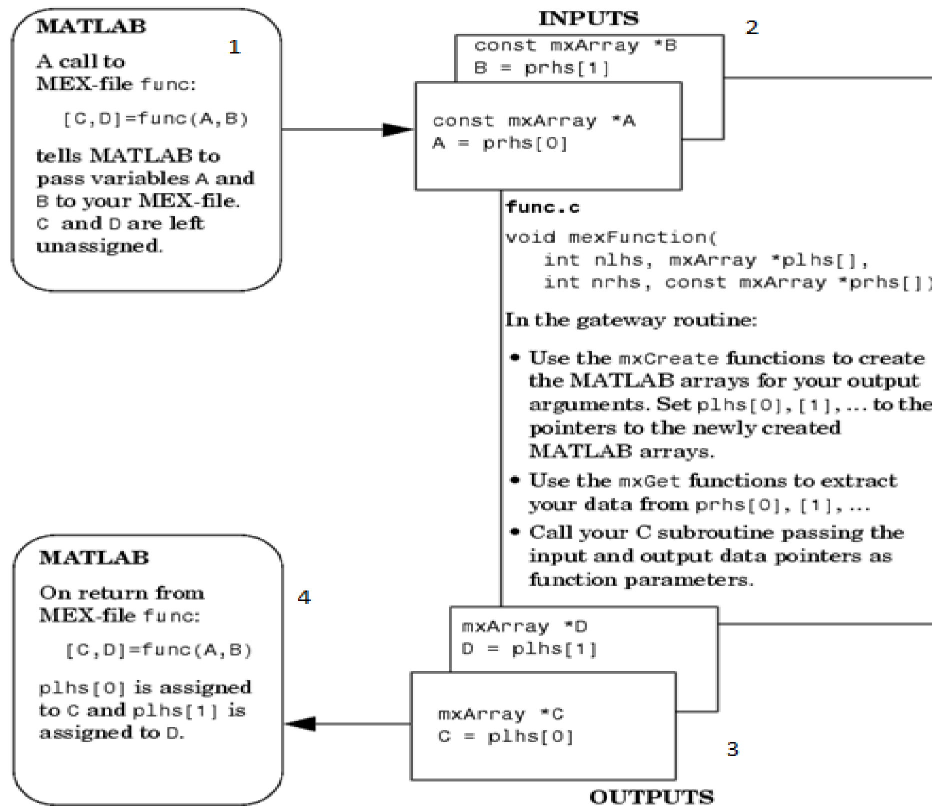


**Fig 3. 3.** Proceso que sigue MEX

Introducimos los parámetros de entrada en Matlab mientras que los parámetros de salida quedan esperando al resultado que dé en C.

En el código C, en la parte de Gateway asignamos los parámetros de entrada con el prhs[n] y en el código C, en la parte de Gateway asignamos los parámetros de salida con el plhs[n]. Una vez tengamos todos los datos, desde el código C calcularemos los parámetros de salida

Tras encontrar los valores salida de nuestro código C, devolvemos los resultados a Matlab.

### 3.3.4. Función

En la figura 3.4., representamos lo que será nuestra mexFunction que ejecutará nuestros datos de Matlab.

Aunque en Matlab ya hemos cargado los datos que necesitamos para ejecutarlo en C, con el mexFunction los traducimos para que podamos ejecutarlo con el C.

```
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
    double in_mu;
    double *in_Xr,*in_Xi,*in_yr,*in_yi,*in_wr,*in_wi;
    double *out_wr,*out_wi;
    size_t in_Nrows, in_Ncols;


    /* inputs */
    in_Xr = mxGetPr(prhs[0]);
    in_Xi = mxGetPi(prhs[0]);
    in_yr = mxGetPr(prhs[1]);
    in_yi = mxGetPi(prhs[1]);
    in_Nrows = mxGetScalar(prhs[2]);
    in_Ncols = mxGetScalar(prhs[3]);
    in_mu = mxGetScalar(prhs[4]);
    in_wr = mxGetPr(prhs[5]);
    in_wi = mxGetPi(prhs[5]);


    /* outputs */
    plhs[0] = mxCreateDoubleMatrix((mwSize)in_Ncols,1,mxCOMPLEX);
    out_wr = mxGetPr(plhs[0]);
    out_wi = mxGetPi(plhs[0]);

    /* call the computational routine */
    C_SLMS(in_Xr,in_Xi,in_yr,in_yi,in_Nrows,in_Ncols,in_mu,in_wr,in_wi,out_wr,out_wi);
}
```

**Fig 3. 4.** La función Mex que utilizaremos para cada tipo de LMS

## 3.4. C

Con C, nos encargaremos solamente de calcular los coeficientes como hemos indicado en el procedimiento de cada tipo de LMS. Obtendremos los datos desde Matlab mediante la función MEX. Los parámetros como el tiempo que tarda en ejecutar la función o NMSE, también lo calcularemos en Matlab.

Tenemos que aclarar que como en C no podemos operar en números complejos, separamos la parte real e imaginaria en dos y las operaciones las haremos independiente de las dos.

# CAPÍTULO 4: PARAMETROS

En este capítulo, hablaremos de los parámetros que tenemos que obtener.

## 4.1. NMSE

El Normalised Mean Square Error o NMSE es una estimación donde hacemos la diferencia de las señales de entrada y salida. La expresión del NMSE es:

$$NMSE(dB) = 10 \cdot log_{10} \frac{1}{N} \sum_{n=1}^{N} \left| \frac{|y_{in}[n]| - |y_{out}[n]|}{|y_{in}[n]|} \right|^2 \quad \textbf{(4.1)}$$

En la fórmula de la NMSE, $y_{in}$ es la señal de entrada, $y_{out}$ es la señal de salida y N es el número de muestras. Utilizaremos varios parámetros μ y con cada tipo de LMS, miraremos cuál es mejor en cada caso. El valor debe de tender a menos infinito, para indicar que tenemos una mínima diferencia entre la señal de entrada y de salida.

## 4.2. Least Square

El principal objetivo del Least Square es con un vector de muestras de señal de entrada(x[0] … x[N]) y otro vector de señal de salida (y[0]… y[N]) calculamos los coeficientes w[n] para que se pueda ajustar al modelo del error cuadrático. Hacemos una comparación y[n] (que es la señal deseada) y ŷ[n] (que es la señal que obtenemos a partir de calcular los coeficientes). La fórmula para calcular ŷ[n] es la siguiente:

$$\hat{y}[n] = X \cdot w[n] \quad \textbf{(4.2)}$$

Con los datos obtenidos de y[n] y de ŷ[n], miramos la estimación que podemos obtener en la gráfica.

En la figura 4.1., podemos ver la representación del Least Square:

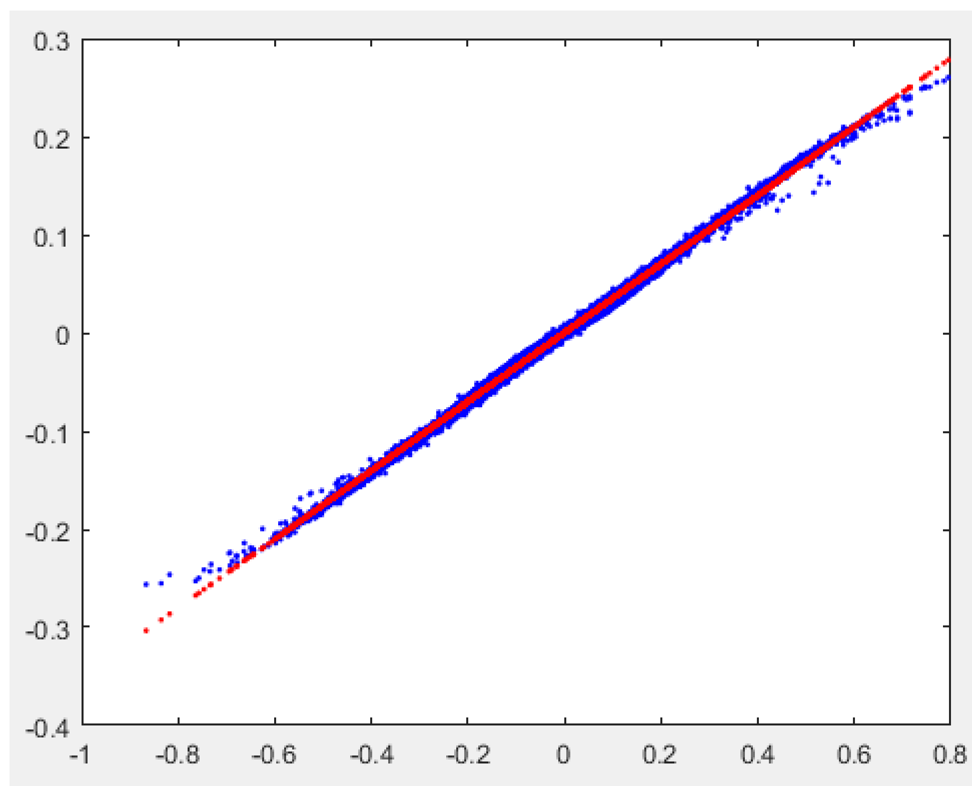


**Fig 4. 1.** Predicción entre la salida deseada y la calculada con los coeficientes obtenidos

# CAPÍTULO 5: RESULTADOS

En esta parte, vamos a explicar los resultados obtenidos a partir del código.

La primera estimación es el tiempo que tardamos en calcular los coeficientes en Matlab para cada tipo de LMS. Compararemos los valores absolutos que hemos obtenido con el valor complejo.

**Tabla 5.1.** El tiempo (en seg.) que tarda en ejecutarse cada LMS

| Nº Columnas | LMS absolute | LMS complex | NLMS absolute | NLMS complex | Sign LMS absolute | Sign LMS complex |
|---|---|---|---|---|---|---|
| 1 | 0,1978 | 0,2251 | 0,2431 | 0,4557 | 0,2072 | 0,216 |
| 3 | 0,2454 | 0,264 | 0,2869 | 0,3065 | 0,2527 | 0,2666 |
| 9 | 0,2457 | 0,2673 | 0,2886 | 0,3071 | 0,2574 | 0,2725 |
| 25 | 0,2535 | 0,2811 | 0,2975 | 0,3211 | 0,2741 | 0,2808 |
| 55 | 0,2685 | 0,3062 | 0,3301 | 0,3622 | 0,2782 | 0,3073 |
| 91 | 0,2816 | 0,3358 | 0,3474 | 0,3954 | 0,2915 | 0,3404 |
| 210 | 0,3332 | 0,4332 | 0,4316 | 0,5033 | 0,3457 | 0,4197 |
| 525 | 0,527 | 0,8283 | 0,7082 | 0,9722 | 0,5294 | 0,8297 |
| 1050 | 0,842 | 1,3032 | 1,055 | 1,4943 | 0,8669 | 1,2991 |
| 2100 | 1,4264 | 2,2226 | 1,706 | 2,5453 | 1,4027 | 2,206 |

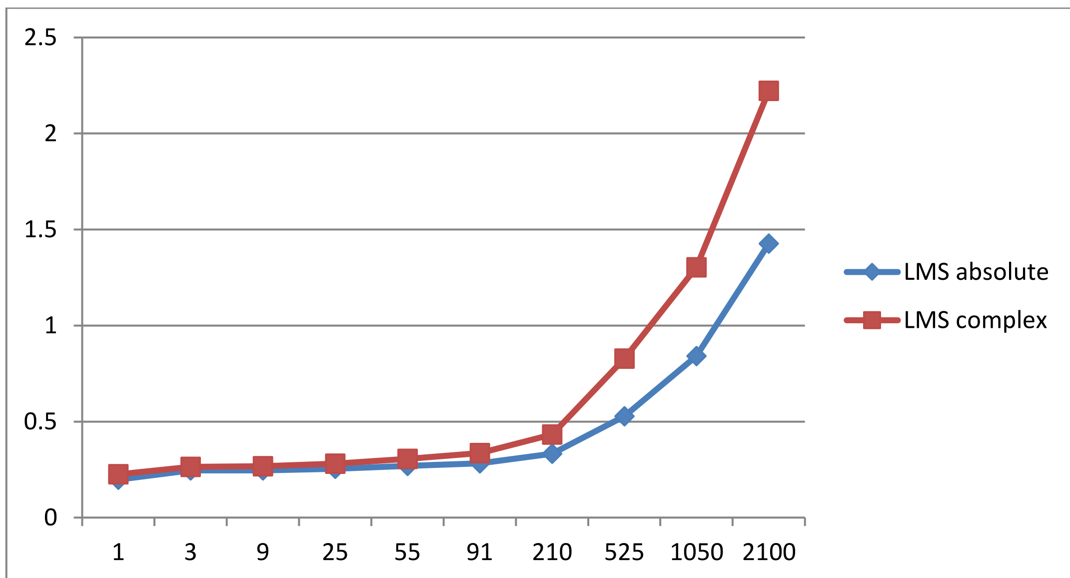


**Fig 5. 1.** Gráfica del LMS con datos absolutos y complejos

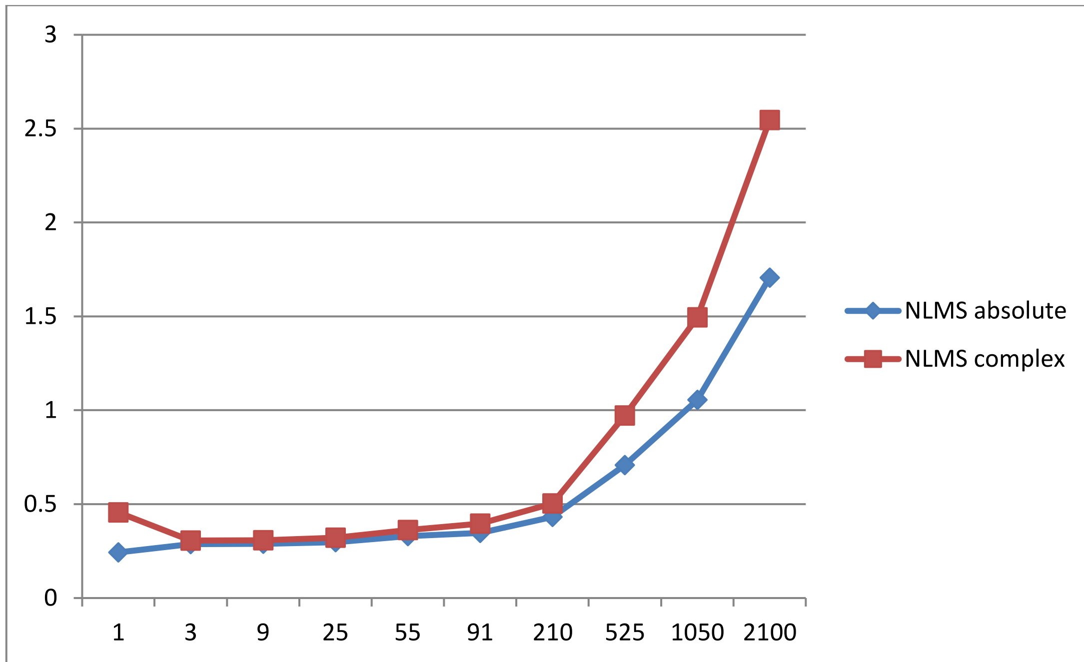


**Fig 5. 2.** Gráfica del NLMS con datos absolutos y complejos

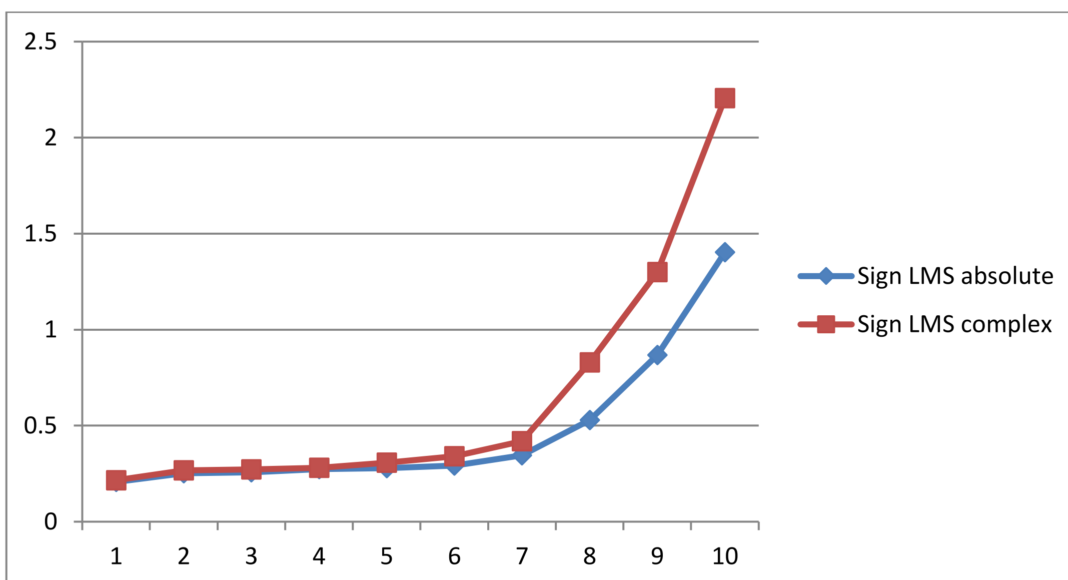


**Fig 5. 3.** Gráfica del Sign LMS con datos absolutos y complejos

Tardamos menos tiempo con el valor absoluto porque pierde la parte imaginaria del valor complejo, que por otra parte no nos interesará ya que a la hora de trazar la predicción entre la señal deseada y la señal que obtenemos, no nos trazará tan bien la predicción.

Por lo tanto, hemos decidido hacer una estimación en tiempo comparando Matlab y C con valores complejos para ver cuál es el que tarda menos.

**Tabla 5.2.** El tiempo (en seg.) que tarda en ejecutarse cada LMS (Matlab y C)

| Nº Columnas | LMS Matlab | LMS complex C | NLMS Matlab | NLMS C | Sign LMS Matlab | Sign LMS C |
|---|---|---|---|---|---|---|
| 1 | 0,2251 | 0,001695 | 0,4557 | 0,00346 | 0,216 | 0,001675 |
| 3 | 0,264 | 0,00166 | 0,3065 | 0,002972 | 0,2666 | 0,001949 |
| 9 | 0,2673 | 0,002114 | 0,3071 | 0,005251 | 0,2725 | 0,00225 |
| 25 | 0,2811 | 0,004935 | 0,3211 | 0,01186 | 0,2808 | 0,003411 |
| 55 | 0,3062 | 0,005665 | 0,3622 | 0,024 | 0,3073 | 0,005775 |
| 91 | 0,3358 | 0,008906 | 0,3954 | 0,0388 | 0,3404 | 0,008762 |
| 210 | 0,4332 | 0,01853 | 0,5033 | 0,0871 | 0,4197 | 0,01822 |
| 525 | 0,8283 | 0,04334 | 0,9722 | 0,2167 | 0,8297 | 0,04385 |
| 1050 | 1,3032 | 0,08151 | 1,4943 | 0,4324 | 1,2991 | 0,09186 |
| 2100 | 2,2226 | 0,1726 | 2,5453 | 1,0864 | 2,206 | 0,1671 |

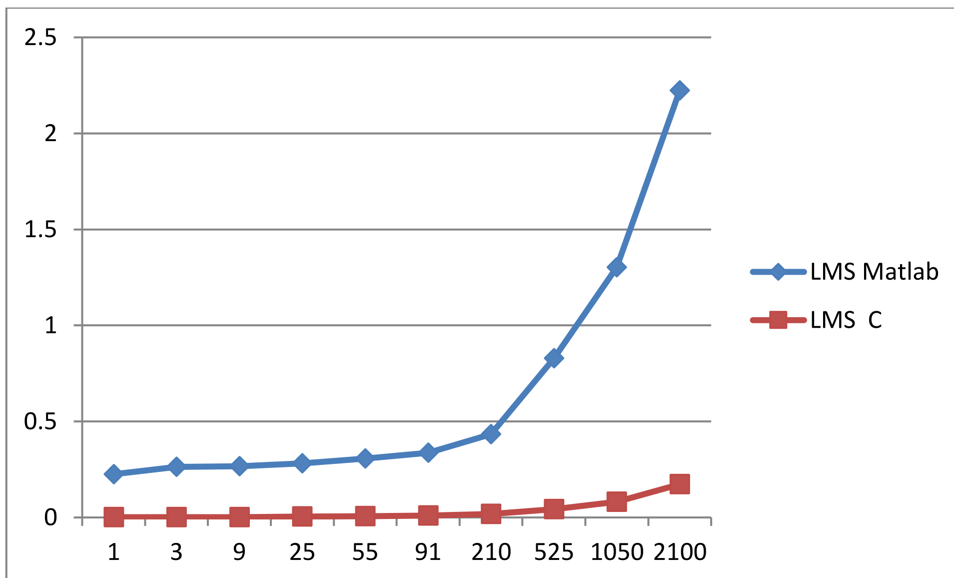


**Fig 5. 4.** Gráfica del LMS con datos en Matlab y C

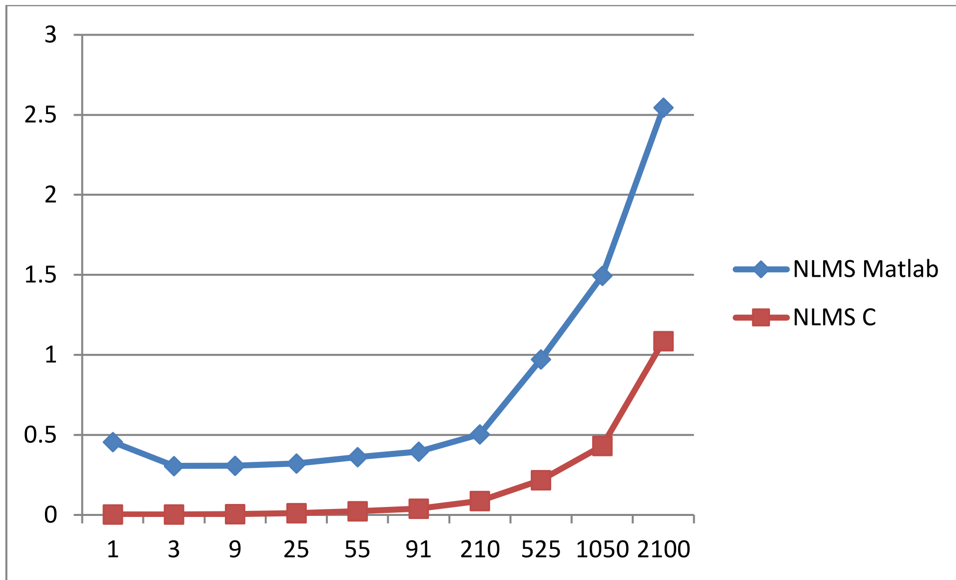


**Fig 5. 5.** Gráfica del LMS con datos en Matlab y C

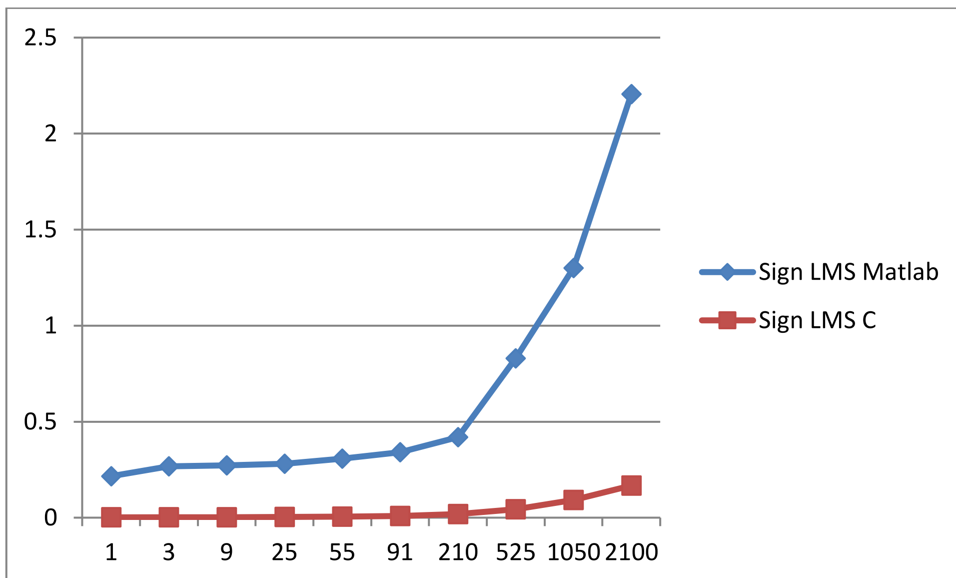


**Fig 5. 6.** Gráfica del LMS con datos en Matlab y C

Como podemos ver, el código en C es más rápido que en Matlab debido a que Matlab es un lenguaje interpretado (traduce cada instrucción y la ejecuta en el momento), en cambio C lo traduce a un programa equivalente para que la máquina sea capaz de interpretarlo.

Otra comparativa que se puede hacer es la velocidad entre el código C y Matlab. Para ello, usamos la siguiente fórmula:

$$Speed = \frac{Tiempo\ en\ Matlab}{Tiempo\ en\ C} \quad \textbf{(5.1)}$$

Con esta fórmula determinaremos cuantas veces es más rápido el código C que el código en Matlab.

**Tabla 5.3**. Velocidad de cada tipo de LMS

| Nº Columnas | Speed LMS | Speed NLMS | Speed Sign LMS |
|---|---|---|---|
| 1 | 132,8023599 | 131,7052023 | 128,9552239 |
| 3 | 159,0361446 | 103,1292059 | 136,7880965 |
| 9 | 126,4427625 | 58,48409827 | 121,1111111 |
| 25 | 56,96048632 | 27,07419899 | 82,32189974 |
| 55 | 54,05119153 | 15,09166667 | 53,21212121 |
| 91 | 37,70491803 | 10,19072165 | 38,84957772 |
| 210 | 23,37830545 | 5,778415614 | 23,03512623 |
| 525 | 19,11167513 | 4,48638671 | 18,92132269 |
| 1050 | 15,9882223 | 3,455827937 | 14,14217287 |
| 2100 | 12,87717265 | 2,342875552 | 13,20167564 |

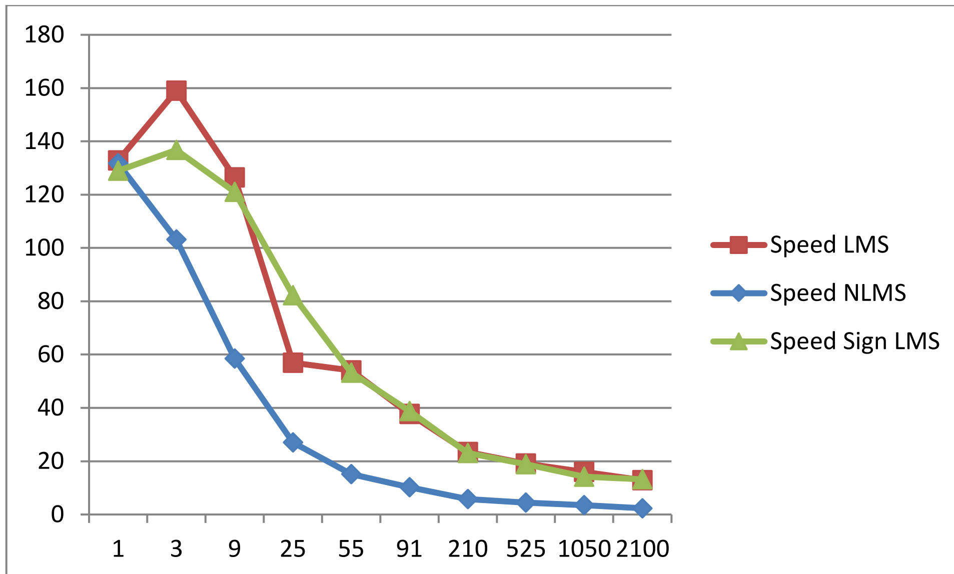


**Fig 5. 7.** Velocidad de LMS obtenidas a partir del speed

Como podemos observar en los datos de la tabla 5.3. y de la figura 5.7., cuando aumentamos el número de columnas, vemos que la diferencia entre las velocidades se va acercando hasta llegar que speed < 20.

Otro cosa que podemos observar, es que la velocidad del NLMS está por debajo de los otros dos tipos, ya que el código que hay en C es más complejo que los otros dos tipos y por eso tarda más en ejecutarse.

El siguiente punto a observar es la gráfica del LMS con el error que hay en cada columna. Como hicimos anteriormente, vamos a comparar el valor absoluto con el complejo de cada tipo de LMS:

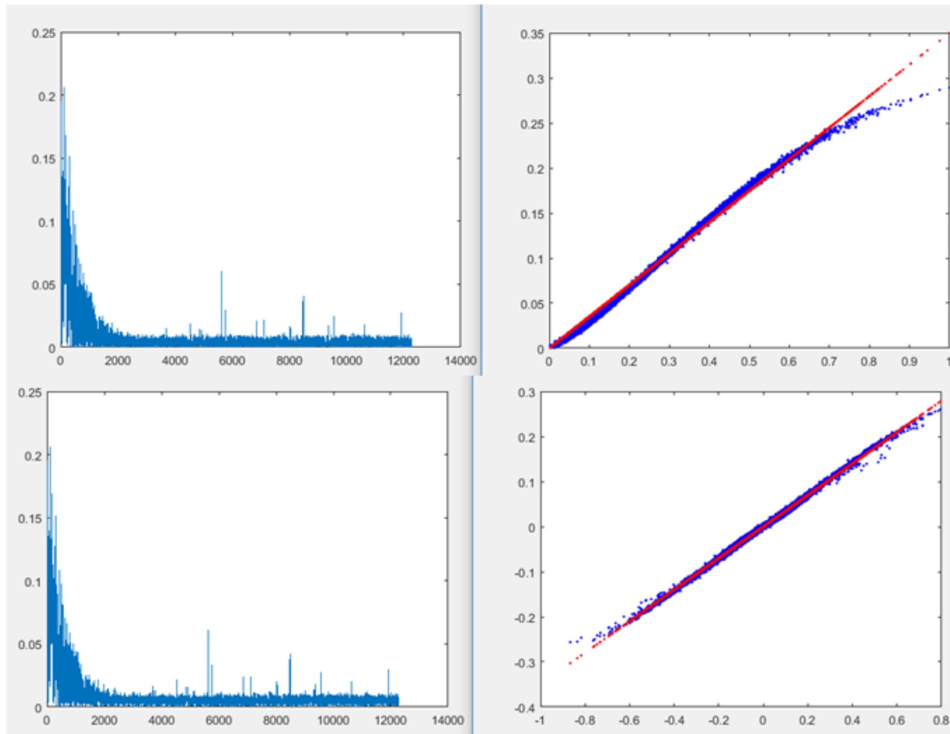


**Fig 5. 8.**Predicción y error del LMS en absoluto y complejo

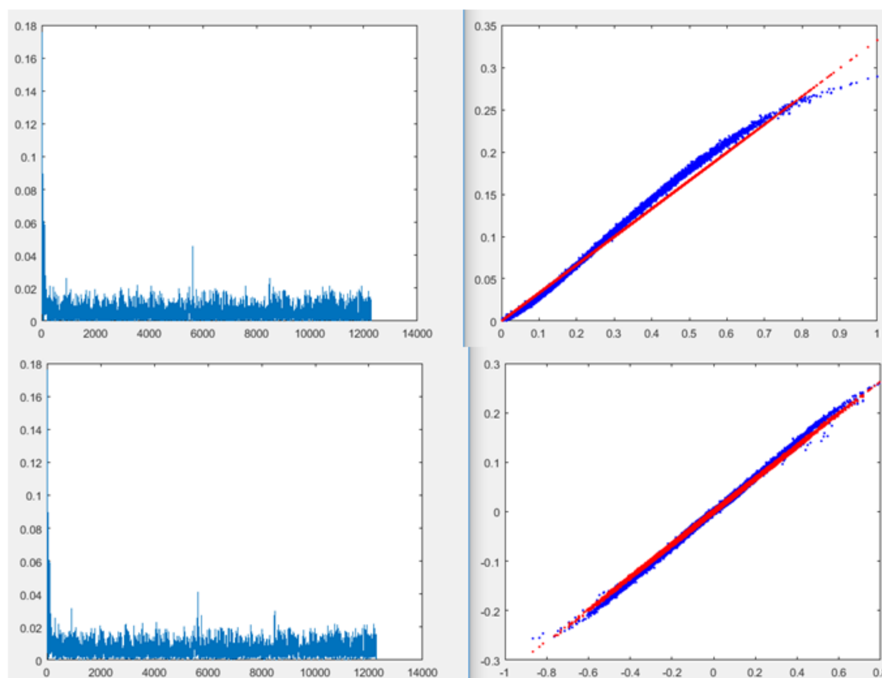


**Fig 5. 9.** Predicción y error del NLMS en absoluto y complejo

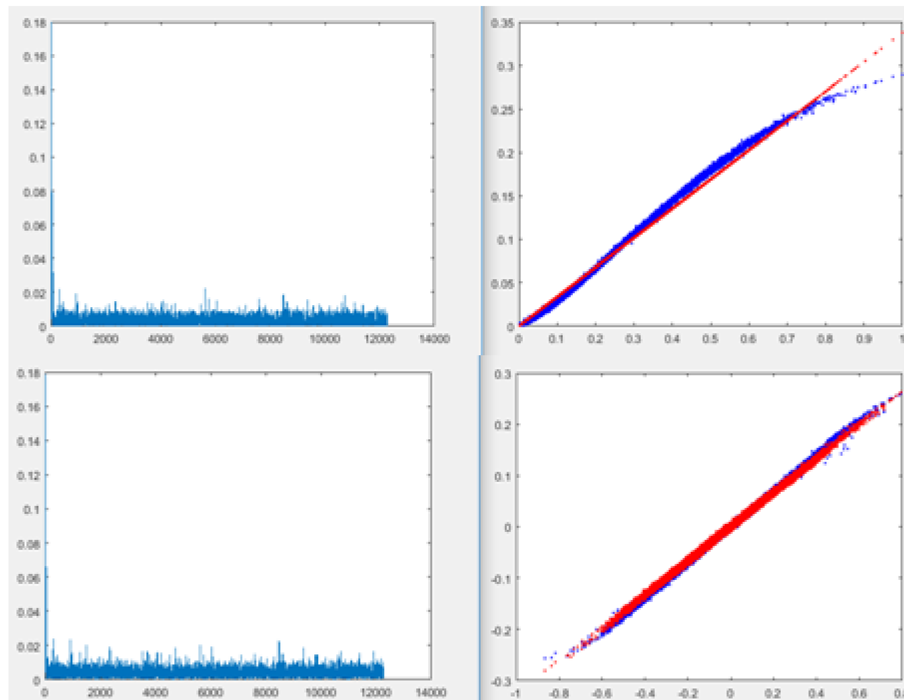


**Fig 5. 10.** Predicción y error del Sign LMS en absoluto y complejo

Podemos confirmar, que aunque en valor absoluto calculemos más rápido los coeficientes, obtenemos más error ya que se pierde la parte compleja y la gráfica donde representamos entre la salida deseada y la salida que obtenemos, con números complejos se asemeja más a lo que queremos obtener.

A continuación, calculamos cada NMSE para cada tipo de LMS. Haremos 3 tablas variando el parámetro μ. En este caso el valor de μ será de 0.015,0.05 y 0,1.

Como hicimos para medir el tiempo, fijaremos el valor que contenga el número de filas.

| Nº Columnas | NMSE LMS | NMSE NLMS | NMSE Sign LMS |
|---|---|---|---|
| 1 | -25,441 | -24,669 | -22,75 |
| 3 | -25,808 | -25,461 | -22,552 |
| 9 | -26,557 | -22,708 | -27,015 |
| 25 | -25,232 | -23,48 | -22,985 |
| 55 | -28,171 | -24,701 | -18,88 |
| 91 | -27,977 | -24,855 | -20,41 |
| 210 | -28,625 | -25,036 | -17,925 |
| 525 | -28,392 | -25,06 | -17,885 |
| 1050 | -28,341 | -25,042 | -17,183 |
| 2100 | -28,257 | -25,024 | -9,701 |

| Nº Columnas | NMSE LMS | NMSE NLMS | NMSE Sign LMS |
|---|---|---|---|
| 1 | -25,44 | -24,738 | -21,132 |
| 3 | -25,83 | -25,422 | -25,632 |
| 9 | -27,386 | -28 | -16,666 |
| 25 | -28,049 | -20,282 | -22,14 |
| 55 | -28,982 | -23,415 | -24,01 |
| 91 | -28,73 | -19,941 | -19,712 |
| 210 | -28,775 | -19 | -19,5 |
| 525 | -1,192 | -20,276 | -11,023 |
| 1050 | 3,08 | -20,141 | -4,738 |
| 2100 | 3,028 | -19,972 | -1,902 |

| Nº Columnas | NMSE LMS | NMSE NLMS | NMSE Sign LMS |
|---|---|---|---|
| 1 | -25,441 | -25,441 | -21,46 |
| 3 | -25,847 | -25,125 | -25,81 |
| 9 | -27,644 | -29,571 | -27,77 |
| 25 | -29,571 | -19,097 | -22,722 |
| 55 | -29,23 | -22,534 | -24,507 |
| 91 | -28,81 | -20,678 | -22,966 |
| 210 | 3,27 | -22,57 | -23,197 |
| 525 | 3,232 | -21,603 | -10,333 |
| 1050 | 3,196 | -20,81 | -5,343 |
| 2100 | 3,21 | -20,693 | 0,635 |

**Fig 5. 11.** Tabla de la NMSE, con valor de μ=0.015, μ=0,05 y μ=0,1

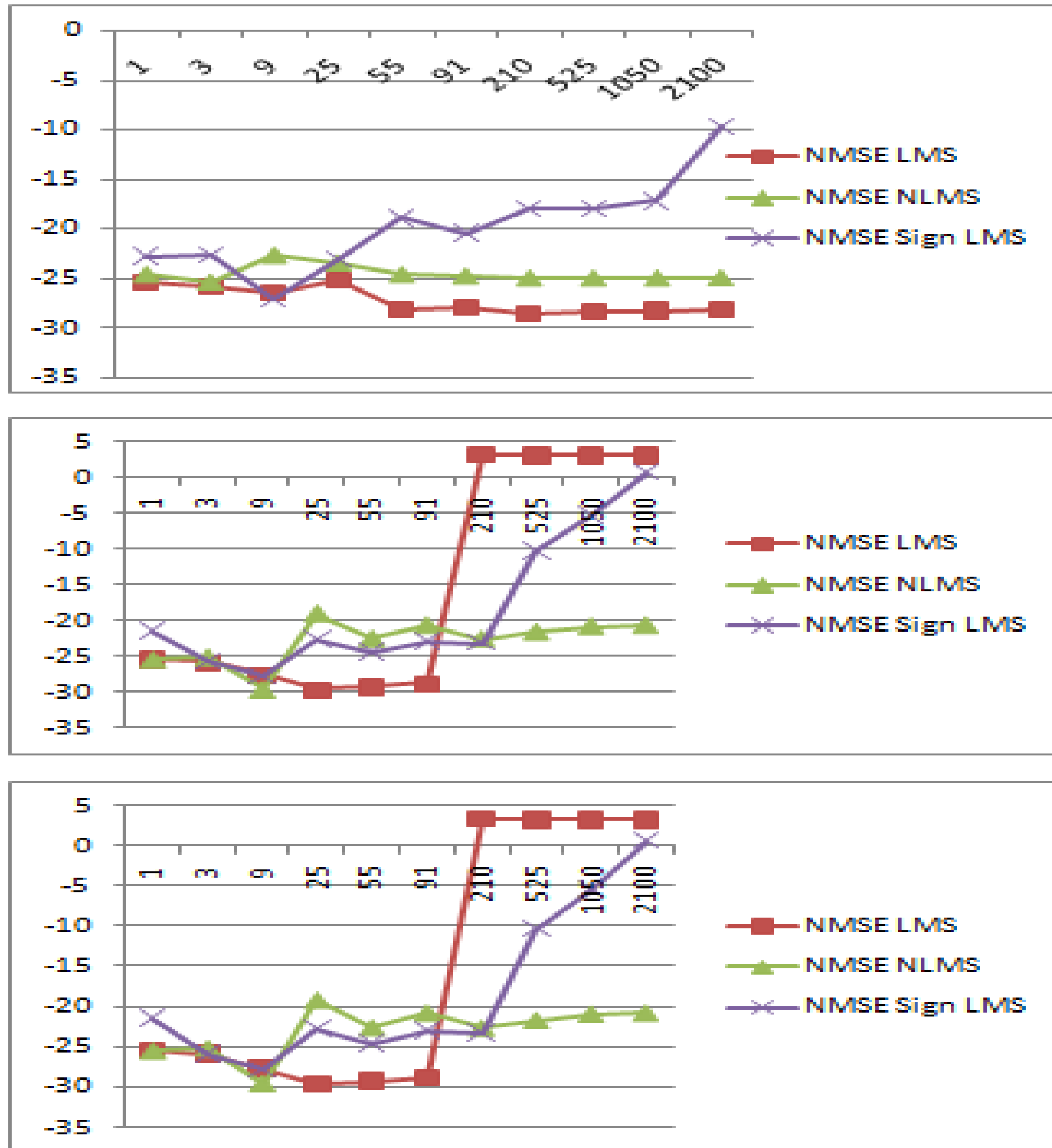


**Fig 5. 12.** Gráfica del NMSE, con valor de μ=0.015, μ=0.05 y μ=0.1

Con la variación de μ, podemos observar que a mayor μ, obtenemos una NMSE mayor. No nos interesa ya que tendremos un error más grande. También podemos observar que cuantos más números de columnas, el NMSE también incrementa.

Como muestran las figuras 5.11. y 5.12., el que menos impacto recibe con la variación de μ y de número de columnas es el NLMS.

# CONCLUSIONES

Como conclusiones finales, hemos profundizado en el funcionamiento de 3 tipos de LMS.

El Standard LMS a nivel de programación, como hemos comentado en el capítulo de teoría, es el más sencillo de todos debido a que los otros dos tipos han sufrido una variación respecto al Standard. Por su contra, el Normalised LMS es el más complejo.

A nivel de tiempo y velocidad (medida entre los tiempo de Matlab y C), los parámetros más importantes para tomar las medidas ha sido el número de columnas y si tenemos en valor absoluto o complejo. El número de columnas cuanto más grande más tardamos en calcular los coeficientes para obtener la curva entre la salida deseada y la salida que obtenemos, nuestra velocidad es peor y obtenemos más error. En valor absoluto calculamos más rápido aunque perdemos la parte imaginaria cosa que no nos interesa ya que la curva que obtenemos es menos precisa, por lo tanto, aunque tardemos más tiempo nos interesa tener valores reales y complejos. El Standard LMS y el Sign LMS tardan el mismo tiempo que es menor al del NLMS. La velocidad es menor en NLMS y mayor en Standard LMS y Sign LMS.

Aunque es muy importante emplear el mínimo tiempo, también es muy importante el NMSE que obtenemos. Los parámetros que hemos modificado para estimar el NMSE han sido el número de columnas y la $\mu$. El parámetro $\mu$ cuanto más pequeño mejor debido a que el incremento de este parámetro puede hacer que tengamos una NMSE muy grande, cosa que no deseamos. El número de columnas también hace incrementar el NMSE debido a que tenemos más datos y como el cálculo del coeficiente no es exacto para obtener la señal deseada cada vez arrastraremos un error más grande. En este caso, el NMSE es mejor debido a que la estimación entre la señal de entrada y de salida son más parecidos que los otros dos tipos y esto producirá que el NMSE que obtengamos sea más pequeño.

# BIBLIOGRAFIA

[1] Douglas, S.C. “Introduction to Adaptive Filters” Digital Signal Processing Handbook Ed. Vijay K. Madisetti and Douglas B. Williams Boca Raton: CRC Press LLC, 1999, Chapter 18.

[2] Pere L. Gilabert. “Chapter 5: Linear and Nonlinear distortion Characterization and Compensation”

[3] Paulo S. R. Diniz “Adaptative Filter, Algorithm and Practical Implementation”, Fourth Edition, Chapter 2.

[4]https://en.wikipedia.org/wiki/Least_mean_squares_filter#/media/File:Lms_filter.svg

[5] Altman, Yair “Accelerating Matlab Performance: 1001 tips to speed up Matlab programs”: CRC Press, Chapter 8

[6] Axelsson, Maria “An introduction to Matlab Mex-files”

[7] http://matlab.izmiran.ru/help/techdoc/matlab_external/ch5_for3.gif

# ANEXOS

## A.1. Código en MATLAB de datos

```
clear all; clc; close all;
rootpath=cd;
addpath(rootpath);
addpath([rootpath '\data']);
addpath([rootpath '\toolbox.utilities']);
addpath([rootpath '\toolbox.mexcode']);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%
load('PAcreeLTE20M.mat');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%
%plot(abs(xBB),abs(yBB),'.b')

Ncoef_poly=1; delays_poly=[-1:1:1];
X=gml_LS_Mgeneration('MEMPOLY2',xBB,Ncoef_poly,delays_poly,0,0);

X=xBB;
Y=yBB;

% X=X(1:20,1:3);
% yBB=yBB(1:20,1);

[Nrows,Ncols]=size(X)

mu=0.05;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%
%%%%% Least Squares solution by matrix inversion
disp(' '); disp('LS: Doing matrix inversion');
tic
w_reference=((X'*X)^-1)*X'*yBB;
toc
fprintf('-
>NMSE= %f\n',dpd_Qmeasurements(yBB,X*w_reference,'NMSE'));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%
%%%%% LS solved with LMS in .m
disp(' '); disp('LS: Doing LMS');
Tstart=tic;
w_LMS=complex(zeros(Ncols,1));
%En caso que usemos Standard LMS
[w_LMS]=M_LMS(X,Y,Nrows,Ncols,mu,w_LMS);
```

```
%En caso que usemos Normalised LMS
[w_LMS]=M_NLMS(X,Y,Nrows,Ncols,mu,w_LMS);
%En caso que usemos Sign LMS
[w_LMS]=M_SLMS(X,Y,Nrows,Ncols,mu,w_LMS);
%%%%%%
T_LMS=toc(Tstart);
fprintf('->time= %f\n',T_LMS);
figure; plot(X,Y,'.b');
hold on; plot(X,X*w_LMS,'.r')
fprintf('->NMSE= %f\n',dpd_Qmeasurements(yBB,X*w_LMS,'NMSE'));
fprintf('->NMSE of w (reference vs
m)= %f\n',dpd_Qmeasurements(w_reference,w_LMS,'NMSE'));



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%
%%%%% LS solved with LMS in C
disp(' '); disp('LS: Doing LMS in C')
Cpath=[rootpath '\toolbox.mexcode\'];
cd(Cpath)
mex C_LMS.c
cd(rootpath)

X_CLMS=complex(transpose(X));
yBB_CLMS=complex(yBB);
w_CLMS=complex(zeros(Ncols,1));

Tstart=tic;
%En caso que usemos Standard LMS
[w_CLMS]=C_LMS(X_CLMS,yBB_CLMS,Nrows,Ncols,mu,w_CLMS);
%En caso que usemos Normalised LMS
[w_CLMS]=C_NLMS(X_CLMS,yBB_CLMS,Nrows,Ncols,mu,w_CLMS);
%En caso que usemos Sign LMS
[w_CLMS]=M_SLMS(X,Y,Nrows,Ncols,mu,w_LMS);
%%%
T_CLMS=toc(Tstart);
fprintf('->time= %f\n',T_CLMS);
fprintf('->speed_up= %f\n',T_LMS/T_CLMS);
figure; plot(X,Y,'.b');
hold on; plot(X,X*w_CLMS,'.r')
fprintf('->NMSE
post= %f\n',dpd_Qmeasurements(yBB,X*w_CLMS,'NMSE'));
fprintf('->NMSE of w (m vs
c)= %f\n',dpd_Qmeasurements(w_LMS,w_CLMS,'NMSE'));
```

## A.2. Código en Matlab para ejecutar lostipos LMS

### A.2.1. Standard LMS

```
function [w]=M_LMS(X,y,Nrows,Ncols,mugain,w)

y_est=zeros(Nrows,1);
error=zeros(Nrows,1);



for ncol=1:Nrows
    new_row=X(ncol,:);
    y_est(ncol)=new_row*w;
error(ncol)=y(ncol)-y_est(ncol);
deltaw=new_row'*error(ncol);
    w=w+deltaw*mugain;

end

figure; plot(abs(error));

end
```

### A.2.2. Normalised LMS

```
function [w]=M_NLMS(X,y,Nrows,Ncols,mugain,w)

y_est=zeros(Nrows,1);
error=zeros(Nrows,1);

for ncol=1:Nrows
    new_row=X(ncol,:);
    y_est(ncol)=new_row*w;
error(ncol)=y(ncol)-y_est(ncol);
deltaw=(new_row'*error(ncol))/(new_row*new_row');
deltaw=deltaw(:,1);
    w=w+deltaw*mugain;
end

figure; plot(abs(error));

end
```

### A.2.3. Sign LMS

```
function [w]=M_SLMS(X,y,Nrows,Ncols,mugain,w)

y_est=zeros(Nrows,1);
```

```
error=zeros(Nrows,1);

for niter=1:Nrows
  new_row=X(niter,:);
  y_est(niter)=new_row*w;
error(niter)=y(niter)-y_est(niter);
deltaw=(sign(real(error(niter)))+j*sign(imag(error(niter))))*new
_row';
  w=w+deltaw*mugain;
end

figure; plot(abs(error));

end
```

## A.3. Código en C para ejecutar los tipos LMS

### A.3.1. Standard LMS

```
void C_LMS(double *Xr,double *Xi,double *yr,double *yi,size_t
Nrows, size_t Ncols,double mu,double *wr_ini,double
*wi_ini,double *wr,double *wi)
{
size_t nrow,ncol,pos;
double yr_est,yi_est,er,ei,dr,di;

for (ncol=0; ncol<Ncols; ncol++) {
    wr[ncol]=wr_ini[ncol];
    wi[ncol]=wi_ini[ncol];
}

for (nrow=0; nrow<Nrows; nrow++) {
//printf("nrow= %d \n",nrow);

yr_est=0;
    yi_est=0;
    er=0;
    ei=0;

for (ncol=0; ncol<Ncols; ncol++) {
        pos=(nrow*Ncols)+ncol;
        yr_est=yr_est+(wr[ncol]*Xr[pos]-wi[ncol]*Xi[pos]);
        yi_est=yi_est+(wr[ncol]*Xi[pos]+wi[ncol]*Xr[pos]);
    }

//x_dot_conjy(&yr_est,&yi_est,&Xr[nrow*Ncols],&Xi[nrow*Ncols],wr
,wi,Ncols);

    er=yr[nrow]-yr_est;
    ei=yi[nrow]-yi_est;

for (ncol=0; ncol<Ncols; ncol++) {
pos=(nrow*Ncols)+ncol;
        wr[ncol]=wr[ncol]+mu*(Xr[pos]*er+Xi[pos]*ei);
        wi[ncol]=wi[ncol]+mu*(Xr[pos]*ei-Xi[pos]*er);
```

```
}
} // end of rows processing
}
```

## A.3.2. Normalised LMS

```
void C_NLMS(double *Xr,double *Xi,double *yr,double *yi,size_t
Nrows, size_t Ncols,double mu,double *wr_ini,double
*wi_ini,double *wr,double *wi)
{

//Declaramos las variables
size_t nrow,ncol,pos;
double yr_est,yi_est,er,ei,dr,di,abs_val,abs_valr,abs_vali;


for (ncol=0; ncol<Ncols; ncol++) {
    wr[ncol]=wr_ini[ncol];
    wi[ncol]=wi_ini[ncol];
}

printf("nrow= %d \n",Nrows);
printf("ncol= %d \n",Ncols);

for (nrow=0; nrow<Nrows; nrow++) {
//printf("nrow= %d \n",nrow);

yr_est=0;
    yi_est=0;
    er=0;
    ei=0;
    abs_val=0;
    abs_valr=0;
    abs_vali=0;



for (ncol=0; ncol<Ncols; ncol++) {
        pos=(nrow*Ncols)+ncol;
//printf("%lf\n",Xr[pos]);
        yr_est=yr_est+(wr[ncol]*Xr[pos]-wi[ncol]*Xi[pos]);
        yi_est=yi_est+(wr[ncol]*Xi[pos]+wi[ncol]*Xr[pos]);
    }



    er=yr[nrow]-yr_est;
//printf("%f\n",er);
    ei=yi[nrow]-yi_est;
//printf("%f\n",ei);

for (ncol=0; ncol<Ncols; ncol++) {
        pos=(nrow*Ncols)+ncol;
        abs_valr=(Xr[pos]*Xr[pos]-Xi[pos]*Xi[pos]);
        abs_vali=2*Xr[pos]*Xi[pos];
```

```
//printf("%lf\n",abs_valr);
abs_valr=pow(abs_valr,2);
        abs_vali=pow(abs_vali,2);
//printf("%lf\n",abs_valr);
        abs_val=abs_val+sqrt(abs_valr+abs_vali);
//printf("The absolute value is : %lf\n",abs_val);
    }

for (ncol=0; ncol<Ncols; ncol++) {
//printf("The absolute value is : %lf\n",abs_val);
pos=(nrow*Ncols)+ncol;
        wr[ncol]=wr[ncol]+(mu*(Xr[pos]*er+Xi[pos]*ei))/abs_val;
        wi[ncol]=wi[ncol]+(mu*(Xr[pos]*ei-Xi[pos]*er))/abs_val;
    }
//printf("%f\n",wr[ncol]);
//printf("%f\n",wi[ncol]);
} // end of rows processing
}
```

## A.3.3. Sign LMS

```
void C_SLMS(double *Xr,double *Xi,double *yr,double *yi,size_t
Nrows, size_t Ncols,double mu,double *wr_ini,double
*wi_ini,double *wr,double *wi)
{

//Declaramos las variables
size_t nrow,ncol,pos;
double yr_est,yi_est,er,ei,dr,di;


//Hacemos un bucle para que w tenga todas las posiciones a 0

for (ncol=0; ncol<Ncols; ncol++) {
    wr[ncol]=wr_ini[ncol];
    wi[ncol]=wi_ini[ncol];
}

//printf("nrow= %d \n",Nrows);
//printf("ncol= %d \n",Ncols);

for (nrow=0; nrow<Nrows; nrow++) {
//printf("nrow= %d \n",nrow);

yr_est=0;
    yi_est=0;
    er=0;
    ei=0;

for (ncol=0; ncol<Ncols; ncol++) {
        pos=(nrow*Ncols)+ncol;
//printf("%lf\n",Xr[pos]);
        yr_est=yr_est+(wr[ncol]*Xr[pos]-wi[ncol]*Xi[pos]);
        yi_est=yi_est+(wr[ncol]*Xi[pos]+wi[ncol]*Xr[pos]);
    }
```

```
    er=yr[nrow]-yr_est;
//printf("Real part:%f\n",er);
    ei=yi[nrow]-yi_est;
//printf("Imag part:%f\n",ei);


   dr = copysignf(1.0, er);
   di = copysignf(1.0, ei);
//printf("The absolute value is : %lf\n",abs_val);

for (ncol=0; ncol<Ncols; ncol++) {

pos=(nrow*Ncols)+ncol;
        wr[ncol]=wr[ncol]+(mu*(Xr[pos]*dr+Xi[pos]*di));
//printf("Real part:%f\n",wr[ncol]);
wi[ncol]=wi[ncol]+(mu*(Xr[pos]*di-Xi[pos]*dr));
//printf("Imag part:%f\n",wi[ncol]);
}
}

}
```