UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño de sistema de filtrado para auxiliares auditivos mediante filtros adaptativos

TESIS

Que para obtener el título de

Ingeniera Mecatrónica

P R E S E N T A

Pamela Gerardo Suárez

DIRECTOR(A) DE TESIS

M.A. Luis Yair Bautista Blanco

Ciudad Universitaria, Cd. Mx., 2018

Agradecimientos

A mis papás, Alicia y Plinio, por apoyarme incansablemente a lo largo de mi vida, enseñarme a ser fuerte y a luchar por lo que quiero, compartirme sus enseñanzas, acompañarme en los momentos más difíciles y por amarme siempre.

A mi hermano, Plinio, por estar a mi lado siempre, compartir grandes momentos desde pequeño y hacerme saber lo que significo para él.

A mi novio, Kazu, por siempre creer en mí, escucharme, apoyarme e impulsarme a lograr mis sueños.

A mis amigos, por escucharme, aconsejarme y seguir conmigo a través de los años creando nuevos recuerdos.

A mis compañeros de la carrera, por los aprendizajes compartidos, las experiencias vividas y los lazos que formamos.

A mi asesor, Yair, por su apoyo no solo para realizar este trabajo, sino también a lo largo de la carrera como mi profesor, por los conocimientos que me aportó y la ayuda que me ofreció para crecer profesionalmente.

Índice

Resumen

La presente tesis plantea y evalúa una solución para los errores observados en los auxiliares auditivos empleando filtros adaptativos. Así mismo, la optimización del algoritmo para asegurar su convergencia y velocidad.

El trabajo se basa en la teoría de filtros digitales y filtro de Wiener, partiendo de ello para poder diseñar la mejor solución teórica, tomando en cuenta las características de los algoritmos existentes y de los modelos de filtros adaptativos para llevar a cabo la mejor elección.

Para la realización este proyecto se llevaron a cabo tres etapas: la primera, el diseño del filtro para el problema planteado y tomando lo requerimientos para el mismo; la segunda etapa consistió en la simulación del algoritmo adaptativo verificando los resultados obtenidos y la obtención de los parámetros específicos para la siguiente etapa, con la ayuda del software MATLAB R2013b. La tercera etapa consistió en validación del algoritmo en tiempo real con los parámetros previamente obtenidos y de igual manera, verificando su correcto funcionamiento.

1. Introducción

La idea de realizar este trabajo de tesis surge a partir de la necesidad de darle solución a un problema real y que se sufren las personas con limitaciones auditivas. Tomando como punto de arranque lo descrito anteriormente, lo primero fue identificar claramente lo que se quería resolver, esto se realiza en el capítulo 2, además de establecer los objetivos a los que se pretendía llegar con el trabajo y el alcance que se quería obtener para delimitar el trabajo.

Posteriormente, en el capítulo 3 se presenta toda la teoría relacionada con filtros adaptativos, comenzando por retomar un poco el procesamiento de señales, se describen los diferentes tipos de filtros en cuanto al rango de frecuencias que filtran, el tipo de señales: ya sea analógicas o digitales, y dentro de los digitales el tipo de filtro que conviene elegir. Particularmente de los filtros adaptativos se muestran los modelos y algoritmos existentes y las consideraciones a tomar para elegir lo más adecuado para la implementación, así como los parámetros que deben de seleccionarse para controlar ciertos aspectos de los filtros adaptativos.

Una vez conociendo todo lo anterior y habiendo elegido la configuración a implementar, en el capítulo 4 se muestra la simulación del algoritmo con señales pregrabadas para elegir los parámetros que entregaran mejores resultados, verificar la validez del algoritmo implementado y la viabilidad para su implementación en tiempo real con los parámetros que hayan actuado con mayor eficiencia.

Comprobada la viabilidad para la implementación en tiempo real, en el capítulo 5 se establecieron los requerimientos en cuanto a hardware y software y se procedió a su validación considerando toda la información recabada de la teoría y simulación hecha previamente.

Finalmente, los resultados obtenidos de la implementación en tiempo real se muestran en el capítulo 6, analizando detenidamente todo lo obtenido para sacar las conclusiones pertinentes del trabajo realizado y poder identificar las mejoras que se podrían hacer junto con el trabajo a futuro para lo desarrollado.

2. Problema de Investigación

2.1. Identificación y planteamiento del problema

Un problema que se ha presentado en los humanos y ciertamente con mayor frecuencia en estos tiempos, es la pérdida parcial o total de la audición producida por diversas causas, ya sea causas genéticas, complicaciones en el parto, algunas enfermedades infecciosas, infecciones crónicas del oído, el empleo de determinados fármacos, la exposición al ruido excesivo y el envejecimiento, según la OMS (Organización Mundial de la Salud).

Como solución a este problema y dependiendo de cada persona, se han desarrollado dos soluciones: trasplantes para las personas candidatas a ello o bien auxiliares auditivos, dispositivos diseñados para hacer llegar el sonido de una forma más efectiva al oído, amplificándolo. La mayoría de estos dispositivos son de tipo analógico; sin embargo, debido al avance tecnológico que se ha tenido y con la necesidad de mejorar la calidad y eficiencia de estos dispositivos, se han desarrollado auxiliares auditivos con características digitales

Los auxiliares auditivos digitales ofrecen ventajas sobre los analógicos, entre ellas está que con los auxiliares digitales se pueden restringir las frecuencias que se amplifican de acuerdo a la pérdida auditiva que cada persona muestra, mientras que con los aparatos auditivos analógicos se amplifica todo lo que recibe el aparato.

Ciertamente, los avances han permitido que los auxiliares auditivos de última generación, de tres años para acá, ofrezcan mayores servicios a los usuarios en cuanto a conectividad y manejo remoto del auxiliar, pero a pesar de todo lo anterior, aún no se ha podido diseñar un sistema acústico que sea completamente eficiente y corrija varios errores que se presentan en los aparatos auditivos, por ejemplo la realimentación acústica y el ruido en los sistemas. Estos errores comunes hacen que la calidad auditiva de los usuarios se vea mejorada hasta cierto punto y que haya incomodidad con los auxiliares.

Esta tesis tiene como finalidad proponer un sistema que permita reducir y/o eliminar los errores más comunes detectados en estos dispositivos para potencialmente mejorar la calidad de vida de los usuarios.

2.2 Justificación

La necesidad de mejorar las características de los auxiliares auditivos existentes, se respaldan con cifras que muestran la cantidad de personas con alguna limitación en la capacidad para escuchar en nuestro país y no solamente eso, cabe resaltar que personas que ya son usuarios de los auxiliares auditivos tienen problemas para escuchar correctamente.

De acuerdo a los datos encontrados en el INEGI en el año 2010, se registró que a lo largo del país eran 498,640 personas las que tenían alguna limitación para escuchar y donde se incluyen las personas que aún usando auxiliar auditivo tenían problemas para escuchar, lo anterior puede ser apreciado en la siguiente gráfica.

En los datos más recientes de la Encuesta Nacional de la Dinámica Demográfica en el 2014 se registró que la prevalencia de la discapacidad en México era de 6%, esto significa que 7.1 millones de habitantes del país sufren de alguna limitación [1]. Dentro de los datos

anteriores se observa que el 33.5% tiene limitación para escuchar, esto se ve traducido en que alrededor de 2.3 millones de personas, esta cifra comparada con la anterior registrada refleja un gran crecimiento poblacional de las personas que sufren esta limitación.

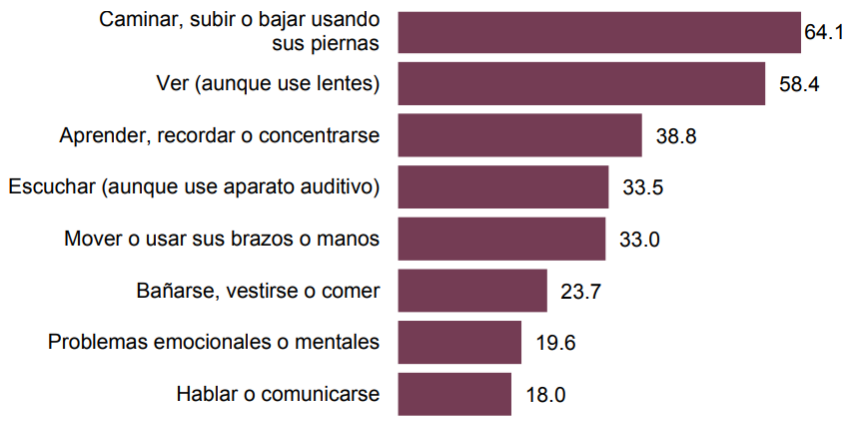


Figura 1. Gráfica de porcentaje de población con discapacidad, por tipo de discapacidad 2014. Fuente: INEGI. Encuesta Nacional de la Dinámica demográfica 2014.

La pérdida de audición tiene efectos importantes en la vida cotidiana de los individuos, ya que los problemas de comunicación pueden generar sensación de soledad, aislamiento y frustración, sobre todo en las personas mayores que padecen la pérdida de ésta. La OMS, por su parte, señala que una persona con sordera congénita podría sentirse más excluida de la vida social que el resto de la población. [1]

Los datos de la Encuesta Nacional de Hogares 2014 muestran que una de cada 10 personas de 3 años y más con discapacidad para escuchar usa algún tipo de aparato auditivo, comportamiento que es muy similar entre varones y mujeres.[1]

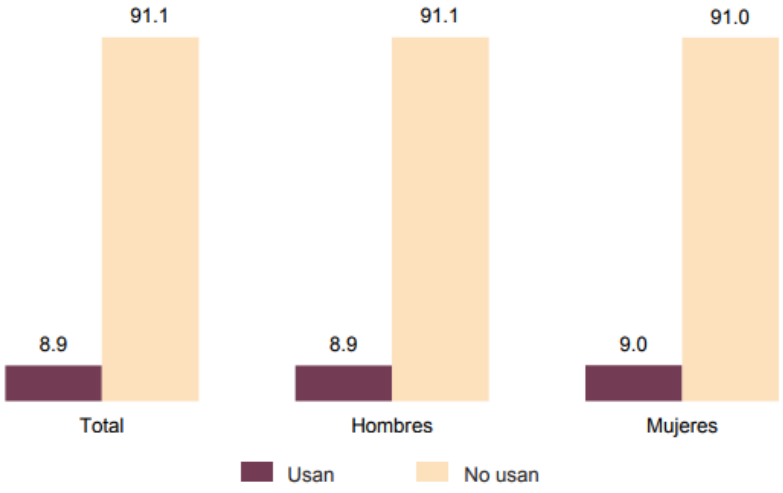


Figura 2. Gráfica de distribución porcentual de población de 3 años o más con discapacidad auditiva, por sexo según condición de uso de aparato auditivo. Fuente: INEGI. Encuesta Nacional de Hogares 2014.

Una de las principales causas por las que poca gente utiliza auxiliares auditivos es el precio, dependiendo de marcas y de la tecnología que maneja puede ir desde los 10,000 hasta los 65,000 pesos.

La experiencia que algunos usuarios han tenido con sus auxiliares auditivos no ha sido tan buena, esto se debe a que han notado ciertos problemas que no permiten una mejora significativa, entre estos problemas se encuentran: que la voz no llega con el suficiente volumen o claridad, el ruido ambiental a veces se ve tan aumentado que resulta incómodo traer el auxiliar, algunos sonidos producen que el aparato produzca un chillido demasiado agudo para el usuario, entre otras cosas.

Con el fin de solucionar los problemas antes presentados se han implementado diversas tecnologías y se han creado auxiliares auditivos con diferentes características para los diversos tipos de pérdida auditiva.

A pesar de todos los esfuerzos y las tecnologías implementadas, las fallas en los auxiliares se siguen presentando e impidiendo que la audición y calidad de vida de las personas se incrementen, por tal motivo es importante hacer hincapié en desarrollar sistemas que permitan eliminar las fallas y tomando otros caminos, no nada más quedarse con lo ya existente.

2.3 Objetivos

2.3.1 Objetivo general

Diseñar e implementar en software de simulación un sistema para el procesamiento de señales acústicas que sea capaz de detectar y eliminar el ruido ambiental y el ruido por realimentación acústica que se da en los auxiliares auditivos, mediante filtros adaptativos.

2.3.2 Objetivos específicos

a) Obtener el sistema de filtrado.
b) Comprobar la funcionalidad del sistema en simulación e identificar los parámetros para su futura implementación en tiempo real.
c) Establecer los requerimientos para la implementación en tiempo real.

2.4 Alcances

Como un primer alcance se plantea diseñar el filtro y evaluar su desempeño con ayuda de software de procesamiento y con ayuda de señales de voz reales pregrabadas.

Una vez realizado lo anterior, se propone la validación del algoritmo de filtrado en tiempo real, buscando su precisión y que cumpla los requerimientos para realizar pruebas de laboratorio y así obtener las características para un sistema que posteriormente pueda ser probado con personas.

3. Marco teórico

3.1 Auxiliares Auditivos

A lo largo de la historia se han inventado diferentes dispositivos para ayudar a las personas a escuchar mejor, el dato más antiguo habla de un tipo de cuerno hueco que permitía amplificar el sonido A partir de ahí y con la tecnología propia de cada época se han ideado otro tipo de dispositivos como tubos flexibles, cornetas, foníferos, prótesis acústicas que tiempo después fueron sustituidas por prótesis electroacústicas [2].

No fue hasta finales del siglo XIX que surgieron las primeras audioprótesis portátiles, pero no fue hasta la invención del transistor donde las prótesis auditivas tomaron llegaron a su mayor auge y se han seguido mejorando con el pasar de los años, llegando así a tener auxiliares auditivos analógicos y digitales.

Actualmente, los auxiliares auditivos digitales ofrecen ventajas importantes sobre los analógicos, entre ellas está que con los auxiliares digitales se pueden restringir las frecuencias que se amplifican de acuerdo a la pérdida auditiva que cada persona muestra, mientras que con los aparatos auditivos se amplifica todo lo que recibe el aparato.

El mejor desempeño de los auxiliares auditivos digitales sobre los analógicos se debe a que poseen ciertas características y tecnología que permiten mejorar la calidad del sonido. Las características más comunes son las siguientes:

- **Microchips**: Permiten programar el auxiliar auditivo mediante computadora, permitiendo ajustar la respuesta del auxiliar, las frecuencias que se deben amplificar y de igual manera para distinguir las señales de ruido de las señales de voz.
- **Alcance de Compresión Amplio y Dinámico:** Esta característica es la que permite ajustar la ganancia para que los sonidos suaves se amplifiquen y los sonidos fuertes no sean demasiado fuertes.[3]
- **Micrófonos direccionales:** Los auxiliares suelen tener ajustes de micrófono entre omni-direccional y direccional. Los micrófonos omni-direccionales toman los sonidos de alrededor, tienen un radio más amplio mientras que los micrófonos direccionales toman los sonidos de una dirección auditiva angosta. [3]
- **Memorias múltiples:** Esta característica se refiere a las diferentes configuraciones de audiciones que un auxiliar auditivo puede tener guardado en la memoria. Estas configuraciones permiten al usuario escoger entre ajustes del auxiliar dependiendo del ambiente en el que se encuentre, por ejemplo en situaciones ruidosas o cuando se escuche televisión o música. [3]
- **Sistemas de frecuencia modulada:** Consiste en que una persona tenga un micrófono y un transmisor de radio mientras que la persona que utiliza auxiliares auditivos los tiene conectados a los recibidores del radio. El sonido es mandado directamente entre ambas personas usando señales de radio inalámbricas. Los sistemas de frecuencia modulada mejoran la audición cuando el aparato por sí mismo no ayuda lo suficiente. [3]

Y con el desarrollo tecnológico de los últimos años se han ido obteniendo más características lo que permite que actualmente existan aparatos auditivos que se conectan a algunas aplicaciones de un celular que permiten controlar el volumen, que el timbre del

teléfono suene directamente en el oído, así como la música que se reproduce en el celular, y se puedan escuchar las instrucciones del Google Maps. El único defecto de estos aparatos es que están unidos entre sí por una diadema y en ocasiones su funcionamiento no es totalmente satisfactorio, debido a que no aíslan por completo el ruido. Por eso los nuevos aparatos auditivos llamados inteligentes, no necesitan estar conectados entre sí ni a un control, están directamente conectados vía bluetooth con un celular y desde ahí se pueden controlar totalmente. A estos aparatos se les puede ajustar el control de volumen de acuerdo al lugar donde se encuentre el usuario. La desventaja es que estos nuevos aparatos de audición aún no han sido distribuidos en todo el mundo, además de ello el precio al que sean vendidos en las diferentes partes del mundo también afectan a su accesibilidad.

A pesar de toda la tecnología implementada en los auxiliares auditivos más recientes, hay problemas constantes en su desempeño por los cuales los usuarios no sienten que su auxiliar les ofrezca una gran ventaja sobre su pérdida auditiva. Los principales problemas que refieren los usuarios de los auxiliares auditivos son:

- ➔ Falta de claridad en la voz al tener una conversación y poco volumen de la voz en ambientes ruidosos
- ➔ Sonidos fuertes llegan a ser muy molestos
- ➔ Se producen chillidos del auxiliar auditivo debido a que el sonido ya procesado y amplificado para llegar al oído del usuario, puede desviarse y volver al micrófono que capta el sonido antes de ser procesado, a lo anterior se le conoce como realimentación acústica.

3.2 Acondicionamiento

3.2.1 Tratamiento digital de señales

Una señal se define como cualquier magnitud física que varía con el tiempo, el espacio o cualquier otra variable y se describe matemáticamente como una función de una o más variables independientes. La generación de una señal está asociada a un sistema que responde a algún estímulo o fuerza [4].

Un sistema también se puede definir como un dispositivo físico o la implementación en software que permite realizar una o diversas operaciones sobre una señal, por ejemplo un filtro.

Se le conoce como tratamiento o procesamiento de señales a la acción de pasar una señal a través de un sistema, las operaciones que sean realizadas sobre la señal es el tratamiento; si lo vemos de esa manera, el tratamiento digital de señales se refiere a las operaciones que se efectúan sobre una señal mediante un software de computadora.

La mayoría de las señales con las que se trabaja en la ingeniería son analógicas, es decir, son señales son funciones de una variable continua, generalmente el tiempo. Si bien las señales analógicas pueden procesarse mediante sistemas analógicos (circuitos lógicos), el tratamiento digital proporciona un método alternativo para el procesamiento.

Para realizar un tratamiento digital, es necesario disponer de una interfaz entre la señal analógica y el procesador digital. Esta interfaz se denomina convertidor analógico digital, que entrega una señal digital adecuada para el procesador.

El procesador digital puede ser una computadora o un microprocesador programado para realizar las operaciones sobre la señal deseada, este tipo de procesadores permite cambiar las operaciones del procesamiento mediante una modificación de software. Sin embargo, si las operaciones a realizar son muy específicas, una implementación de un procesador digital cableado resultaría de lo más conveniente.

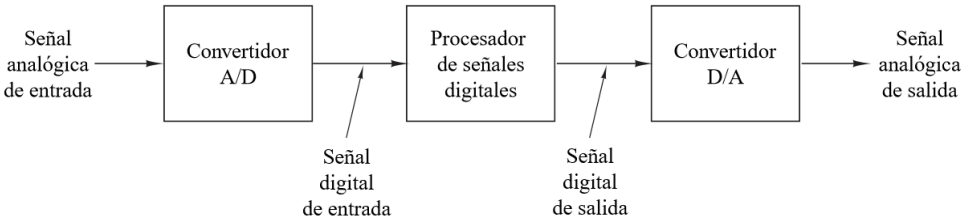


Figura 3. Diagrama de bloques de un sistema de tratamiento digital de señales. [4]

Los métodos que se utilicen para procesar una señal dependerán por completo de los atributos de la señal a procesar; en consecuencia, primero se deberá clasificar la señal implicada.

Antes que nada, es importante aclarar que la función por la que esté dada una señal puede tener magnitud real o magnitud compleja. Una vez aclarado lo anterior, la primera clasificación de las señales está dada por el número de variables independientes de las cuales la señal sea función; si la señal es función de una sola variable, se dice que la señal es unidimensional. Por otro lado, se dice que la señal es M-dimensional si su valor está en función de M variables independientes.

Como la mayoría de las señales están en el dominio del tiempo nos centraremos en la clasificación dentro de este dominio. Las señales en el dominio del tiempo pueden ser continuas y discretas. Las señales continuas en el tiempo o señales analógicas están definidas para cada instante de tiempo y toman sus valores en un intervalo continuo que puede ir desde $-\infty$ hasta ∞. Las señales discretas en el tiempo solo están determinadas en instantes específicos de tiempo, esos instantes no necesariamente deben ser equidistantes. En la práctica las señales discretas en el tiempo pueden originarse de dos formas:

a) Seleccionando valores de una señal analógica en instantes discretos de tiempo, a lo cual se le denomina muestreo.
b) Acumulando una variable en un período de tiempo, por ejemplo el número de personas que pasan por una esquina determinada en una hora.

Los valores de una señal continua o discreta en el dominio del tiempo pueden ser continuos o discretos. Si una señal toma todos los valores posibles en un rango finito o infinito, se dice que es una señal continua. Si la señal toma valores dentro de un conjunto finito de posibles valores, se dice que la señal es discreta. Una señal discreta en el tiempo que tiene un conjunto de valores discretos es una señal digital.

Para poder procesar una señal, esta debe ser necesariamente digital. Si la idea es procesar una señal analógica debe convertirse en una señal digital muestreándola en instantes discretos de tiempo y cuantificando sus valores en un conjunto de valores discretos.

El procesamiento de señales requiere disponer de una descripción matemática para la señal, esta descripción lleva a otra clasificación de las señales: señales deterministas y aleatorias. Las señales deterministas son aquellas que se pueden describir unívocamente mediante una expresión matemática, una tabla de datos o una regla bien definida. Por el contrario, las señales aleatorias no se pueden describir con precisión lo que implica que dichas señales evolucionan con el tiempo de manera no predecible.

Otro concepto clave para el procesamiento de señales es la frecuencia, la cual está relacionada con un tipo de movimiento periódico, descrito mediante funciones sinusoidales. De igual manera el concepto de frecuencia está relacionado con el concepto de tiempo, y su dimensión es la inversa de la del tiempo. Por lo tanto, la naturaleza del tiempo (continuo o discreto) afectará la naturaleza de la frecuencia.

El tema anterior fue revisado en el Capítulo 1 del libro *Tratamiento digital de señales,* presentado en la referencia [4] de la presente tesis.

3.2.2 Filtrado

Los filtros son dispositivos o sistemas que permiten separar señales que están entremezcladas o distorsionadas por ruido presente en ellas, con el fin de obtener señales puras o aisladas para utilizarlas. Estos sistemas tienen una entrada, la señal sin filtrar, y una salida, la señal filtrada; los filtros funcionan en el dominio de la frecuencia y se centra en discriminar señales dependiendo de su frecuencia, es decir, están diseñados para rechazar o permitir pasar señales dentro de un intervalo de frecuencias. Al rango de frecuencias que pasan se le conoce como banda de paso y el rango de frecuencias que son bloqueadas, banda de rechazo.

Dentro el filtrado existen dos grandes categorías que dependen del tipo de señal que se trate, estas categorías son: filtros analógicos que son para tratamiento de señales continuas en el tiempo, y los filtros digitales que trabajan con señales discretas en el tiempo.

3.2.3 Tipos de filtros

Existen cuatro tipos de filtros generales y se clasifican de acuerdo a las frecuencias que discriminan, a continuación se describen los distintos tipos:

- **Filtro pasabajas**: Permite basar bajas frecuencias en su banda de paso y rechazar frecuencias altas, sus bandas quedan establecidas por:

  Banda de paso: de 0 hasta $\boldsymbol{\omega}_p$

  Banda de rechazo: de $\boldsymbol{\omega}_s$ hasta infinito

  $\boldsymbol{\omega}_p$: frecuencia de paso, $\boldsymbol{\omega}_s$: frecuencia de corte

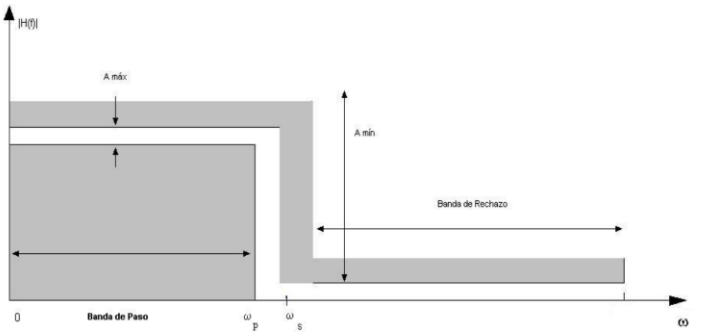


Figura 4: Respuesta en magnitud ideal para un filtro pasabajas [5].

➔ **Filtro pasoaltas:** Deja pasar frecuencias altar en su banda de paso y rechaza frecuencias bajas. Solo permite el bajo de señales arriba de su frecuencia de corte, sus bandas se describen como:

Banda de rechazo: de 0 hasta ω_p

Banda de paso: de ω_s hasta infinito

ω_p: frecuencia de paso, ω_s: frecuencia de corte.

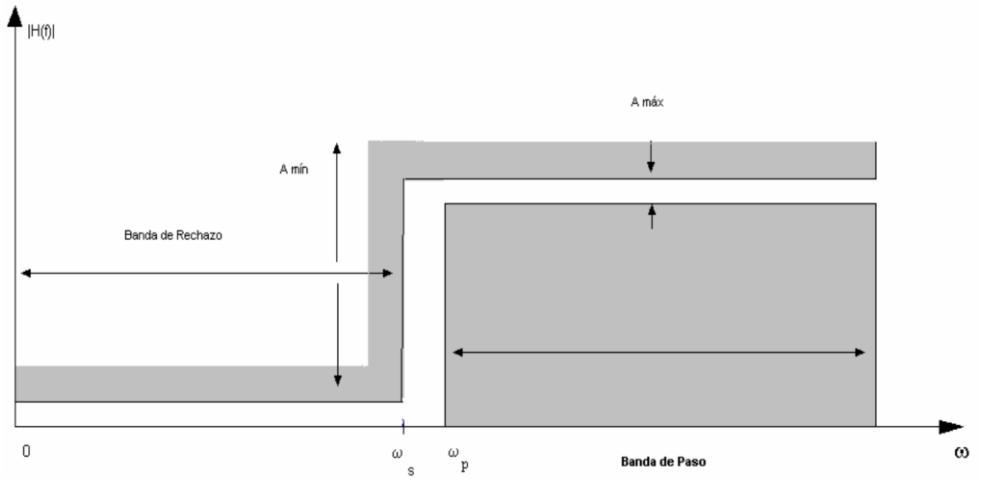


Figura 5: Respuesta en magnitud ideal de un filtro pasoaltas [5].

➔ **Filtro pasabanda**: Este tipo de filtro permite pasar un intervalo de frecuencias, por lo cual maneja dos frecuencias de corte. Permite pasar las frecuencias que se encuentra arriba de la primera frecuencia de corte y por debajo de la segunda frecuencia de corte. En este tipo de filtro hay dos bandas de rechazo, una inferior y una superior, esto queda definido por:

Banda de paso: de ω_{p1} hasta ω_{p2}

Banda de rechazo inferior: de 0 hasta ω_{s1}

Banda de rechazo superior: deω_{s2} hasta infinito

ω_{p1} y ω_{p2}: frecuencia de paso inferior y superior, respectivamente

ω_{s1} y ω_{s2}: frecuencia de corte inferior y superior, respectivamente

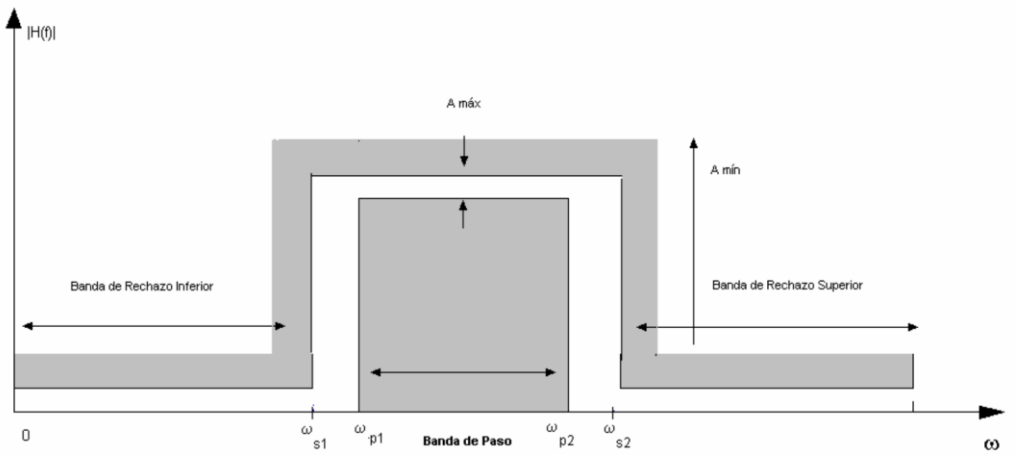


Figura 6: Respuesta en magnitud ideal de un filtro pasabanda [5].

➔ **Filtro rechazabanda:** Un filtro rechazabanda es aquel que permite el paso de frecuencias menores a una primera frecuencia de corte y las mayores a una segunda frecuencia de corte, impide de aquellas que se encuentren entre esas frecuencias de corte. Funciona bloqueando cierto rango de frecuencias, esto se resume en lo siguiente:

Banda de paso inferior: de 0 hasta ω_{p1}
Banda de rechazo: deω_{s1} hasta ω_{s2}
Banda de paso superior: deω_{p2} hasta infinito
ω_{p1} y ω_{p2}: frecuencia de paso inferior y superior, respectivamente
ω_{s1} y ω_{s2}: frecuencia de corte inferior y superior, respectivamente

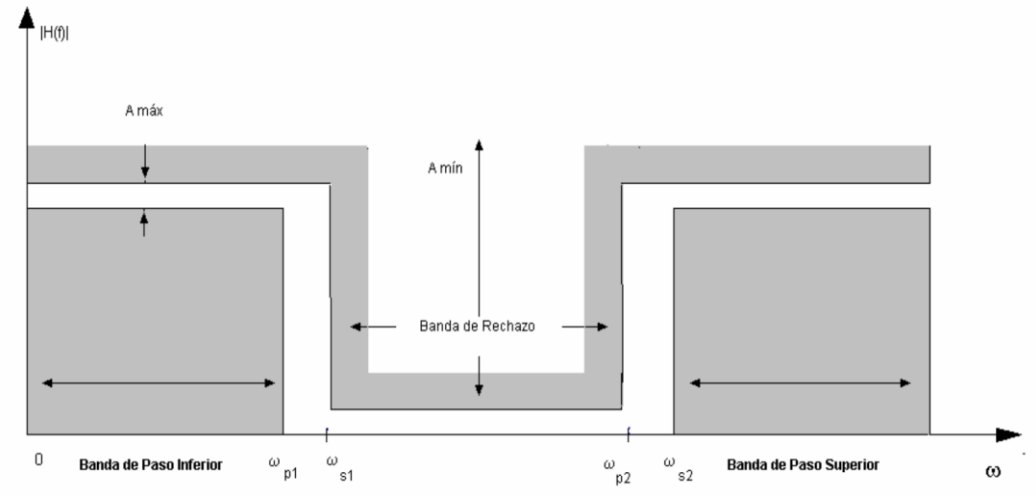


Figura 7: Respuesta de magnitud ideal de un filtro rechazabanda [5].

3.2.4 Conversión A/D y D/A

El desarrollo de este tema se realizó revisando el Capítulo 1 en la sección 1..4 del libro *Tratamiento Digital de Señales,* referencia [4] de la presente tesis.

Como ya se había dicho anteriormente, la mayoría de las señales que se encuentran en el mundo real por naturaleza con analógicas y para procesarlas digitalmente es necesario convertirlas a formato digital. Al anterior procedimiento se le conoce como *conversión analógica-digital (A/D),* y los dispositivos que las realizan son los convertidores A/D (ADC).

Básicamente, la conversión A/D es un proceso de tres pasos:

1) Muestreo: Consiste en convertir la señal continua en el tiempo en una discreta en el tiempo obteniendo muestras de la señal continua en instantes discretos en el tiempo, es decir, intervalos de tiempo definidos.
2) Cuantificación: En este paso el valor de cada muestra de la señal se presenta como un valor dentro de un conjunto finito de posibles valores. La diferencia entre la muestra no cuantificada y la salida cuantificada es el error de cuantificación.
3) Codificación: Cada valor discreto se representa con una secuencia binario de bits.

Para entender bien algunos de los conceptos clave dentro de la conversión A/D, se explicará con detenimiento cada paso del proceso.

El muestreo de una señal analógica se puede realizar de diversas maneras, pero el método más utilizado es el muestreo uniforme o periódico y se describe con la siguiente relación:

$$x(n) = x_a(nT), \qquad -\infty < n < \infty \tag{3.2.4.1}$$

donde x(n) es la señal discreta en el tiempo al tomar muestras de la señal anlógica $x_a(t)$ cada T segundos. El intervalo de tiempo T entre muestras sucesivas es el período de muestreo y su recíproco $1/T=F_S$ se denomina frecuencia de muestreo.

En el muestreo periódico las variables t y n se relacionan linealmente a través del periodo de muestreo, como

$$t = nT = \frac{n}{Fs} \tag{3.2.4.2}$$

consecuencia de la relación anterior, también existe una entre la frecuencia F para las señales analógicas y la frecuencia *f* para las señales discretas. Para establecer la relación, se considera una señal analógica sinusoidal:

$$x_a(t) = A\cos(2\pi F t + \theta) \tag{3.2.4.3}$$

Al muestrear la señal anterior con una frecuencia Fs muestras por segundo, da lugar a:

$$x_a(nT) = x(n) = A\cos(2\pi F nT + \theta) = A\cos(\frac{2\pi nF}{Fs} + \theta) = A\cos(2\pi f + \theta) \tag{3.2.4.4}$$

y se puede observar que la relación entre F y f es la siguiente:

$$f = \frac{F}{Fs} \tag{3.2.4.5}$$

Tabla 1. Relaciones entre las variables de frecuencia [4].

Señales continuas en el tiempo		Señales discretas en el tiempo
$\Omega = 2\pi F$		$\omega = 2\pi f$
$\frac{\text{radianes}}{\text{segundo}}$ Hz		$\frac{\text{radianes}}{\text{muestra}}$ $\frac{\text{ciclos}}{\text{muestra}}$
	$\omega = \Omega T, f = F/F_s$	
	$\Omega = \omega / T, F = f \cdot F_s$	$-\pi \le \omega \le \pi$
		$-\frac{1}{2} \le f \le \frac{1}{2}$
$-\infty < \Omega < \infty$		$-\pi/T \le \Omega \le \pi/T$
$-\infty < F < \infty$		$-F_2/2 \le F \le F_s/2$

Para seleccionar la Fs tenemos que tener información del contenido en frecuencia de la señal a muestrear, si las frecuencias de la señal no exceden una determinada frecuencia conocida, se le denominará F_{max}.

La frecuencia más alta de una señal analógica que puede reconstruirse sin ambigüedades es cuando se muestrea la señal a una frecuencia Fs=1/T es Fs/2. Cualquier frecuencia encima de Fs/2 o menor a -Fs/2 produce muestras idénticas a las correspondientes frecuencias dentro del rango -Fs/2≤ F ≤ Fs/2, lo que es conocido como *aliasing.* Para evitar el fenómeno anterior se selecciona Fs de modo que

$$Fs > 2Fmax \tag{3.2.4.6}$$

A la frecuencia de muestreo anterior también se le conoce como *frecuencia de Nyquist,* que viene el teorema de muestreo que establece que una señal analógica podrá muestrearse sin aliasing se cumple la regla de tomar la frecuencia de muestreo de la siguiente manera:

$$Fs = F_N = 2\,Fmax \tag{3.2.4.7}$$

Sin embargo, en la práctica se recomienda que:

$$F_N > 2\,Fmax \tag{3.2.4.8}$$

La operación de cuantificación de las muestras se denota como Q[x(n)] y se emplea xq(n) para indicar la secuencia de las muestras cuantificadas a la salida. Por tanto,

$$x_q(n) = Q[x(n)] \tag{3.2.4.9}$$

El error de cuantificación es una secuencia eq(n) definida como:

$$e_q(n) = x_q(n) - x(n) \tag{3.2.4.10}$$

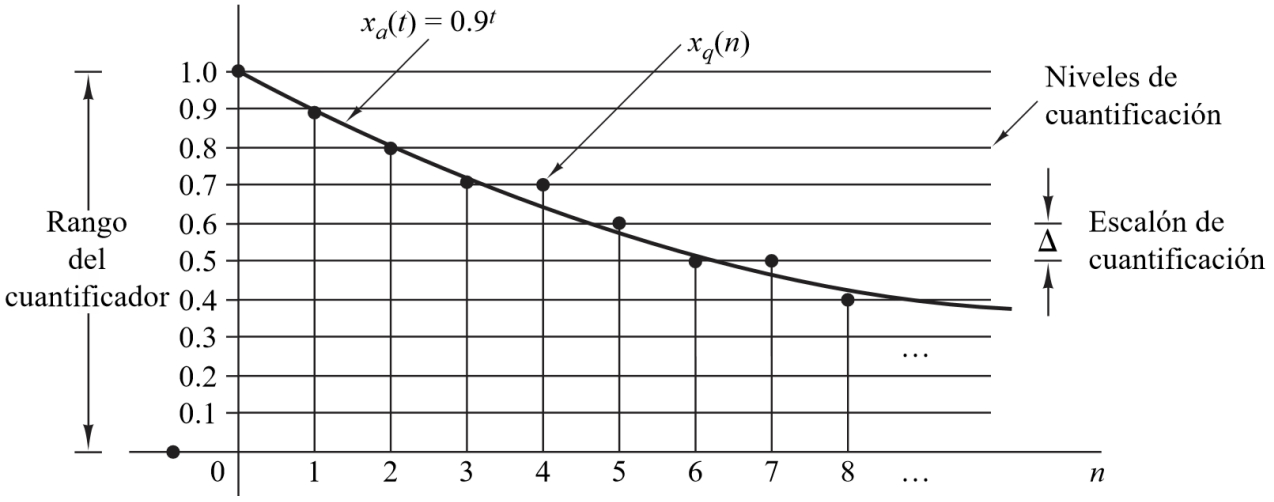


Figura 8. Ilustración de la cuantificación [4].

De la imagen anterior definimos que los valores permitidos en la señal digital son los *niveles de cuantificación,* la distancia *Δ* entre dos niveles de cuantificación define *el tamaño del escalón de cuantificación o resolución.* Existen los cuantificadores por redondeo y por truncamiento, por redondeo se asigna cada muestra de x(n) al nivel de cuantificación más próximo y por truncamiento, al nivel de cuantificación inmediatamente inferior. En la cuantificación por redondeo se tiene que el error de cuantificación instantáneo no puede ser mayor que la mitad del escalón de cuantificación, esto quiere decir que:

$$-\frac{\Delta}{2} \leq e_q(n) \leq \frac{\Delta}{2} \tag{3.2.4.11}$$

La codificación de muestras cuantificadas asigna un número binario unívoco a cada nivel de cuantificación. Si se tienen L niveles se necesitan al menos L números binarios distintos.

Para convertir una señal digital a una analógica se realiza la *conversión digital-analógica (D/A)* . Los dispositivos que realizan esta operación se llaman convertidores D/A y funcionan conectando los puntos de una digital realizando interpolación. Existen diferentes aproximaciones para la conversión D/A, por ejemplo: aproximación mediante escalones, la conexión lineal de una pareja de muestras sucesivas (interpolación lineal), el ajuste de una función cuadrática a través de tres muestras sucesivas (interpolación cuadrática), etc.

En general, las técnicas de interpolación dejan pasar frecuencias por encima de la frecuencia de solapamiento, aquella en que se presenta aliasing, Tales frecuencias no deseadas normalmente se eliminan haciendo pasar la salida de la interpolación por un filtro analógico, al que se le conoce como *post-filtro o filtro de suavizado.*

3.3 Filtrado Digital

La información presentada a continuación se revisó en el Capítulo 10 del libro *Tratamiento Digital de Señales,* referencia [4] de la presente tesis.

Los filtros digitales son sistemas que procesan señales discretas en el tiempo, por ende son sistemas discretos en el tiempo que realizan una operación sobre la señal de entrada *x(n)* para generar la señal de salida *y(n).*

Para conocer el tipo de filtro, se requiere conocer el tipo de sistema y para esto existe una clasificación basada en las propiedades del mismo. A continuación se presenta dicha clasificación:

- **Sistemas estáticos y dinámicos:** Un sistema discreto en el tipo es *estático* si su salida en cualquier instante *n* depende de la muestra de entrada en ese mismo instante, no de muestras pasadas o futuras en la entrada. En cualquier otro caso, el sistema es *dinámico*.

- **Sistemas invariantes y variantes en el tiempo:** Un sistema es invariante en el tiempo si su característica de entrada-salida no cambia con el tiempo, en caso contrario el sistema es variante en el tiempo. Para comprobar si un sistema es invariante en el tiempo se debe excitar al sistema con *x(n)* para generar la salida *y(n)*, a continuación se retarda la secuencia obtenida en cierta cantidad k de tal manera que se cumpla lo siguiente:

$$\begin{aligned} x(n) &\rightarrow y(n) \\ x(n-k) &\rightarrow y(n-k) \end{aligned} \tag{3.3.1}$$

  Si la salida $y(n,k) \neq y(n-k)$es variante en el tiempo.

- **Sistemas lineales y no lineales:** Un sistema lineal es aquel que satisface el *principio de superposición*, que establece que la respuesta del sistema a una suma de señales sea

igual a la suma de las respuestas o salidas del sistema a cada una de las señales individuales de entrada, lo anterior se puede resumir en la siguiente ecuación:

$$\Im[a_1x_1(n) + a_2x_2(n)] = a_1\Im[x_1(n)] + a_2\Im[x_2(n)] = a_1y_1(n) + a_2 \quad (3.3.2)$$

donde $\Im$ representa la operación que se le aplica a la señal discreta.

- **Sistemas causales y no causales:** Un sistema es *causal* si su salida en cualquier instante *n* sólo depende de las entradas actuales y pasadas pero no de las futuras. Si el sistema depende también de las entradas futuras, se dice que es *no causal*.

- **Sistemas estables e inestables:** Un sistema en reposo es estable si y sólo si toda entrada acotada genera una salida acotada (no infinita), en caso contrario es inestable.

Un sistema FIR (*Finite-duration impulse respons*e) es un sistema lineal invariante en el tiempo que tiene un respuesta el impulso que es cero fuera de un determinado intervalo finito, en otras palabras, el sistema actúa como una ventana que solo toma las muestras de la señal de entrada más recientes para formar la salida.

Contrario a los sistemas FIR, los sistemas IIR (*Infinite-duration impulse response*) tiene un respuesta al impulso de duración infinita, lo que quiere decir que el sistema toma todas muestras de entrada actual y todas las pasadas.

Los sistemas discretos en el tiempo se pueden representar mediante ecuaciones en diferencias, en ellas se relacionan los valores pasados de un señal y sus valores de entrada actuales y futuros.

3.3.1 Filtros FIR

Existen varios métodos para diseñar filtros FIR, pero una característica de ellos es que son de fase lineal por lo cual no se produce distorsión en la fase y para su diseño es necesario considerar que la señal o sistema a filtrar debe de ser causal.

Un filtro FIR de longitud M con entrada *x(n)* y salida *y(n)* se describe mediante la ecuación en diferencias:

$$y(n) = b_0x(n) + b_1x(n-1) + \ldots + b_{M-1}x(n-M+1) = \sum_{k=0}^{M-1} b_kx(n-k) \quad (3.3.1.1)$$

donde $\{b_k\}$ es el conjunto de coeficientes del filtro. Por otra parte podemos expresar la salida del filtro *y(n)* como una convolución de la entrada *x(n)* con la respuesta a impulso del filtro *h(n)*:

$$y(n) = \sum_{k=0}^{M-1} h(k) \cdot x(n-k) \quad (3.3.1.2)$$

Ya que las dos ecuaciones anteriores son idénticas, se puede demostrar que los coeficientes *bk=h(k)*. Así mismo, se puede demostrar que la respuesta de un filtro FIR es de fase lineal si los coeficientes *h(n)* cumplen:

$$h(n) = \pm h(M-1-n), \quad n+0,1,\ldots,M-1 \quad (3.3.1.3)$$

Es decir, los coeficientes tienen algún tipo de simetría. Al llevar lo anterior a la transformada Z se encuentra que las raíces ocurren en pares recíprocas, es decir una raíz y su recíproco serán ceros y de igual manera con las raíces complejas. A lo anterior, se le llama que es un filtro FIR simétrico, en caso contrario el filtro es antisimétrico.

El primer método de diseño para los filtros FIR de fase lineal es con el *uso de ventanas*, en este método a partir de la respuesta en frecuencia deseada $H_d(\omega)$ se determina la correspondiente respuesta el impulso unitario $h_d(n)$. La respuesta al impulso unitario está relacionada con la respuesta en frecuencia deseada por la transformada de Fourier

$$H_d(\omega) = \sum_{n=0}^{\infty} h_d(n) e^{-j\omega n} \tag{3.3.1.4}$$

$$h_d(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(\omega) e^{j\omega n} d\omega \tag{3.3.1.5}$$

Por lo tanto, dada $H_d(\omega)$ se puede determinar la respuesta al impulso unitario evaluando la integral anterior. Generalmente, la respuesta al impulso obtenida es infinita en duración y tiene que truncarse en algún punto, por ejemplo $n=M-1$, para obtener un filtro FIR de longitud M. Truncar $h_d(n)$ a un longitud $M-1$ es equivalente a multiplicar $h_d(n)$ por una "ventana rectangular", definida como

$$w(n) = \begin{cases} 1, & n = 0, 1, \ldots, M-1 \\ 0, & \text{en otro caso} \end{cases} \tag{3.3.1.6}$$

Y la respuesta al impulso unitario queda definida como

$$h(n) = h_d(n) w(n) = \begin{cases} h_d(n), & n = 0, 1, \ldots, M-1 \\ 0, & \text{en otro caso} \end{cases} \tag{3.3.1.7}$$

Es necesario considerar lo que se presenta como una multiplicación en el dominio de la frecuencia representa una convolución de $H_d(\omega)$ con $W(\omega)$, donde $W(\omega)$ es la representación en el dominio de la frecuencia de la función de la ventana, es decir,

$$W(\omega) = \sum_{n=0}^{M-1} w(n) e^{-j\omega n} \tag{3.3.1.8}$$

Luego la convolución de $H_d(\omega)$ con $W(\omega)$ proporciona la respuesta en frecuencia del filtro FIR truncado.

Las características de la ventana rectangular desempeñan un papel importante en la determinación de la respuesta en frecuencia resultante del filtro FIR obtenido por truncamiento de $h_d(n)$ a un longitud M, la convolución en el dominio de la frecuencia tiene el efecto de suavizar $H_d(\omega)$. Cuando M aumenta, $W(\omega)$ se hace más estrecha y el suavizado se reduce.

Existen funciones de ventana establecidos que nos ayudan al diseño pero depende de las características de nuestro sistema la ventana a utilizar.

Tabla 2. Funciones de ventana para el diseño de filtro FIR [4]

Nombre de la ventana	Secuencia en el dominio del tiempo, $h(n), 0 \le n \le M-1$
Bartlett (triangular)	$1 - \frac{2\left\lvert n - \frac{M-1}{2}\right\rvert}{M-1}$
Blackman	$0.42 - 0.5\cos\frac{2\pi n}{M-1} + 0.08\cos\frac{4\pi n}{M-1}$
Hamming	$0.54 - 0.46\cos\frac{2\pi n}{M-1}$
Hanning	$\frac{1}{2}\left(1 - \cos\frac{2\pi n}{M-1}\right)$
Kaiser	$\frac{I_0\left[\alpha\sqrt{\left(\frac{M-1}{2}\right)^2 - \left(n - \frac{M-1}{2}\right)^2}\right]}{I_0\left[\alpha\left(\frac{M-1}{2}\right)\right]}$
Lanczos	$\left\{\frac{\text{sen}\left[2\pi\left(n - \frac{M-1}{2}\right)\Big/(M-1)\right]}{2\pi\left(n - \frac{M-1}{2}\right)\Big/\left(\frac{M-1}{2}\right)}\right\}^L, \; L > 0$
	$1, \left\lvert n - \frac{M-1}{2}\right\rvert \le \alpha\frac{M-1}{2}, \quad 0 < \alpha < 1$
Tukey	$\frac{1}{2}\left[1 + \cos\left(\frac{n - (1+a)(M-1)/2}{(1-\alpha)(M-1)/2}\pi\right)\right]$
	$\alpha(M-1)/2 \le \left\lvert n - \frac{M-1}{2}\right\rvert \le \frac{M-1}{2}$

Las ventanas de la tabla anterior proporcionan más suavizado a través de la operación de convolución en el dominio de la frecuencia y, como resultado, la región de transición de la respuesta del filtro FIR es más ancha. Para reducir la anchura de la región de transición, simplemente se aumenta la longitud de la ventana.

Tabla 3. Características en el dominio de la frecuencia de algunas funciones de ventana.

Tipo de ventana	Anchura aproximada de la región de transición del lóbulo principal	Pico del lóbulo secundario (dB)
Rectangular	$4\pi/M$	−13
Bartlett	$8\pi/M$	−25
Hanning	$8\pi/M$	−31
Hamming	$8\pi/M$	−41
Blackman	$12\pi/M$	−57

El segundo método para el diseño de filtros FIR de fase lineal es el basado en *muestreo en frecuencia,* en este método se especifica la respuesta en frecuencia deseada $H_d(\omega)$ en un conjunto de frecuencias equiespaciadas, es decir,

$$\omega_k = \tfrac{2\pi}{M}(k+\alpha), \quad k = 0, 1, \ldots, \tfrac{M-1}{2}, \quad M \text{ impar}$$
$$k = 0, 1, \ldots, \tfrac{M}{2} - 1, \quad M \text{ par}$$
$$\alpha = 0 \text{ ó } \tfrac{1}{2} \qquad (3.3.1.9)$$

y se resuelve para obtener la respuesta al impulso unitario *h(n)* del filtro FIR a partir de la especificación de frecuencia.

La respuesta en frecuencia deseada del filtro es

$$H_d(\omega) = \sum_{n=0}^{M-1} h_d(n) e^{-j\omega n} \tag{3.3.1.10}$$

Suponiendo que se especifica la respuesta en frecuencia del filtro en las frecuencias dadas por ω_k, se obtiene

$$H(k+\alpha) \equiv H\left(\frac{2\pi}{M}(k+\alpha)\right)$$

$$H(k+\alpha) \equiv \sum_{n=0}^{M-1} h(n) e^{-j2\pi(k+\alpha)n/M}, \; k=0,1,\ldots, M-1 \tag{3.3.1.11}$$

La expresión anterior se invierte para expresar *h(n)* en función de *H (*k + α). Si se multiplican ambos lados de la ecuación por la exponencial (j2km/M), *m= 0,1,...M-1*, y se suma para *k=0,1,...,M-1,* se puede obtener que

$$h(n) = \frac{1}{M} \sum_{n=0}^{M-1} H(k+\alpha) e^{j2\pi(k+\alpha)n/M}, \qquad n=0,1,\ldots,M-1 \tag{3.3.1.12}$$

La relación anterior permite calcular los valores de la respuesta al impulso unitario a partir de las especificaciones de las muestras en frecuencia *H (k+α*).

Existe un problema en la realización basada en el muestreo en frecuencia de filtros FIR de fase lineal, en la implementación práctica los efectos de cuantificación impiden una cancelación perfecta de los polos y los ceros, no se proporciona un amortiguamiento del ruido debido al redondeo de los cálculos. Como resultado, el ruido tiende a aumentar con el tiempo pudiendo destruir el funcionamiento normal del filtro.

El tercer método de diseño de filtros FIR de fase lineal es el de *rizado constante óptimo,* este método se formula como problema de la aproximación de Chebyshev, puede verse como un criterio de diseño óptimo ya que el error de la aproximación ponderado entre la respuesta en frecuencia deseada y la respuesta en frecuencia real se dispersa a lo largo de la banda de paso e igualmente a lo largo de la banda eliminada del filtro, minimizando el error máximo. El filtro resultante presenta rizados tanto en la banda de paso como en la banda eliminada.

La respuesta en frecuencia de un filtro cuya respuesta al impulso h(n) es una secuencia simétrica puede ponerse como

$$H(\omega) = \sum_{k=0}^{L} \alpha\,(k) cos(\omega k) \tag{3.3.1.13}$$

Esta forma es un polinomio de Chebyshev y se debe escoger α para que el diseño sea óptimo, el teorema de alternancia ofrece una pista para seleccionar α.

El teorema de alternancia establece que sea S un subconjunto del intervalo [0, π}. Una condición necesaria y suficiente para que

$$H(\omega) = \sum_{k=0}^{L} \alpha\,(k) cos(\omega k) \tag{3.3.1.14}$$

sea la mejor aproximación de Chebyshev para la respuesta en frecuencia deseada en S y el error ponderado en la aproximación se definido como

$$E(\omega) = W(\omega)[H_d(\omega) - H(\omega)] \tag{3.3.1.15}$$

presente al menos L+2 frecuencias extremas en S. Es decir, deben existir al menos L+2 frecuencias {ωi} en S tal que $\omega 1<\omega 2<\ldots<\omega_{L+2}$, $E(\omega i)=-E(\omega_{i+1})$, y

$$\left|E(\omega_i)\right| = máx\,|E(\omega)|\,, \quad i=1,2,\ldots,L+2 \tag{3.3.1.16}$$

Hay disponible un programa software escrito por Parks y McClellan para diseñar filtros FIR de fase lineal basado en los criterios de la aproximación de Chebyshev e implementado mediante el algoritmo de intercambio de Remez, un algoritmo iterativo para obtener los parámetros de frecuencias extremas y errores. Este programa puede emplearse para diseñar filtros paso bajo, paso alto o paso banda. El programa de Parks–McClellan requiere una serie de parámetros de entrada que determinen las características del filtro. En particular, deben especificarse los siguientes parámetros:

-NFILT: la longitud del filtro, designada anteriormente por M.
-JTYPE: el tipo de filtro : JTYPE = 1 da lugar a un filtro de banda eliminada/paso banda múltiple. JTYPE = 2 da lugar a un diferenciador. JTYPE = 3 da lugar a un transformador de Hilbert.
-NBANDS: el número de bandas de frecuencia desde 2 (para un filtro paso bajo) a un máximo de 10 (para un filtro de múltiples bandas).
-LGRID: la densidad de la cuadrícula para interpolar la función de error E(ω). El valor predeterminado es 16, si no se especifica.
-EDGE: las bandas de frecuencia especificadas por las frecuencias de corte superior, hasta un máximo de 10 bandas (una matriz de tamaño 20 como máximo). Las frecuencias se proporcionan en función de la variable f = ω /2 π, donde f = 0.5 corresponde a la frecuencia de solapamiento.
- FX: una matriz de tamaño máximo 10 que especifique la respuesta en frecuencia deseada Hd(ω) en cada banda.
- WTX: una matriz de tamaño máximo 10 que especifique la función de ponderación en cada banda.

3.3.2 Filtros IIR

Al igual que para los filtros FIR, existen varios métodos para diseñar filtros digitales IIR y todas ellas se basan en la conversión de un filtro analógico en un filtro digital.

Un filtro analógico puede describirse mediante su función de sistema,

$$Ha(s) = \frac{B(s)}{A(s)} = \frac{\sum_{k=0}^{M} \beta_k s^k}{\sum_{k=0}^{N} \alpha_k s^k} \tag{3.3.2.1}$$

donde {α_k} y {β_k} son los coeficientes del filtro. Alternativamente, el filtro analógico cuya función de sistema relacional es H(s) puede describirse mediante la ecuación en diferencias

$$\sum_{k=0}^{N} \alpha_k \frac{d^k y(t)}{dt^k} = \sum_{k=0}^{M} \beta_k \frac{d^k x(t)}{dt^k} \tag{3.3.2.2}$$

donde x(t) designa la señal de entrada e y(t) designa la salida del filtro.

El sistema analógico lineal e invariante en el tiempo H(s) es estable si todos sus polos se encuentran en la mitad izquierda del plano s. En consecuencia, para que la técnica de conversión sea efectiva, deberá poseer las siguientes propiedades:
1. El eje jΩ del plano s debe corresponderse con la circunferencia unidad en el plano z. Así, existirá una relación directa entre las dos variables de frecuencia en ambos dominios.
2. El semiplano izquierdo del plano s debe corresponderse con el interior de la circunferencia unidad en el plano z. Por tanto, un filtro analógico estable se convertirá en un filtro digital estable.

A continuación se mencionan brevemente los métodos de conversión de filtros analógicos a digitales IIR:

- Diseño de filtros IIR mediante aproximación en derivadas: Consiste en aproximar la ecuación diferencial dada por

$$\sum_{k=0}^{N} \alpha_k \frac{d^k y(t)}{dt^k} = \sum_{k=0}^{M} \beta_k \frac{d^k x(t)}{dt^k} \qquad (3.3.2.3)$$

  mediante una ecuación en diferencias equivalente. Para esto en la expresión de la derivada $dy(t)/dt$ en el instante $t=nT$, sustituimos la diferencia $[y(nT)-y(nT-1)]/T$ donde T representa el intervalo de muestreo.
- Diseño de filtros IIR basado en la invarianza del impulso: El objetivo de este método es diseñar un filtro IIR que tenga una respuesta el impulso unitario $h(n)$ que sea la versión muestreada de la respuesta al impulso del filtro analógico.
- Diseño de filtros IIR mediante la transformación bilineal: Es una correspondencia que transforma el eje $j\Omega$ en la circunferencia unidad en el plano z una vez, evitando el efecto de aliasing. Todos los puntos del semiplano izquierdo s se corresponden con el interior de la circunferencia unidad en el plano z y todos los puntos del semiplano derecho s se corresponden con los puntos externos a la circunferencia unidad en el plano z.

3.4 Filtrado adaptativo

Los filtros adaptativos son filtros con coeficientes ajustables, los cuales incorporan algoritmos que permiten cambiar los coeficientes del filtro a los parámetros estadísticos de la señal. Una consideración en el uso de un filtro adaptativo es el criterio de optimizar los parámetros ajustables del filtro. El criterio no solo debe proporcionar una medida significativa del rendimiento del filtro, sino también debe dar un algoritmo que sea realizable en la práctica.

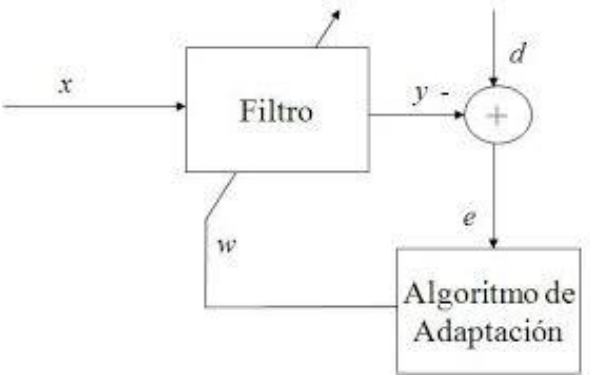


Figura 9. Modelo general del filtrado adaptativo [21]

Como se aprecia en la imagen anterior en el filtrado adaptativo se ingresa una señal x (señal con ruido) al filtro, la señal que se quiere filtrar; a la salida del filtro de obtiene una señal y, que es la estimada obtenida después del filtro y se resta a la señal d (señal deseada) que sirve de referencia para el sistema, de la resta anterior se obtiene el error que entra al algoritmo de adaptación que es donde se evalúa el valor del error y qué tanto se acerca a un valor de 0 (error óptimo) para que con dicha información se puedan recalcular los w(coeficientes del filtro) que disminuyan el valor del error para realizar la siguiente iteración y así hasta llegar al resultado óptimo.

Dos criterios que ofrecen buenas medidas del rendimiento en filtrado adaptativo son el criterio de mínimos cuadrados y el error cuadrático medio (MSE, *mean-square-error*).

3.4.1 Modelos de filtrado adaptativo

En el filtrado adaptativo se dispone de un sistema desconocido a identificar. El sistema se modela mediante un filtro FIR con *M* coeficientes ajustables. El sistema desconocido y el modelo se excitan con una secuencia de entrada *x(n).* La salida del sistema desconocido se designa con *y(n)* y $\hat{y}(y)$designa la salida del modelo,

$$\hat{y}(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \tag{3.4.1.1}$$

La secuencia de error es

$$e(n) = y(n) - \hat{y}(n), \qquad n = 0, 1, \ldots \tag{3.4.1.2}$$

y se seleccionan los coeficientes *h(k)* para minimizar

$$\varepsilon_M = \sum_{n=0}^{N} [y(n) - \sum_{k=0}^{M-1} h(k)x(n-k)]^2 \tag{3.4.1.3}$$

donde *N*+1 es el número de observaciones.

El criterio de mínimos cuadrados lleva al conjunto de ecuaciones lineales que permite determinar los coeficientes del filtro, es decir,

$$\sum_{k=0}^{M-1} h(k)r_{xx}(l-k) = r_{yx}(l), \qquad l = 0, 1, \ldots, M-1 \tag{3.4.1.4}$$

En la ecuación anterior $r_{xx}(l)$ es la autocorrelación de la secuencia de entrada y $r_{yx}(l)$ es la correlación cruzada de la salida del sistema con la secuencia de entrada. Dado que los parámetros se obtienen a partir de la medida de los datos en la entrada y la salida del sistema, sin tener conocimiento previo del sistema desconocido, decimos que el modelo del filtro FIR es un filtro adaptativo.

Dependiendo de las aplicaciones y consideraciones a tomar para el modelado de los filtros adaptativos, existen los siguientes modelos:

- **Ecualización de canal adaptativa:** Se utiliza en sistemas de comunicaciones para compensar la distorsión causada por el medio de transmisión.

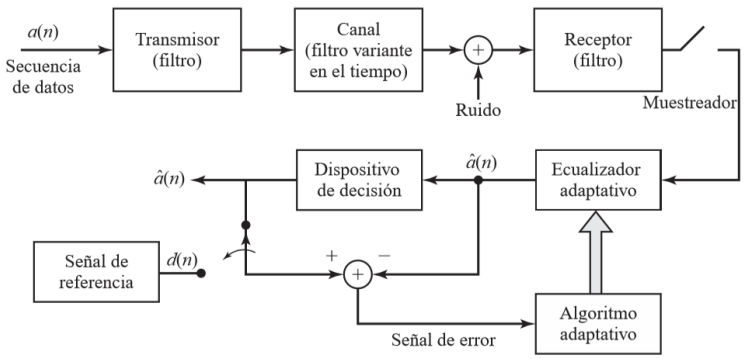


Figura 10. Aplicación de filtrado adaptativo a la ecualización de un canal. [4]

De la imagen anterior observamos que la secuencia de símbolos de información *a(n)* se aplica al filtro transmisor, cuya salida es

$$s(t) = \sum_{k=0}^{\infty} a(k)p(t - kT_s) \tag{3.4.1.5}$$

donde *p(t)* es la respuesta al impulso del filtro en el transmisor y Ts es el intervalo de tiempo entre los símbolos de información; es decir, 1/Ts es la frecuencia de símbolos.

El canal, que normalmente está modelado como un filtro lineal, distorsiona los pulsos y, por tanto, produce interferencias entre símbolos. Los filtros producen distorsión de fase y amplitud, la señal distorsionada también se ve corrompida por ruido aditivo, que normalmente es ruido de banda ancha.

En el extremo receptor del sistema de comunicaciones, la señal se pasa primero a través de un filtro que está diseñado fundamentalmente para eliminar el ruido de fuera de la banda de frecuencias ocupada por la señal. Podemos suponer que este filtro es un filtro FIR de fase lineal que limita el ancho de banda del ruido, aunque produce una distorsión adicional despreciable sobre la señal distorsionada por el canal.

Si se ignoran las posibles variaciones temporales en el canal, la salida muestreada en el receptor se puede expresar como

$$x(nTs) = \sum_{k=0}^{\infty} a(k)q(nTs - kTs) + w(nTs) = a(n)q(0) + \sum_{k=0}^{\infty} a(k)q(nTs - kTs) + w(nTs) \tag{3.4.1.6}$$

donde *w(t)* representa el ruido aditivo y *q(t)* representa el impulso distorsionada en la salida del filtro receptor.

El propósito del ecualizador adaptativo es el de compensar la señal en lo que se refiere a la distorsión del canal, de modo que la señal resultante pueda detectarse con fiabilidad.

- **Cancelación de eco en la transmisión de datos a través de canales telefónicos:** En la transmisión de datos a través de canales telefónicos se utilizan modems para proporcionar una interfaz entre la secuencia de datos digital y el canal analógico.

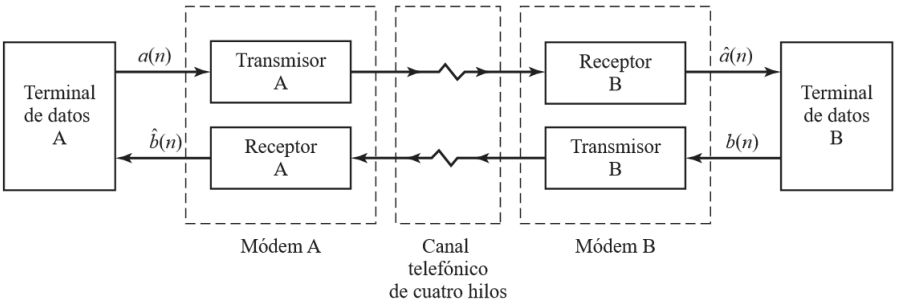


Figura 11. Transmisión de datos full-duplex sobre canales telefónicos[4].

El problema en este tipo de transmisión es el coste de alquilar un canal telefónico de cuatro hilos, en el caso de poco volumen o transmisión infrecuente de datos lo mejor es emplear una red telefónica de marcación conmutada, en este caso el enlace es una línea de dos hilos, lo que se conoce como *bucle local*. La línea de dos hilos se conecta a la de cuatro hilos, creando una línea troncal. Utilizando un acoplamiento mediante transformador, se aíslan los canales de transmisión y de recepción del *full-duplex.* A causa de la desadaptación de impedancias, el nivel de aislamiento a menudo es insuficiente y, en consecuencia, parte de la señal del lado del transmisor se acopla y distorsiona la señal del lado del receptor, produciendo "eco" que a veces puede escucharse en las comunicaciones de voz.

Para eliminar los ecos en las transmisiones, las compañías utilizan *supresores de eco*. En transmisiones de datos, la solución consiste en emplear un cancelador de eco dentro de cada módem. Los canceladores son filtros adaptativos con coeficientes ajustables automáticamente. El cancelador de eco adaptativo intentar estimar las componentes de eco en la transmisión, el estimado se resta de la señal recibida muestreada, y la señal de error resultante puede minimizarse por mínimos cuadrados para ajustar de forma óptima los coeficientes del cancelador de eco.

- **Supresión de interferencias de banda estrecha en una señal de banda ancha:** Un problema que surge en la práctica, sobre todo en la detección de señales y en las comunicaciones digitales. Suponiendo que se tiene una secuencia formada por una señal de banda anda deseada *w(n)* distorsionada por una interferencia de banda estrecha aditiva *x(n).* Las secuencias resultan del muestreo de una señal analógica.

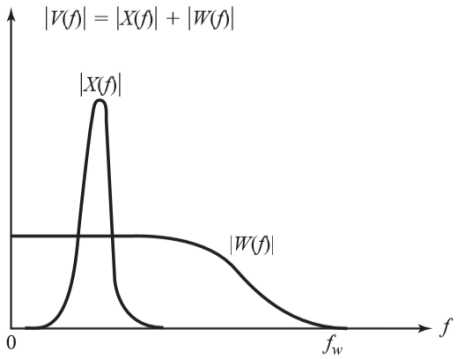


Figura 12. Interferencia de banda muy estrecha X(f) en una señal de banda ancha W(f) [4].

El objetivo es emplear un filtro que suprima la interferencia de banda estrecha. Si la interferencia no es estacionaria, su ocupación en la banda de frecuencias puede variar con el tiempo y además puede ser desconocida, por eso es necesario un filtro adaptativo.

El filtro adaptativo tiene la función de un predictor lineal del valor de la banda estrecha x(n), el valor predicho se sustrae de v(n) para proporcionar un estimado de w(n), y también se define la secuencia de error para que tras un tiempo de aprendizaje se reduzca lo más posible.

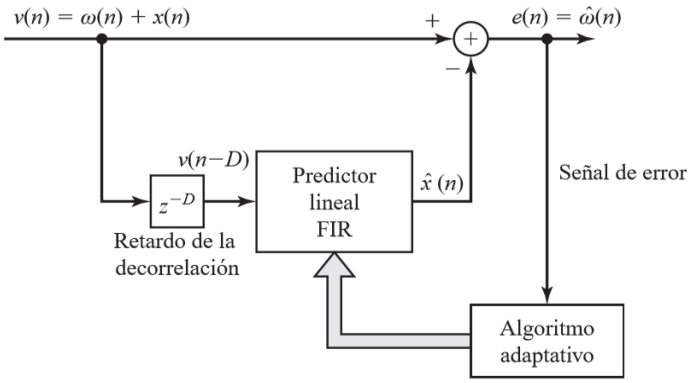


Figura 13. Filtro adaptativo para estimar y suprimir una interferencia de banda estrecha en una señal de banda ancha[4].

- **Mejorador de línea adaptativo:** Tiene la misma configuración que el filtro supresor de interferencias pero su objetivo es diferente. En el mejorador de línea adaptativo, x(n) es la señal deseada de banda estrecha y w(n) representa una componente de ruido de banda ancha.

- **Cancelación de ruido adaptativo:** La cancelación de eco, supresión de interferencia de banda estrecha y el mejorador están relacionados con otra forma del filtrado adaptativo denominado *cancelación de ruido adaptativo.* La señal de entrada principal consta de una señal deseada x(n) distorsionada por una secuencia de ruido aditivo w1(n) y una interferencia aditiva w2(n) y el propósito es utilizar las tres aplicaciones mencionadas para solucionar el problema de ruido.

- **Codificación lineal predictiva de señales de voz**: Dado que la señal digital de voz se transmite desde el origen hasta un destino,uno de los principales objetivos de los codificadores de voz es el de minimizar el número de bits necesario para representar la señal de voz, a la vez que se mantiene la inteligibilidad de la voz. El filtrado adaptativo tiene aplicación en estos sistemas de codificación de las señales de voz basados en modelos. A continuación se describe un método muy efectivo conocido como codificaciónlineal predictiva (LPC), en este tipo de codificación el tracto bucal se modela como un filtro lineal y existen dos funciones de excitación mutuamente excluyentes, utilizadas para modelar los sonidos sonoros y sordos. Los parámetros del modelo de filtro se determinan fácilmente a partir de las muestras de voz por medio de la operación de predicción lineal.

3.4.2 Filtro de Wiener

El filtro de Wiener es uno de los filtros lineales óptimos más importantes y permite determinar el comportamiento o estimar la respuesta de un sistema. Para explicar su funcionamiento se puede definir a *x(n)* como una señal discreta de N elementos, a *h(n)* como la respuesta impulso de un filtro FIR a estimar, de coeficientes M de longitud, cuyas características den lugar a la señal discreta *y[n]* lo más parecida a la señal discreta de referencia *d[n]* de N+M-1 muestras de longitud.

El parecido entre las señales se evalúa bajo el criterio del mínimo error medio cuadrático (MSE).

$$\zeta = \{[n]^2\} = E\{|d[n]-y[n]|^2\} \quad (3.4.2.1)$$

Se realiza la convolución de la señal de entrada *x[n]* con la respuesta impulso conjugada $h^*[n]$ para obtener la señal de salida:

$$y[n] = h^*[n] * x[n] = h^H X_n \quad (3.4.2.2)$$

El superíndice H indica que el vector de la respuesta impulso es transpuesto conjugado.

- El vector X_n está conformado por los desplazamientos hasta M de la señal *x[n]* .

Desarrollando la expresión del MSE y tomando la expresión de la salida *y[n]*, se obtiene:

$$\zeta = E\{|d[n]|^2\} + h^H \cdot R \cdot h^H \cdot P - P^H \cdot h \quad (3.4.2.3)$$

donde R es la matriz de autocorrelación de la señal de entrada *x[n],* definida como matriz es hermitiana y matriz de Toeplitz:

$$R = E\{X_n \cdot X_n^H\} = \begin{bmatrix} R_{xx}(0) & R^*_{xx}(1) & R^*_{xx}(2) & \dots & R^*_{xx}(N-1) \\ R_{xx}(1) & R_{xx}(0) & R^*_{xx}(1) & \dots & R^*_{xx}(N-2) \\ R_{xx}(2) & R_{xx}(1) & R_{xx}(0) & \dots & R^*_{xx}(N-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ R_{xx}(N-1) & R_{xx}(N-2) & R_{xx}(N-3) & & R_{xx}(0) \end{bmatrix} \quad (3.4.2.4)$$

Y donde P es el vector de correlación cruzada entre la señal de entrada *x[n]* y la señal de referencia o deseada *d[n]*:

$$P = E\{X_n \cdot d[n]\} = [P(0) \quad P(-1) \quad \dots \quad P(1-N)]^T \quad (3.4.2.5)$$

Derivando de la expresión anterior del error respecto al vector h^H e igualando a cero, se obtiene el **vector óptimo para la respuesta impulso del sistema o filtro Wiener:**

$$h_{opt} = R^{-1} \cdot P \quad (3.4.2.6)$$

Por tanto, la señal de salida *y[n]* queda de la siguiente forma:

$$y[n] = h^H_{opt} \cdot X_n = P^H \cdot R^{-1} \cdot X_n \quad (3.4.2.7)$$

El error mínimo cuadrático a partir del filtro Wiener queda de la siguiente manera:

$$\zeta_{min} = E\{|d[n]|^2\} - P^H \cdot R^{-1} \cdot P = E\{|d[n]|^2\} - h^H_{opt} \cdot R \cdot h_{opt} \quad (3.4.2.8)$$

Al minimizar el error se minimiza la norma del vector Ɛ[n], con esto y con la definición del producto escalar se deduce que el error será ortogonal al plano de los datos de la señal de entrada:

$$\varepsilon \perp X_n \Rightarrow E\{\varepsilon * [n] \cdot x[n-m]\} = 0 \quad con \quad m = 0,1,\dots,N-1 \quad (3.4.2.9)$$

De lo anterior se puede deducir que cuanto menos se parezca la señal de entrada del error o no correlacionadas, los coeficientes del filtro estarán más cerca del valor óptimo.

Por último, el principio de ortogonalidad permite escribir una expresión alternativa para el MSE mínimo que queda:

$$\zeta_{min} = E\{\varepsilon(n)\, d^*(n)\} \quad (3.4.2.10)$$

Hay ciertas condiciones bajo las cuales se puede asegurar que se va a encontrar la solución óptima al filtro de Wiener, es decir, los coeficientes de w que permitan que el error sea igual a 0, las condiciones son:

- La matriz R debe ser positiva definida
- Los eigenvalores de R deben ser reales y positivos

De no cumplirse las condiciones anteriores, se asegura que existe más de una solución al problema.

Si se encuentran los valores óptimos de w, la salida del filtro y el error serán ortogonales, ya que el error será igual a 0

Para poder utilizar el filtro de Wiener se debe considerar que preferentemente el sistema debe ser estacionario, si no es así debe ser por lo mínimo wide-sense-stationary (WSS) lo que significa que los valores estadísticos hasta el segundo orden (media y correlación) deben ser invariantes en el tiempo y deben conocerse.

El tema anterior fue revisado en el libro *Theory and Design of Adaptive Filters.* Texas , referencia [9] de la presente tesis.

3.4.3 Algoritmos adaptativos

Un proceso aleatorio normalmente no es estacionario, por lo cual los estimados varían con el tiempo y esto implica que los coeficientes del filtro adaptativo deben variar con el tiempo para incorporar las características estadísticas variantes.

Hay varias formas para que los coeficientes del filtro adaptativo puedan variarse con el tiempo, el método más popular consiste en adaptar el filtro recursivamente muestra a muestra, a medida que se recibe una nueva muestra de la señal. Un segundo método consiste en estimar la correlación y autocorrelación bloque a bloque, sin intentar mantener la continuidad en los valores de los coeficientes del filtro de un bloque de datos a otro, en este caso el tamaño del bloque tiene que ser relativamente pequeño, abarcando un intervalo de tiempo corto.

La mejor forma de reducir el error de estimación es con filtros recursivos y por tal motivo se consideran los algoritmos recursivos en el tiempo que actualizan los coeficientes del filtro muestra por muestra. Se consideran dos tipos de algoritmos: los algoritmos LMS (*Least-mean square)*, que se basan en una búsqueda de tipo gradiente para llevar a cabo un seguimiento de las características de la señal variante en el tiempo, conocidos también como métodos de paso descendente (*Steepest descent methods*) y los algoritmos recursivos por mínimos cuadrados.

Algoritmos por método de paso descendente

Contrario al filtro de Wiener, en los algoritmos iterativos se busca minimizar la función de costo J con sus valores estadísticos reales reemplazados por sus estimados. Una forma de atacar este problema es mediante el método de paso descendente o gradiente descendente, el cual consiste en calcular el gradiente de la función de costo, en este caso el error mínimo

cuadrático, en cada iteración del algoritmo con los valores de w (coeficientes del filtro) calculados, es decir:

$$\nabla J = \frac{\partial J}{\partial w}\Big|_{w(n)} \tag{3.4.3.1}$$

y en términos del error quedaría como:

$$\nabla \xi = \frac{\partial E\left\{\left|e(n)^2\right|\right\}}{\partial w}\Big|_{w(n)} \tag{3.4.3.2}$$

En cada iteración del algoritmo se busca que el valor del gradiente tienda o se iguale a cero ya que con esos valores de w se encontraría w_0(la solución óptima), por lo cual lo anterior nos habla de la dirección del gradiente, que como su nombre lo dice, será descendente con el fin de hallar w_0. Todo lo anterior se puede apreciar en la siguiente imagen:

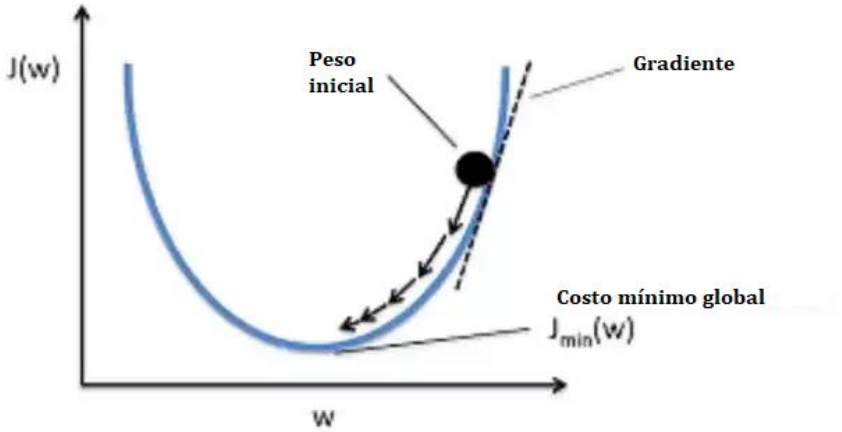


Figura 14. Método de gradiente descente [7].

Existen diversos algoritmos que parten de utilizar el método de gradiente descendente, siendo el más importante el algoritmo LMS (*Least-mean square*) cuyas variaciones dan lugar a otros algoritmos. A continuación se detallan un poco más los algoritmos.

Algoritmo LMS

Este algoritmo no requiere de conocer los valores estadísticos de media y varianza de la señal de entrada, se basa en la estimación en cada iteración de la matriz de correlación R_xy el vector de correlación cruzada p_{dx} teniendo la señal de entrada a filtrar $x(n)$ y la señal deseada $d(n)$, pero con la ventaja de que no requiere de encontrar la matriz inversa de R_x.

Debido a que el algoritmo trabaja con estimaciones y no con valores exactos a lo largo de su realización se van produciendo errores que si bien no permiten llegar al vector óptimo de coeficientes del filtro, permite llegar a una solución con una variación muy pequeña alrededor de la solución óptima. Sin embargo, hay ciertos parámetros involucrados en el algoritmo que permiten reducir la variación lo más posible.

Para este algoritmo, se requiere tener la señal de entrada $x(n)$ y la señal deseada $d(n)$, a partir de estas dos señales se calculan los coeficientes del filtro, la señal de salida y el error del filtro:

$y(n) = w^T(n)x(n)$ salida del filtro (3.4.3.3)

$e(n) = d(n) - y(n)$ error del filtro (3.4.3.4)

$$w(n+1) = w(n) \underbrace{+ 2\mu e(n)x(n)}_{\text{GRADIENTE}} \text{ actualización del vector de pesos del filtro} \qquad (3.4.3.5)$$

Dentro del gradiente, μ es una constante llamada paso de adaptación que se escoge de acuerdo a los requerimientos de la aplicación del filtro.

Otro parámetro a escoger para el algoritmo es M que indica el orden del filtro o número de coeficientes del filtro.

Para analizar el desempeño del filtro se establece una función de costo que comúnmente es la de error mínimo cuadrático $J = E\{|e(n)^2|\}$y que dará lugar a la curva de aprendizaje del filtro, que es la gráfica de desempeño de *J* contra el número de iteración *n.*

Variaciones del algoritmo LMS

- LMS Complejo

El algoritmo LMS complejo se utiliza cuando la señal a filtrar *x(n)* es compleja, depende de los mismos parámetros μ y M, pero la manera para calcular la salida y los coeficientes del filtro varía. Quedan de la siguiente forma:

$$y(n) = w^H(n)x(n)^{[7]} \qquad (3.4.3.6)$$

donde el superíndice H indica la conjugación transpuesta del vector de coeficientes

$$w(n+1) = w(n) + 2\mu e^*(n)x(n)^{[7]} \qquad (3.4.3.7)$$

donde el asterística indica que debe realizarse la conjugación compleja del error

- LMS Normalizado(NLMS) y ε-NLMS

En el algoritmo LMS normalizado el paso de adaptación, μ, pasa de ser un valor fijo a un valor variable en el tiempo, $\mu(n)$, y ese paso variable es inversamente proporcional a la energía instantánea de la señal de entrada *x(n)*.

$$\mu(n) = \frac{1}{||x(n)||\text{^}2} \qquad (3.4.3.8)$$

quedando el algoritmo finalmente como:

$$w(n+1) = w(n) + \frac{\mu}{||x(n)||\text{^}2} e(n)x(n) \qquad (3.4.3.9)$$

Una variación del algoritmo NLMS (ε-NLMS) consiste en agregar un número positivo pequeño ε pero evitar la división entre cero cuando la señal de entrada se convierte en cero o extremadamente pequeña, quedando el algoritmo como:

$$w(n+1) = w(n) + \frac{\mu}{\varepsilon + ||x(n)||\text{^}2} e(n)x(n) \qquad (3.4.3.10)$$

El algoritmo NLMS resulta ser más robusto que el algoritmo LMS normal

- Algoritmos LMS acelerados

Partiendo del algoritmo LMS se propone avanzar en dirección contraria del gradiente descendente y a la ecuación de actualización de los coeficientes *w* se le agrega una matriz C de la siguiente manera:

$$w(n+1) = w(n) + \mu e(n) \underset{\downarrow}{C} x(n) \qquad (3.4.3.11)$$

Matriz seleccionada para hacer converger más rápido el algoritmo

- Signo de error y signo de dato de LMS

Para estas variaciones se le aplica la función signo al error y al dato de entrada en la ecuación de actualización de los coeficientes *w.* Puede aplicarse la función de signo únicamente al error, al dato de entrada o una combinación donde se aplica a ambos, a continuación se muestra cómo queda la ecuación de actualización en cada uno de los casos:

➔ Caso 1: Signo del error

$$w(n+1) = w(n) + \mu sign[e(n)]x(n) \tag{3.4.3.12}$$

donde:

$$sign[e(n)] = \begin{cases} 1 & \text{si } d(n)\text{-}y(n)>0 \\ 0 & \text{si } d(n)\text{-}y(n)=0 \\ -1 & \text{si } d(n)\text{-}y(n)<0 \end{cases} \tag{3.4.3.13}$$

Lo anterior facilita la implementación ya que ofrece una mejor idea de la manera en que deben cambiar los coeficientes de adaptación para llegar al óptimo.

➔ Caso 2: Signo del dato

$$w(n+1) = w(n) + \mu e(n) sign[x(n)] \tag{3.4.3.14}$$

donde la función de signo se aplica elemento por elemento al vector de datos *x(n)*

➔ Caso 3: Signo del error y signo del dato

$$w(n+1) = w(n) + \mu sign[e(n)] sign[x(n)] \tag{3.4.3.15}$$

Cabe mencionar que cada uno de los casos anteriores se puede aplicar al algoritmo NLMS (LMS Normalizado).

Algoritmos recursivos

Los algoritmos recursivos se desarrollaron con la finalidad de ir construyendo muestra a muestra el vector de pesos óptimos para el filtro; sin embargo, la implementación de los algoritmos recursivos necesita que para cada actualización de los coeficientes del filtro se calcule la matriz inversa de autocorrelación R_x y requiere de predicción de error y de señal de salida del filtro.

Por lo antes descrito y tomando en cuenta la complejidad entre los algoritmos, se optó por enfocar esta tesis a los algoritmos de paso descendente.

4. Diseño de la solución o desarrollo.

Para el diseño de la solución a los problemas o errores que se menciona se han visto en los auxiliares auditivos se traducen a requerimientos de diseño y se muestran en la siguiente tabla.

Tabla 4. Tabla de requerimientos para el diseño de la solución

PROBLEMÁTICA	REQUERIMIENTO
Falta de claridad en la voz	Identificar y amplificar señales de voz
Mucho ruido ambiental	Identificar y suprimir ruido
Realimentación acústica	Identificar realimentación y evitar que llegue al usuario

Tomando en cuenta los requerimientos el primer paso para desarrollar la solución es la selección del modelo del filtro y algoritmo que se adecúe mejor a la aplicación.

4.1 Selección del modelo del sistema de filtrado y algoritmo adaptativo.

Como primera selección se establece que la estructura del filtro será del tipo FIR, ya que para su desarrollo únicamente serán útiles y necesarios los valores pasados y actuales de entrada y salida del filtro.

Con el fin de reducir el ruido ambiental y evitar la realimentación acústica, se propone el modelo cancelador de ruido adaptativo, que permite reducir el ruido, el eco y la reverberación. El modelo mencionado nos entregará una señal con prioridad en la voz por lo cual únicamente será necesario establecer una ganancia mayor a la unitaria en el filtro.

Una vez elegido el modelo, las consideraciones para la selección del algoritmo son las siguientes:

- Robustez: La capacidad del algoritmo para ejecutarse sin generar fallas o bloquearse.
- Complejidad computacional: Se refiere a la cantidad de operaciones que el algoritmo va a realizar o costo de cálculos, para este tipo de algoritmo se realizan 2*M+1 sumas y 2*M+1 multiplicaciones por iteración, eso se verá traducido en el tiempo de ejecución de programa.
- Desajuste: Mide la diferencia entre la solución de Wiener y la obtenida con el algoritmo adaptativo. Generalmente, el desajuste es inversamente proporcional a la velocidad de convergencia y a la complejidad.
- Precisión: Que el algoritmo llegue al resultado óptimo.
- Velocidad de convergencia: Que tan rápido el algoritmo llega o converge al valor óptimo o solución de Wiener.
- Conocimiento de valores estadísticos de las señales: Si se conocen o no los valores estadísticos, cuál es el algoritmo apropiado para implementar.
- No estacionariedad: El proceso o señales a filtrar no son estacionarias, varían en el tiempo.

Con base en esas consideraciones se optó por utilizar el algoritmo ε-NLMS, ya que sus características permiten reducir el costo computacional comparándolo con el algoritmo LMS y

gracias a que el algoritmo permite que el paso de adaptación sea variable, ofrece una ventaja en cuanto a la velocidad de convergencia y desajuste.

Con el fin de hacer al algoritmo más eficiente se propone agregar un coeficiente de fuga (Coefficiente *Leakage*), el cual hace que para cada estimación de la matriz de autocorrelación R_x se evite tener eigenvalores iguales a cero con lo cual se garantiza que el filtro converja a una única solución.

4.2 Estimación de parámetros

En esta sección se eligen los valores de los parámetros que involucra el algoritmo ε-NLMS y también el valor del coeficiente de fuga (γ).

Selección del parámetro de paso de adaptación (μ)

El parámetro de paso de adaptación nos indica la velocidad con la que el algoritmo convergerá a el valor de coeficientes óptimos pero también nos indica qué tanto ruido de desajuste pueda haber en el filtro y hay que ser muy cuidadosos al seleccionarlo ya que como tal no existe un fórmula para su cálculo pero sí recomendaciones de elección, que se pudieron obtener de los autores de las referencias [8] y [9] de la bibliografía:

- El rango de valores de μen el que se asegura que el algoritmo converge a la solución óptimo es:

$$0 < \mu < \frac{2}{\mu_{máx}} \tag{4.2.1}$$

- Para tener una convergencia suave se recomienda escoger un μ de orden dos veces menor al $\mu_{máx}$ el cual se obtiene de la siguiente manera:

$$\mu_{máx} = \frac{2}{\lambda_{máx}} \tag{4.2.2}$$

- Al tener un valor más grande paso aumenta la velocidad de convergencia a la señal deseada pero también aumenta el ruido de desajuste.

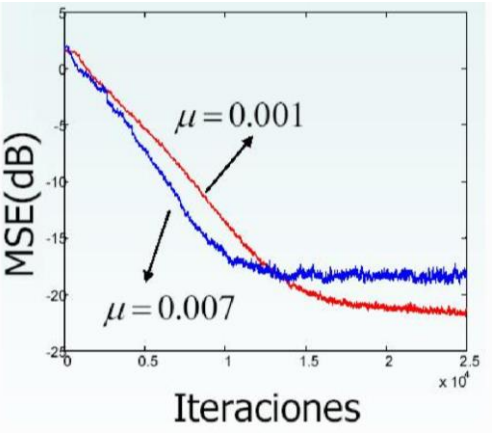


Figura 15. Gráfica de ruido de desajuste con diferentes valores de μ

Tomando en cuenta las recomendaciones, se propone iniciar el algoritmo con un valor en alfa de 0.015 para poder calcular en cada iteración el paso de adaptación, valor que irá cambiando con el tiempo para irse adaptando mejor.

Selección de orden del filtro *M*

El orden del filtro debe ser menor el número de datos N de la señal de entrada *x(n)* y de él también depende la velocidad de convergencia y el ruido de desajuste: si se toma un valor de M pequeño respecto de N, el ruido de desajuste será mayor, en cambio si se toma un valor mayor de M el ruido de desajuste será menor pero el tiempo de convergencia se verá aumentado. Sin embargo, se debe considerar que para cada iteración del algoritmo el número de adiciones que requerirá es de $2 * M + 1$ y el número de multiplicaciones será también de $2 * M + 1$.

Con base en lo anterior se plantea que lo que se debe sopesar es exactitud y precisión contra tiempo de convergencia. Para la selección óptima del orden del filtro en la literatura se recomienda comenzar a hacer pruebas con un número pequeño razonable e ir observando cómo se va comportando el error mínimo cuadrático (MSE), el mejor valor de M será el que combinado con el paso de adaptación reduzca lo más posible el error.

Selección de ε

En el algoritmo ε-NLMS el valor de ε debe ser un número positivo pequeño, debido a que no hay otras restricciones para la selección de su valor y sólo sirve para evitar la división entre 0, se elegirá un valor un valor cercano a alfa que será 0.02.

Selección del coeficiente de fuga (γ)

El coeficiente de fuga se introduce al algoritmo con el fin de solucionar el problema que se presenta cuando la matriz de autocorrelación asociada R tiene uno o más eigenvalores iguales a cero, lo que podría hacer que el filtro no converja a una única solución. El coeficiente de fuga se introduce de la siguiente manera:

$$w(n+1) = (1 - \mu\gamma) \cdot w(n) + \mu e(n)x(n) \qquad (4.2.3)$$

El valor de γdebe ser tan pequeño como μ y ya que ambos valores son pequeños comparados con 1 y positivos en signo, el factor $1 - \mu\gamma$ será casi 1. Basado en lo anterior, se propone un coeficiente de fuga de 0.1, ya que el multiplicarse por el paso de adaptación de cada iteración el valor se verá reducido.

4.3 Implementación en software

La implementación en software se realiza con la ayuda MATLAB ya que tiene un ambiente fácil de utilizar y funciones que ayudan al procesamiento de señales. Se plantea realizar pruebas con 3 diferentes señales: una canción en la que se buscará eliminar la voz, una conversación grabada con ruido y la conversación sin ruido como señal deseada con el fin de eliminar el ruido ambiental y finalmente, una conversación grabada con ruido de la que se tendrá que calcular su deseada. Además, en cada una de las señales se variará el valor del orden del filtro y el valor del paso de adaptación, monitoreando el error obtenido y el tiempo de ejecución del programa.

4.3.1 Conversión A/D

- Señal 1:

La señal de la canción ya se encuentra digitalizada pero es importante conocer la frecuencia de muestreo:

$$fs = 44100\ Hz$$

- Señal 2:

La conversación deseada así como la conversación con ruido se grabaron con una aplicación en celular que permitía elegir la frecuencia de muestreo y se optó por:

$$fs = 44100\ Hz$$

- Señal 3:

La conversación con ruido se graba con una aplicación en celular con una de frecuencia de muestreo de:

$$fs = 44100\ Hz$$

4.3.2 Filtrado y algoritmo adaptativo.

En el Anexo A se presenta el código del algoritmo adaptativo que se diseñó y posteriormente las pruebas realizadas con las combinaciones de los diversos parámetros para cada una de las señales. Los parámetros que se utilizarán para evaluar el rendimiento de cada una de las pruebas es el de correlación y SNR (*Signal-to-Noise Ratio*) o su variación SSNR(*Signal-to-Signal Noise Ratio)*, el primero permitiendo observar la similitud entre la señal deseada y la salida del filtro, ya que cuantifica la variación conjunta entre las dos señales a medida que los parámetros van cambiando y el segundo, nos indicará el valor de la potencia de la señal sobre el valor de la potencia del ruido o bien el valor de la potencia de la señal sobre el valor de la señal a ruido y se encuentra expresado en decibeles.

El coeficiente de correlación está definido de la siguiente manera:

$$\rho_{xy} = \frac{Cov_{xy}}{\sigma_x \sigma_y} \qquad (4.3.2.1)$$

Donde:

Cov_{xy} = Covarianza entre "x" y "y" $\quad \sigma_x$ = desviación de x $\quad \sigma_y$ = desviación de y

El SNR queda expresado de la siguiente manera:

$$SNR = 10log_{10}\frac{Psignal}{Pnoise} \qquad (4.3.2.2)$$

Donde:

$$Psignal = \frac{1}{N}\sum_{n=0}^{N}|\hat{s}(n)|^2 \quad \text{y} \quad Pnoise = \frac{1}{N}\sum_{n=0}^{N}|s(n) - \hat{s}(n)|^2 = \xi(n)$$

El SNR queda expresado de la siguiente manera:

$$SNR = 10log_{10}\frac{Psignal}{Psignal-noise} \qquad (4.3.2.3)$$

Donde:

$Psignal = \frac{1}{N}\sum_{n=0}^{N}|\hat{s}(n)|^2$ y $Psignal - noise = \frac{1}{N}\sum_{n=0}^{N}|s(n) - \hat{s}(n)|^2 = \xi(n)$

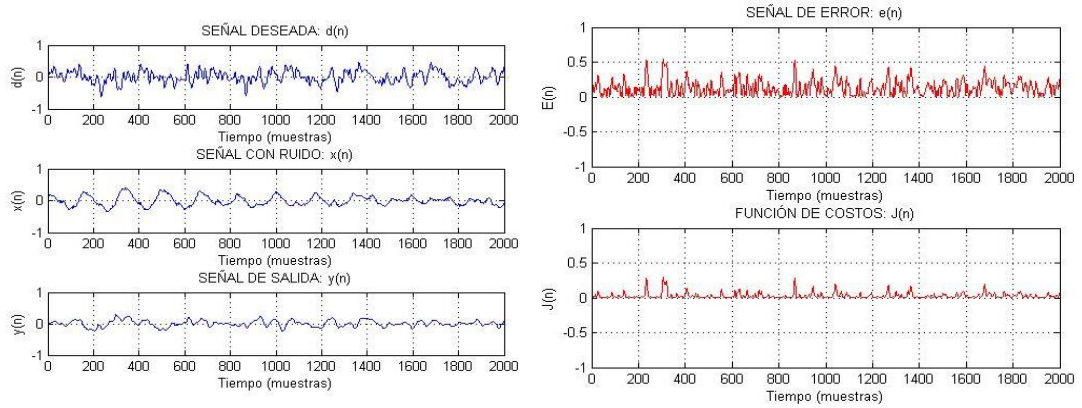
Los valores con los que se harán las combinaciones para las pruebas son los siguientes:

- ➔ M será de 64, 128, 256 y dos de 512.
- ➔ alfa será de 0.015, 0.009 y 0.003
- ➔ ep de 0.02 y 0.008
- ➔ El coeficiente de fuga será de 0.1

Cada uno de los valores de M se combinará con cada valor de alfa y ep, por tal motivo se llevan a cabo numerosas pruebas.

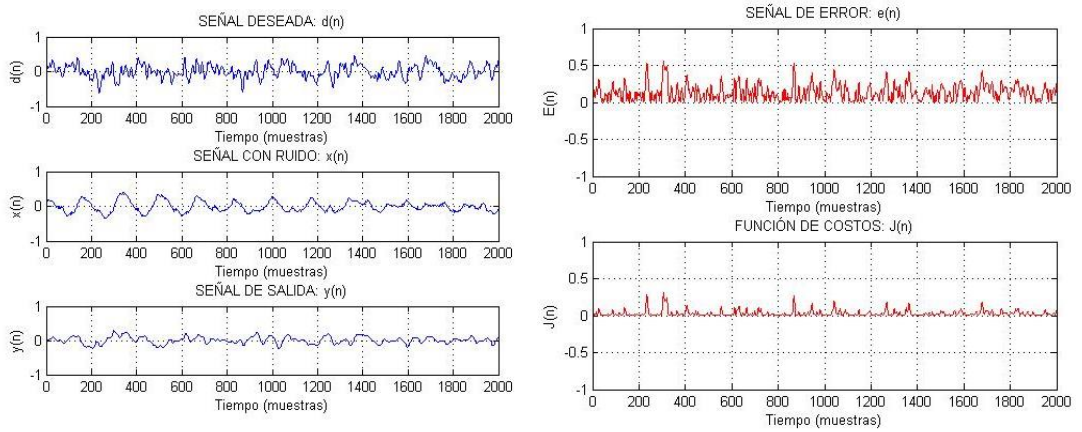
Señal 1:

Prueba 1: M=64, alfa=0.015, ep=0.02, ga=0.1



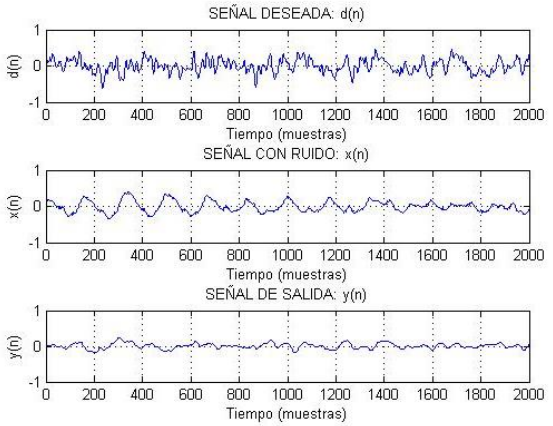
Promedio de la función de costos= 0.0176 Promedio del error = 1.4410e-04
Promedio del error absoluto =0.0948 Tiempo de ejecución= 235.888329 [seg]
Coeficiente de correlación=0.5647
SSNR d,x= 4.0961 SSNR y',x=-2.6968

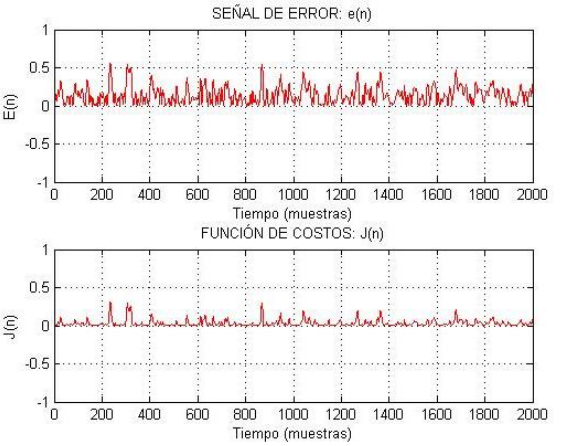
Prueba 2: M=64, alfa=0.015, ep=0.008, ga=0.1



Promedio de la función de costos = 0.0174 Promedio del error = 1.2693e-04
Promedio del error absoluto = 0.0942 Tiempo de ejecución=229.223665 [seg]
Coeficiente de correlación=0.5735
SSNR d,x=4.0961[dBc] SSNR y',x=-2.5737[dBc]

Prueba 3: M=64, alfa=0.009, ep=0.02, ga=0.1



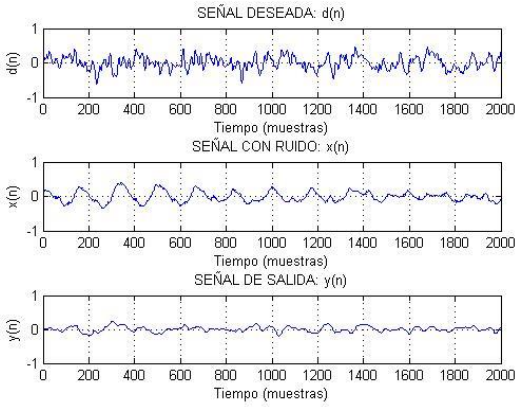


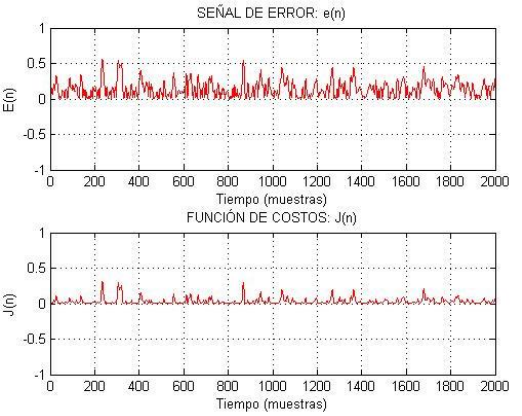
Promedio de la función de costos = 0.0203
Promedio del error absoluto = 0.1017
Coeficiente de correlación=0.4564
SSNR d,x=4.0961[dBc]

Promedio del error = 4.3705e-04
Tiempo de ejecución=226.761399 [seg]

SSNR y',x=-3.9858[dBc]

Prueba 4: M=64, alfa=0.009, ep=0.008, ga=0.1



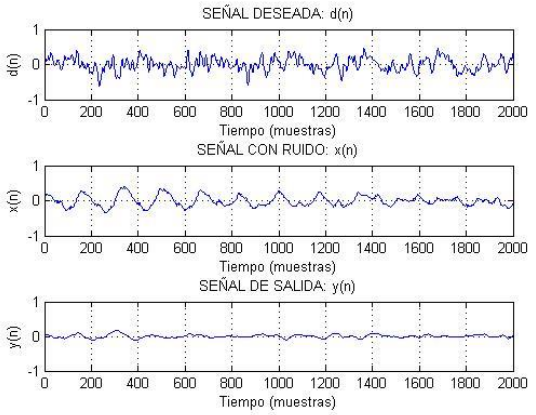


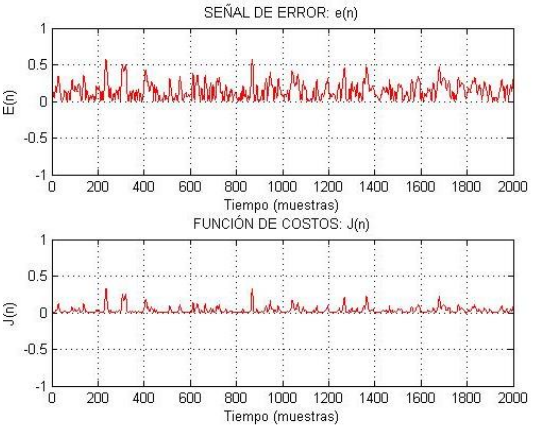
Promedio de la función de costos = 0.0201
Promedio del error absoluto =0.1013
Coeficiente de correlación=0.4632
SSNR d,x=4.0961[dBc]

Promedio del error = 4.2648e-04
Tiempo de ejecución=224.005704 [seg]

SSNR y',x=-3.8408[dBc]

Prueba 5: M=64, alfa=0.003, ep=0.02, ga=0.1



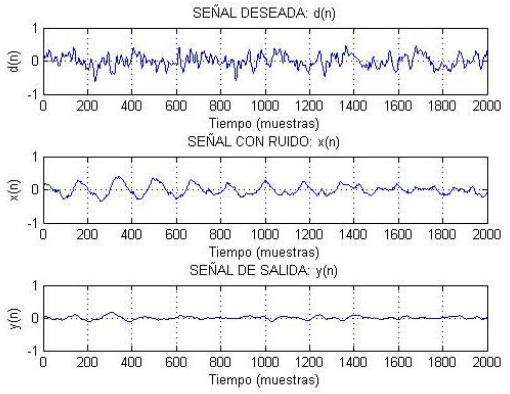


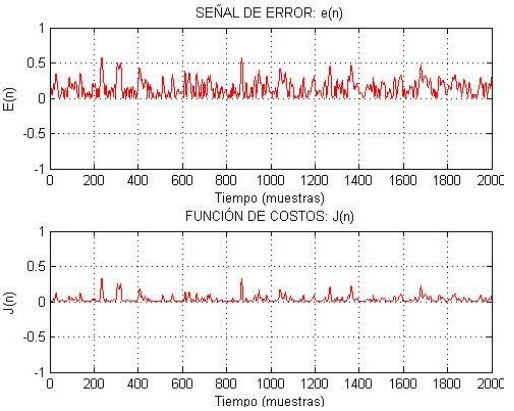
Promedio de la función de costos = 0.0233
Promedio del error absoluto= 0.1092
Coeficiente de correlación=0.2887
SSNR d,x=4.0961[dBc]

Promedio del error = 4.7174e-04
Tiempo de ejecución=229.528287 [seg]

SSNR y',x=-7.1214[dBc]

Prueba 6: M=64, alfa=0.003, ep=0.008, ga=0.1



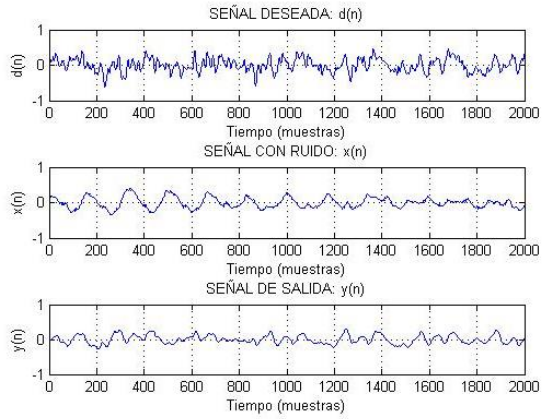


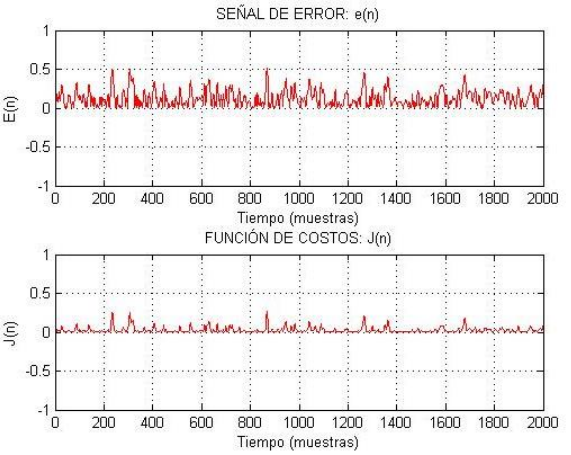
Promedio de la función de costos = 0.0233
Promedio del error absoluto = 0.1091
Coeficiente de correlación=0.2912
SSNR d,x=4.0961[dBc]

Promedio del error = 4.9400e-04
Tiempo de ejecución=233.853612 [seg]

SSNR y',x=-6.9466[dBc]

Prueba 7: M=128, alfa=0.015, ep=0.02, ga=0.1



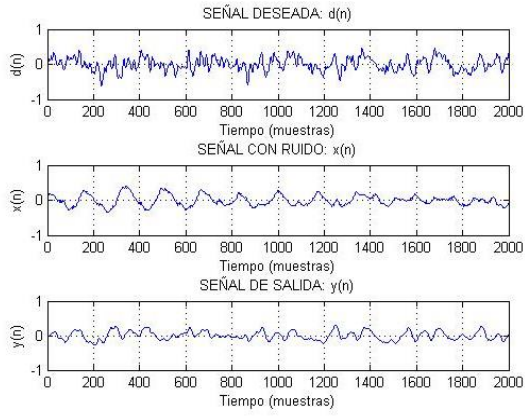


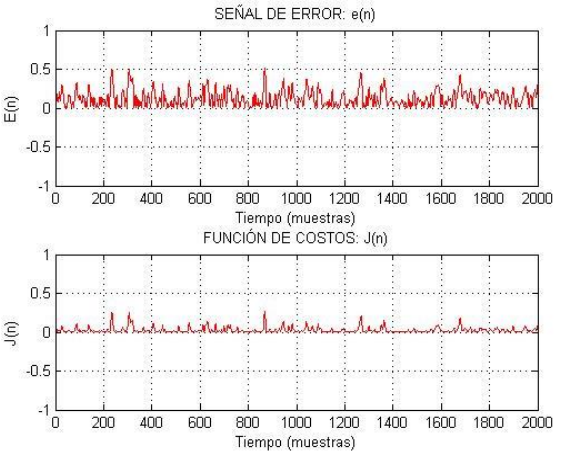
Promedio de la función de costos = 0.0166
Promedio del error absoluto =0.0919
Coeficiente de correlación=0.5994
SSNR d,x=4.0961[dBc]

Promedio del error = 6.4000e-04
Tiempo de ejecución=345.387290 [seg]

SSNR y',x=-1.8927[dBc]

Prueba 8: M=128, alfa=0.015, ep=0.008, ga=0.1



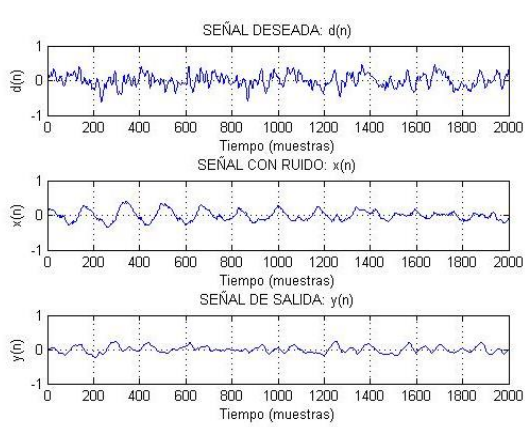


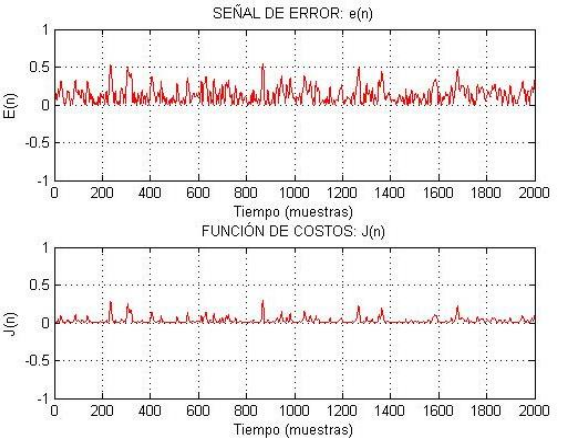
Promedio de la función de costos = 0.0164
Promedio del error absoluto = 0.0916
Coeficiente de correlación=0.6036
SSNR d,x=4.0961[dBc]

Promedio del error= 6.3612e-04
Tiempo de ejecución=341.003143 [seg]

SSNR y',x=-1.8395[dBc]

Prueba 9: M=128, alfa=0.009, ep=0.02, ga=0.1



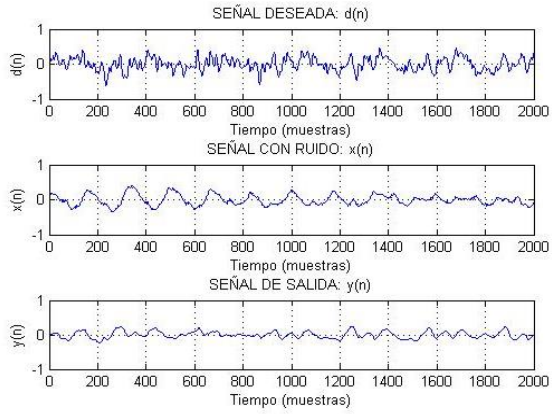


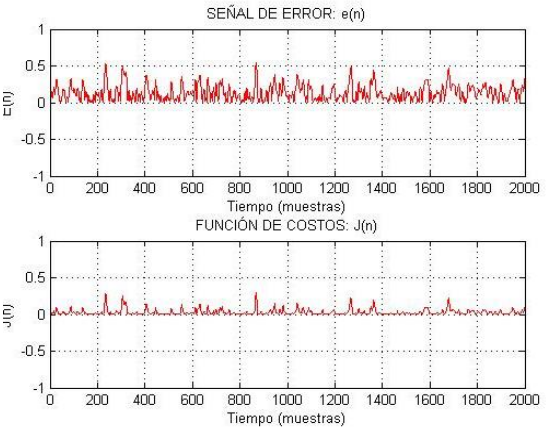
Promedio de la función de costos = 0.01892
Promedio del error absoluto = 0.0990
Coeficiente de correlación=0.4982
SSNR d,x=4.0961[dBc]

Promedio del error= 7.1019e-04
Tiempo de ejecución=356.918351 [seg]

SSNR y',x=-3.1412[dBc]

Prueba 10: M=128, alfa=0.009, ep=0.008, ga=0.1



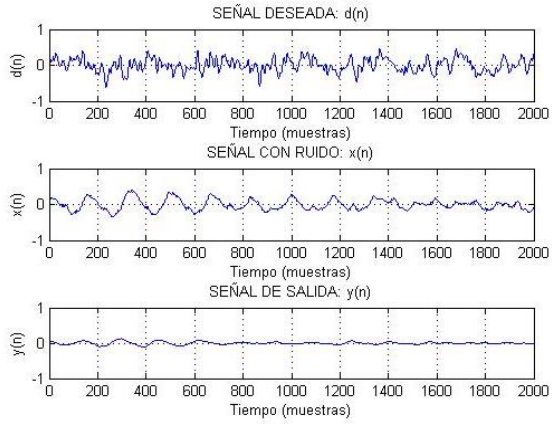


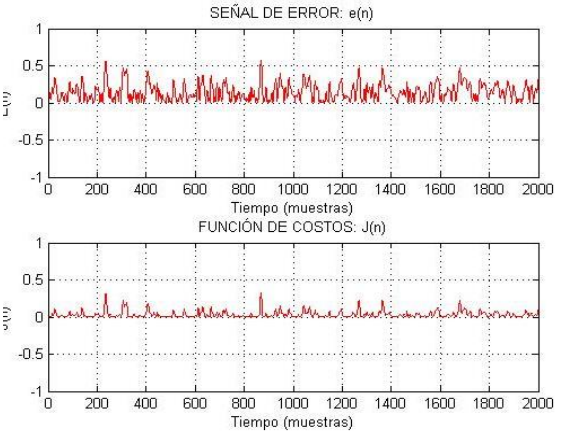
Promedio de la función de costos = 0.0216
Promedio del error absoluto = 0.1045
Coeficiente de correlación=0.3947
SSNR d,x=4.0961[dBc]

Promedio del error = 4.4821e-04
Tiempo de ejecución=374.230469 [seg]

SSNR y',x=-5.2857[dBc]

Prueba 11: M=128, alfa=0.003, ep=0.02, ga=0.1



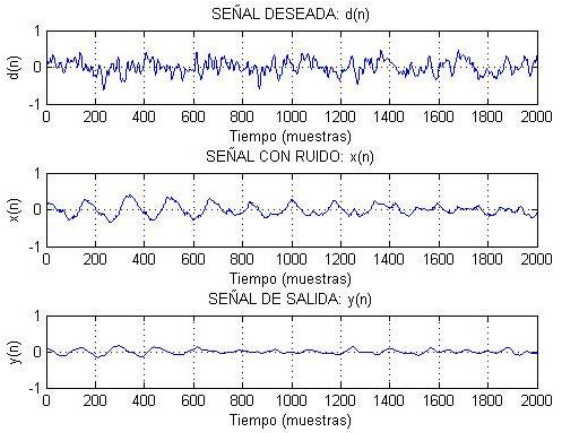


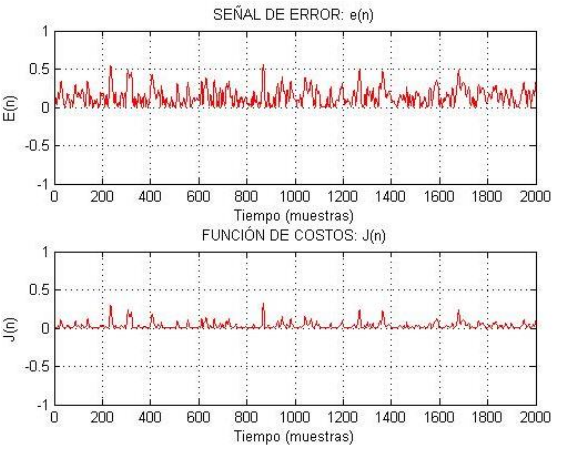
Promedio de la función de costos =0.0236
Promedio del error absoluto =0.1096
Coeficiente de correlación=0.2719
SSNR d,x=4.0961[dBc]

Promedio del error = 2.3461e-04
Tiempo de ejecución=379.387097 [seg]

SSNR y',x=-8.3316[dBc]

Prueba 12: M=128, alfa=0.003, ep=0.008, ga=0.1



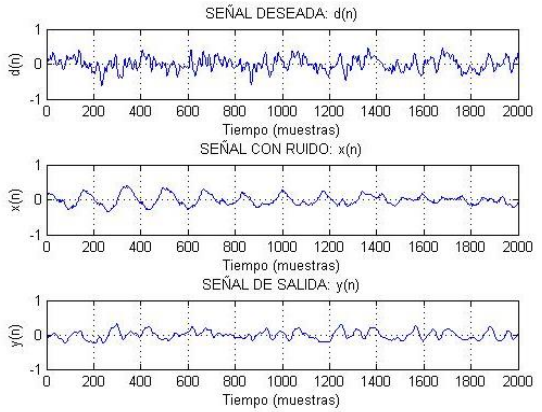


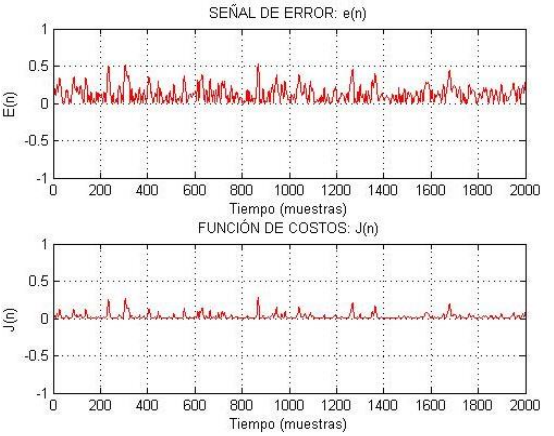
Promedio de la función de costos = 0.0226
Promedio del error absoluto = 0.1073
Coeficiente de correlación=0.3351
SSNR d,x=4.0961[dBc]

Promedio del error = 4.3630e-04
Tiempo de ejecución=374.995640 [seg]

SSNR y',x=-5.9455[dBc]

Prueba 13: M=256, alfa=0.015, ep=0.02, ga=0.1



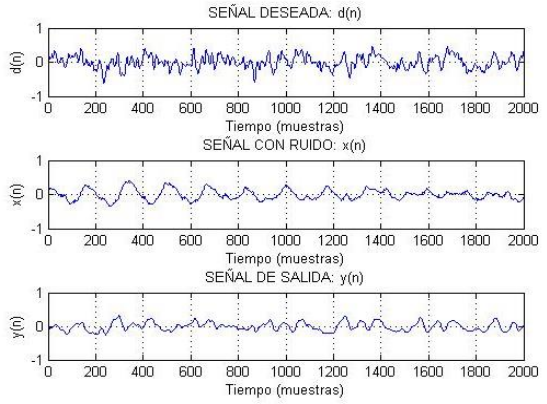


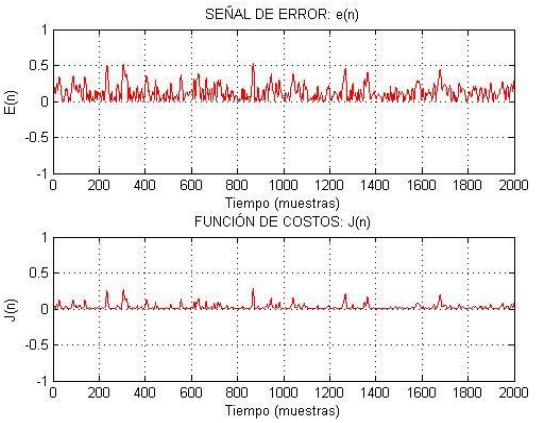
Promedio de la función de costos = 0.0154
Promedio del error absoluto = 0.0882
Coeficiente de correlación=0.6368
SSNR d,x= 4.0961[dBc]

Promedio del error = 2.0714e-04
Tiempo de ejecución=639.511216 [seg]

SSNR y',x=-1.3045[dBc]

Prueba 14: M=256, alfa=0.015, ep=0.008, ga=0.1



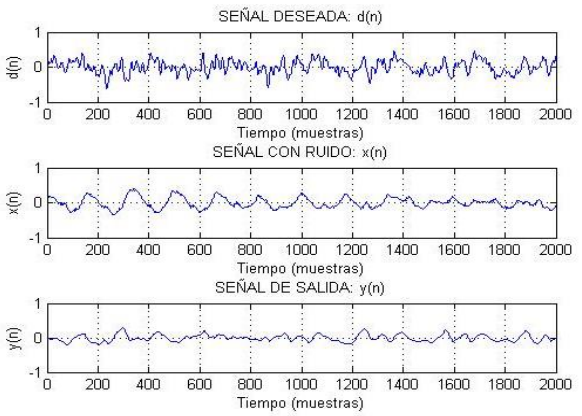


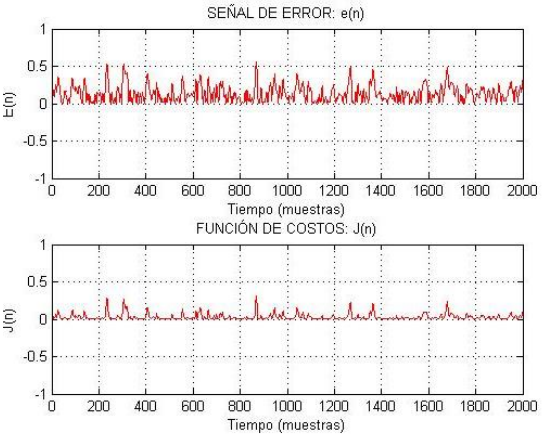
Promedio de la función de costos =0.0153
Promedio del error absoluto =0.0881
Coeficiente de correlación=0.6387
SSNR d,x=4.0961[dBc]

Promedio del error =2.0636e-04
Tiempo de ejecución=659.819352 [seg]

SSNR y',x=-1.2812[dBc]

Prueba 15: M=256, alfa=0.009, ep=0.02, ga=0.1



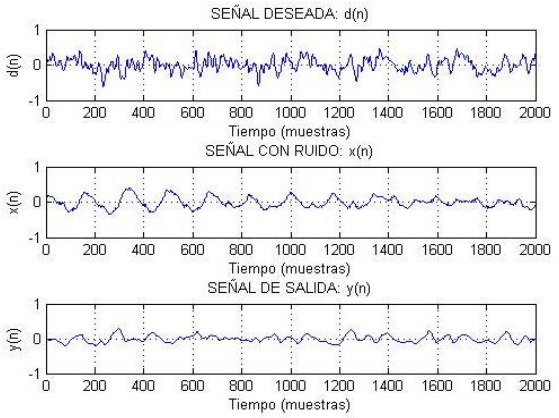


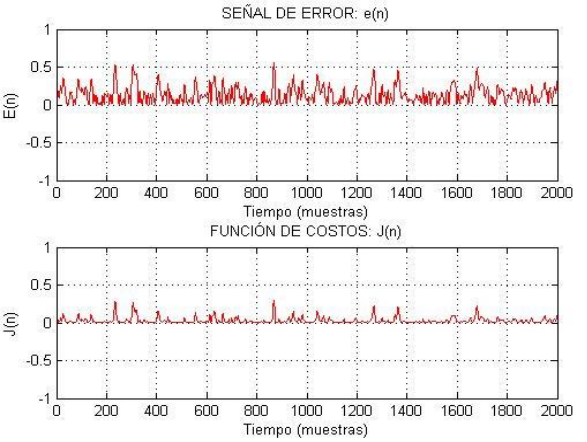
Promedio de la función de costos =0.0181
Promedio del error absoluto = 0.0956
Coeficiente de correlación=0.5431
SSNR d,x=4.0961[dBc]

Promedio de error = 3.0088e-04
Tiempo de ejecución=632.641323 [seg]

SSNR y',x=-2.4717[dBc]

Prueba 16: M=256, alfa=0.009, ep=0.008, ga=0.1



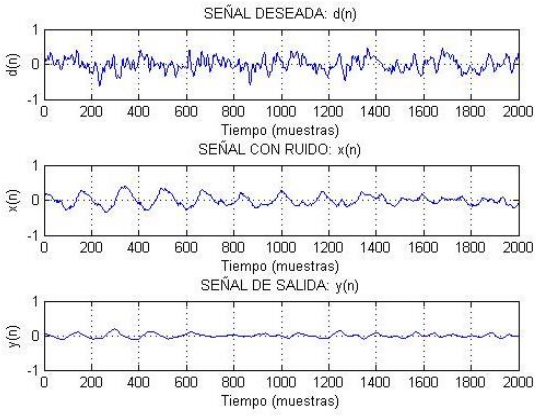


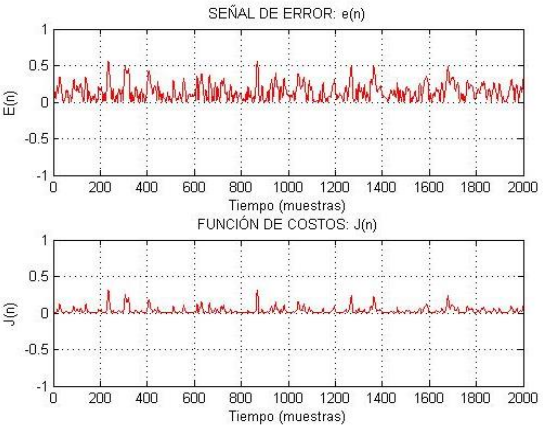
Promedio de la función de costos = 0.0180
Promedio del error absoluto =0.0955
Coeficiente de correlación=0.5447
SSNR d,x=4.0961[dBc]

Promedio del error = 3.0119e-04
Tiempo de ejecución=644.665609 [seg]

SSNR y',x=-2.4465[dBc]

Prueba 17: M=256, alfa=0.003, ep=0.02, ga=0.1



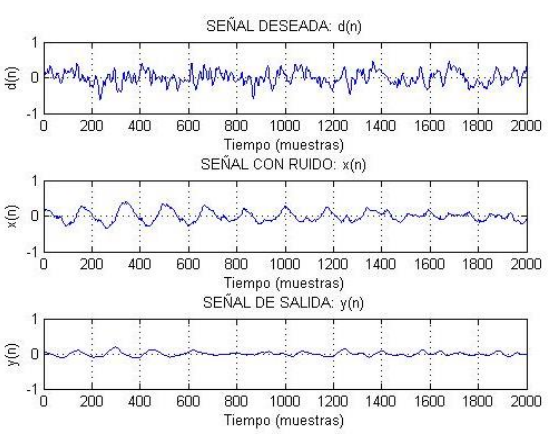


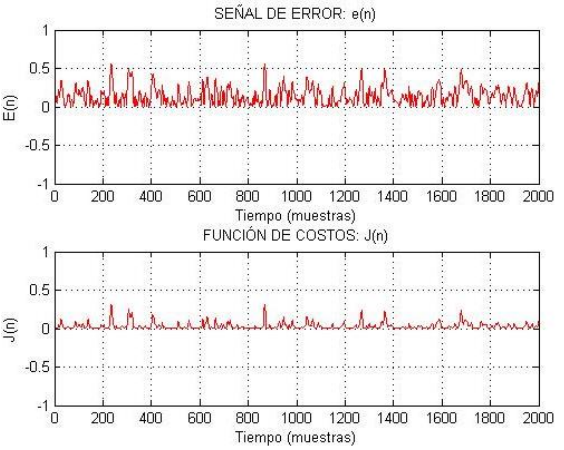
Promedio de la función de costos =0.0219
Promedio del error absoluto =0.1052
Coeficiente de correlación=0.3731
SSNR d,x=4.0961[dBc]

Promedio del error = 2.6953e-04
Tiempo de ejecución=671.321931 [seg]

SSNR y',x=-5.0055[dBc]

Prueba 18: M=256, alfa=0.003, ep=0.008, ga=0.1



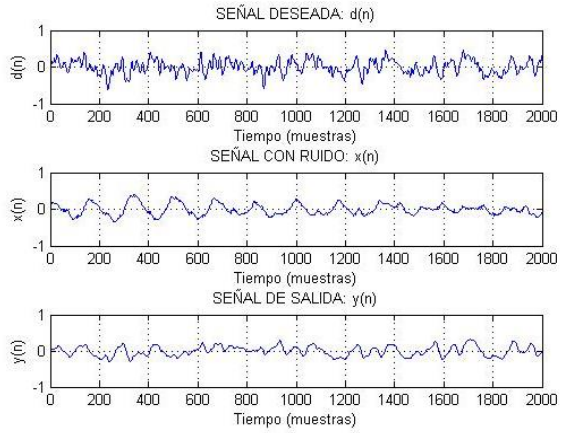


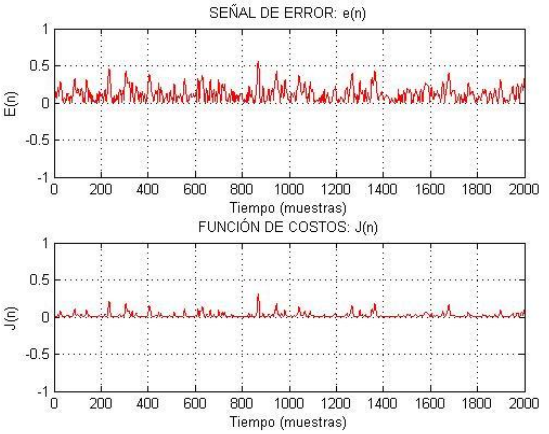
Promedio de la función de costos =0.0219
Promedio del error absoluto = 0.1052
Coeficiente de correlación=0.374
SSNR d,x=4.0961[dBc]

Promedio del error = 2.6973e-04
Tiempo de ejecución=643.870748 [seg]

SSNR y',x=-4.9747[dBc]

Prueba 19: M=512, alfa=0.015, ep=0.02, ga=0.1



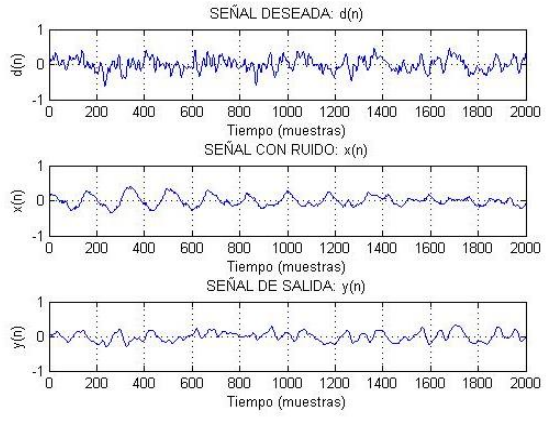


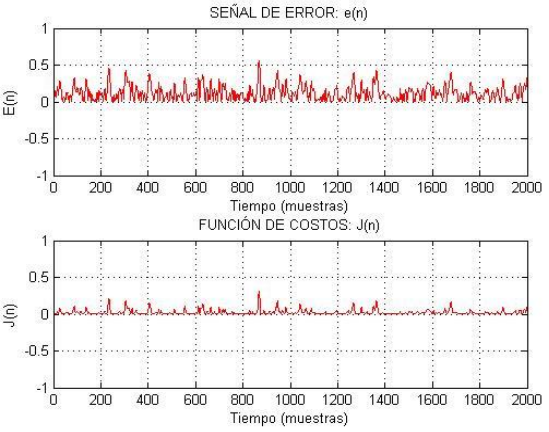
Promedio de la función de costos =0.0144
Promedio del error absoluto =0.0849
Coeficiente de correlación=0.6658
SSNR d,x=4.0961[dBc]

Promedio del error =3.2007e-04
Tiempo de ejecución=1143.011429 [seg]

SSNR y',x=-0.7394[dBc]

Prueba 20: M=512, alfa=0.015, ep=0.008, ga=0.1



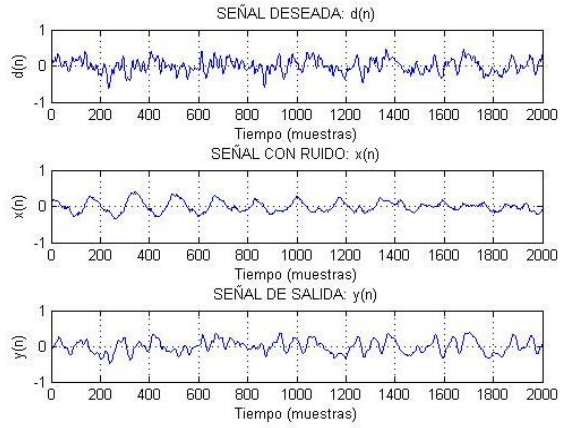


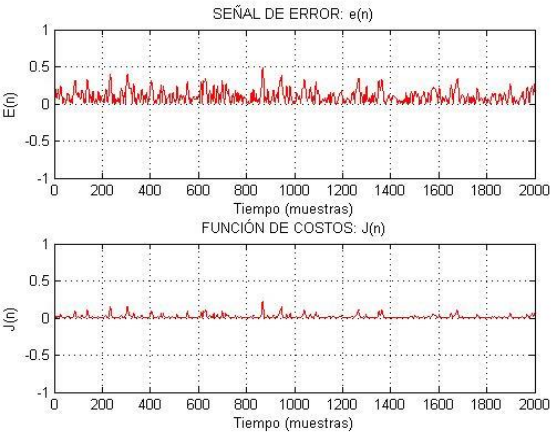
Promedio de la función de costos =0.0144
Promedio del error absoluto =0.0848
Coeficiente de correlación=0.6667
SSNR d,x=4.0961[dBc]

Promedio del error= 3.2032e-04
Tiempo de ejecución=1167.943230 [seg]

SSNR y',x=-0.7291[dBc]

Prueba 21: M=512, alfa=0.03, ep=0.00000008, ga=0.1



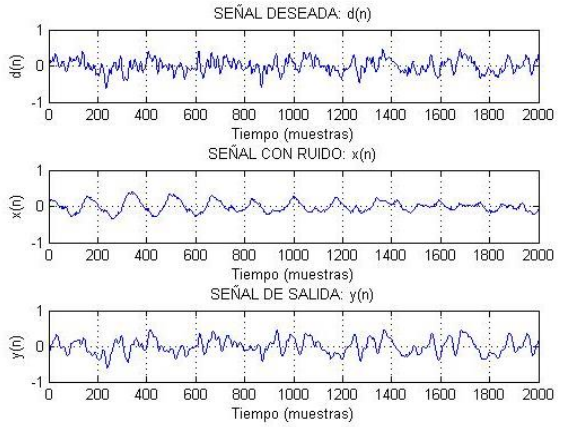


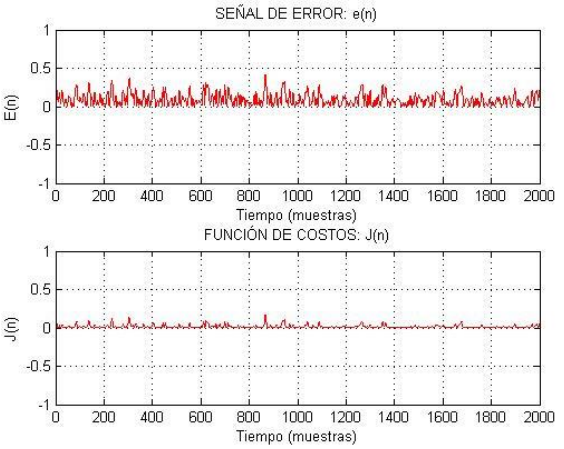
Promedio de la función de costos =0.0101
Promedio del error absoluto =0.0712
Coeficiente de correlación=0.7863
SSNR d,x=4.0961[dBc]

Promedio del error= 3.6562e-04
Tiempo de ejecución=1205.074933 [seg]

SSNR y',x=0.6003[dBc]

Prueba 22: M=512, alfa=0.045, ep=0.000000000008, ga=0.1



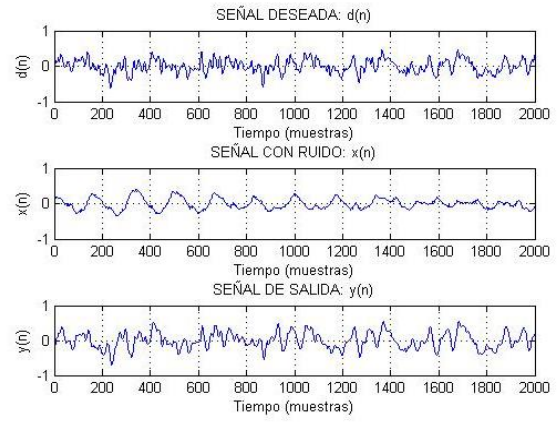


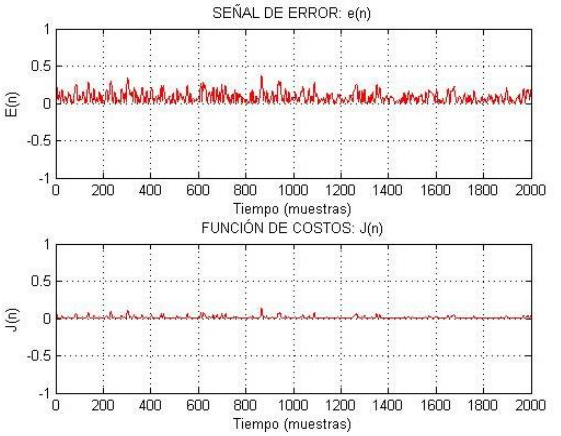
Promedio de la función de costos =0.0076
Promedio del error absoluto =0.0617
Coeficiente de correlación=0.8476
SSNR d,x=4.0961[dBc]

Promedio del error= 3.2742e-04
Tiempo de ejecución=1196.266560 seg]

SSNR y',x=1.3052[dBc]

Prueba 23: M=512, alfa=0.06, ep=0.000000000000008, ga=0.1



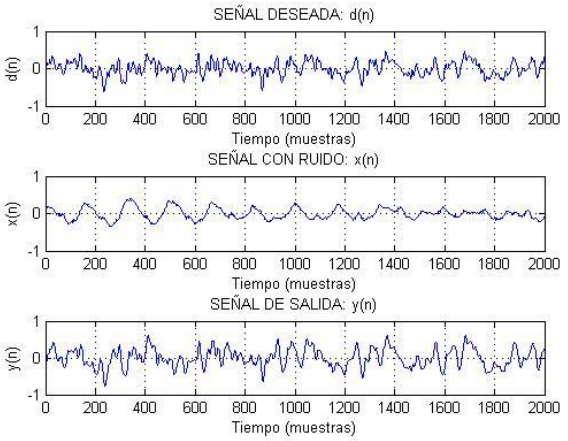


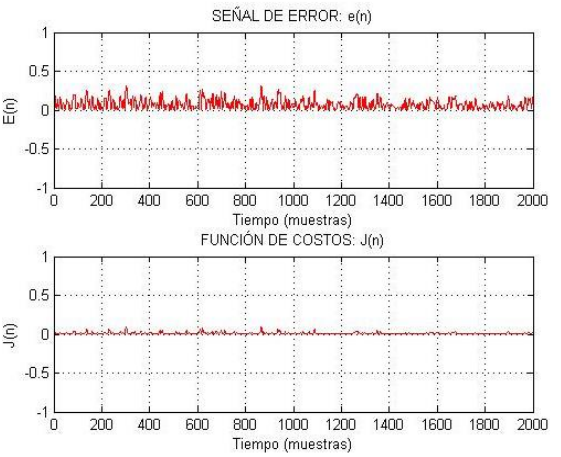
Promedio de la función de costos =0.0059
Promedio del error absoluto =0.0547
Coeficiente de correlación=0.8844
SSNR d,x=4.0961[dBc]

Promedio del error= 2.6475e-04
Tiempo de ejecución=1155.080497 [seg]

SSNR y',x=1.7610[dBc]

Prueba 24: M=512, alfa=0.08, ep=0.000000000000000008, ga=0.1





Promedio de la función de costos =0.0045
Promedio del error absoluto =0.0476
Coeficiente de correlación=0.9149
SSNR d,x=4.0961[dBc]

Promedio del error= 1.8431e-04
Tiempo de ejecución=1617.317947 [seg]

SSNR y',x=2.1731[dBc]

Señal 2:

Para la señal 2 se parte de los valores óptimos de las pruebas con la primera señal y considerando las conclusiones obtenidas. (Anexo B)

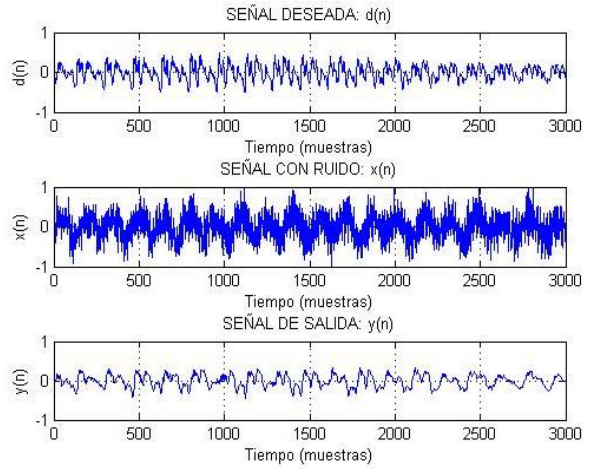
- alfa=0.08, si el valor aumenta mejora el rendimiento
- ep=0.000000000000000008, si el valor disminuye mejora el rendimiento
- ga=0.1, si el valor aumenta mejora el rendimiento
- Mediante el sonido se deberá analizar en qué valor del coeficiente de correlación la reverberación no distorsiona la señal
- El crecimiento del orden del filtro mejora el coeficiente de correlación, por ende el rendimiento del filtro.

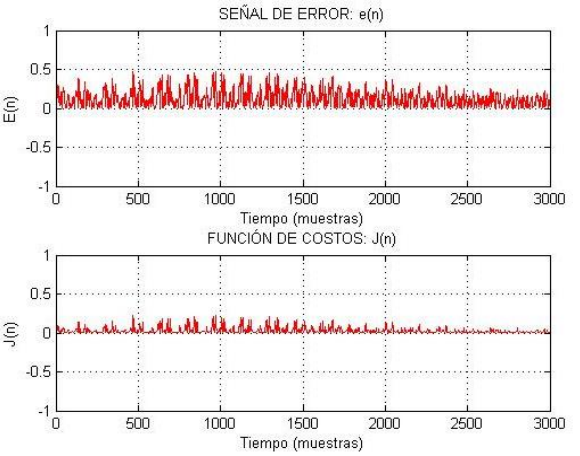
También se debe considerar que para esta segunda señal lo que se pretende reducir es el ruido ambiental y mantener la voz, por tal motivo se hará una primera prueba con los valores óptimos de la señal 1 y se irán cambiando hasta que la señal esté prácticamente limpia.

Para esta señal se realizaron pruebas con dos señales con ruido, una donde la voz es bastante entendible y con ruido de trasfondo y la segunda donde el ruido hace casi imposible escuchar la voz.

SEÑAL CON VOZ Y RUIDO DE TRASFONDO

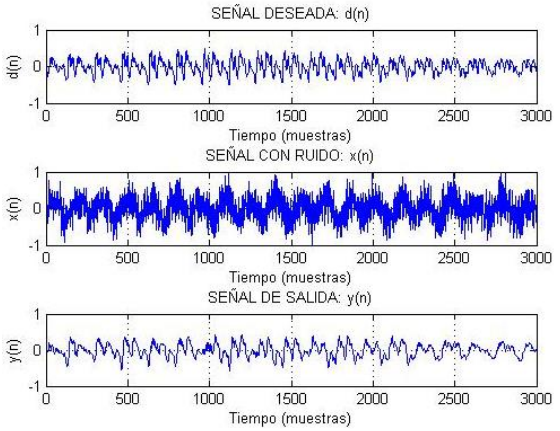
Prueba 1: M=64, alfa=0.08, ep=0.000000000000000008, ga=0.1

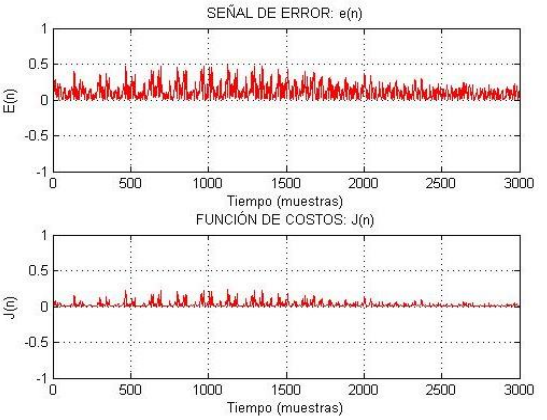




Promedio de la función de costos =0.0048
Promedio del error absoluto =0.0335
SNR d,x=-8.4075[dBc] SNR y',x=-15.6241[dBc]
Promedio del error =-4.0550e-05
Coeficiente de correlación=0.4238
Tiempo de ejecución=113.845025 [seg]

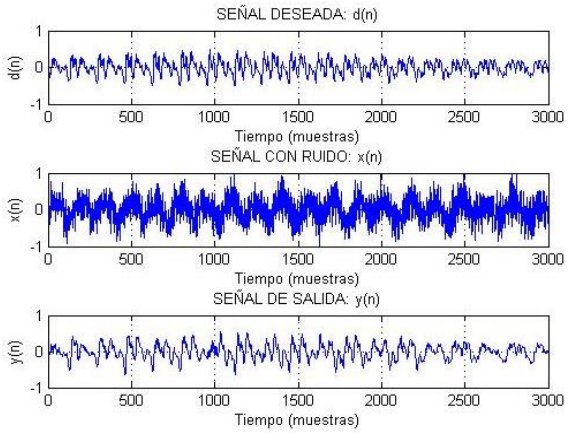
Prueba 2: M=64, alfa=0.13, ep=0.0000000000000000008, ga=0.1

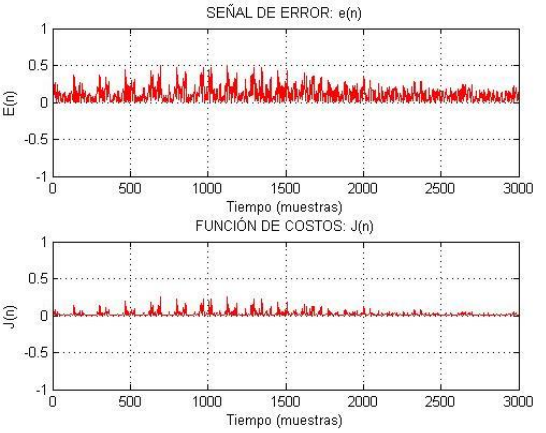




Promedio de la función de costos= 0.0045
Promedio del error absoluto = 0.0319
SNR d,x=-8.4075[dBc] SNR y',x=-14.3870[dBc]
Promedio del error =-2.9868e-05
Coeficiente de correlación=0.4858
Tiempo de ejecución=117.088688 [seg]

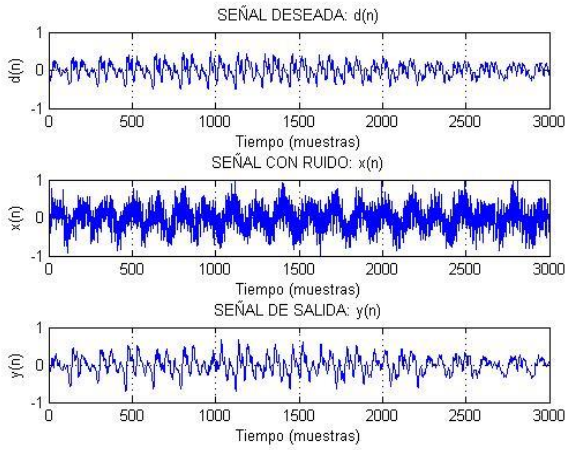
Prueba 3: M=64, alfa=0.2, ep=0.00000000000000000008, ga=0.1

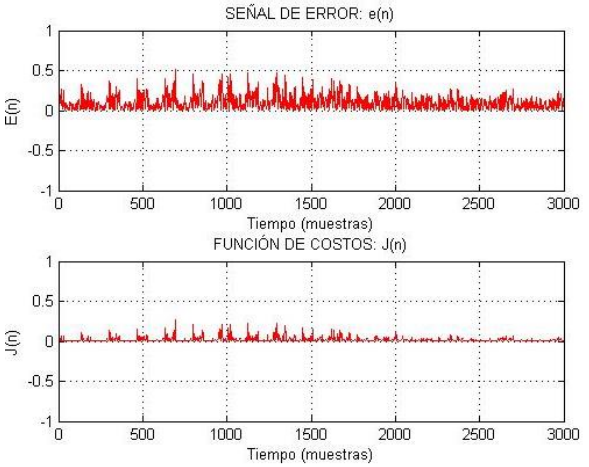




Promedio de la función de costos =0.0042
Promedio del error absoluto =0.0303
SNR d,x=-8.4075[dBc] SNR y',x=-13.4111[dBc]
Promedio del error =-1.6242e-05
Coeficiente de correlación=0.5372
Tiempo de ejecución=116.215116[seg]

Prueba 4: M=64, alfa=0.3, ep=0.000000000000000000000005, ga=0.1

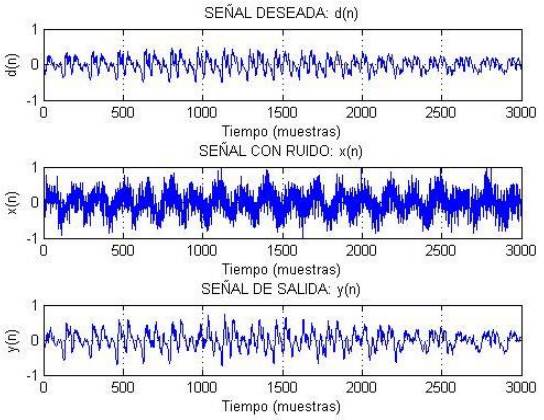


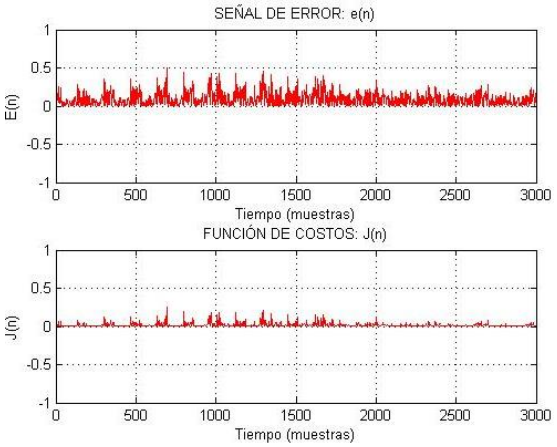


Promedio de la función de costos=0.0039
Promedio del error absoluto = 0.0287
SNR d,x=-8.4075[dBc] SNR y',x=-12.5922[dBc]

Promedio del error =4.2194e-05
Coeficiente de correlación=0.5771
Tiempo de ejecución=115.821612[seg]

Prueba 5: M=64, alfa=0.4, ep=0.000000000000000000000000005, ga=0.1

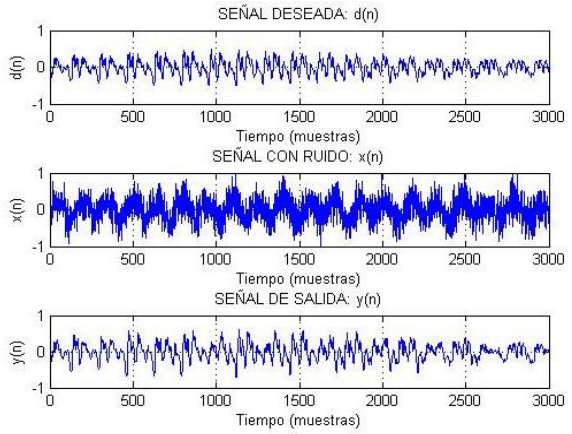


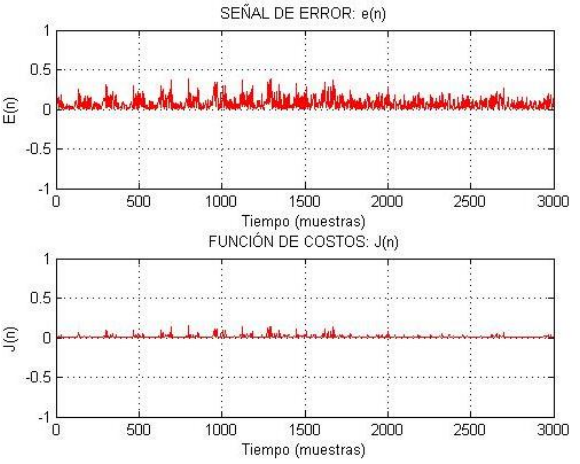


Promedio de la función de costos = 0.0038
Promedio del error absoluto =0.0277
SNR d,x=-8.4075[dBc] SNR y',x=-12.0521[dBc]

Promedio del error = 7.4789e-05
Coeficiente de correlación=0.5976
Tiempo de ejecución=115.156349 [seg]

Prueba 6: M=64, alfa=0.5, ep=0.000000000000000000000000005, ga=0.25

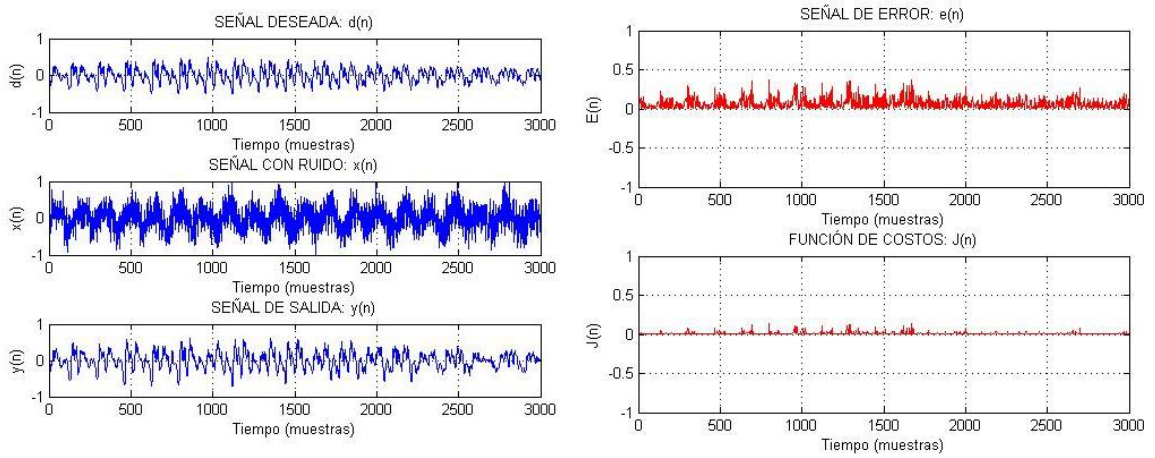




Promedio de la función de costos =0.0032
Promedio del error absoluto =0.0256
SNR d,x=-8.4075[dBc] SNR y',x=-13.299[dBc]

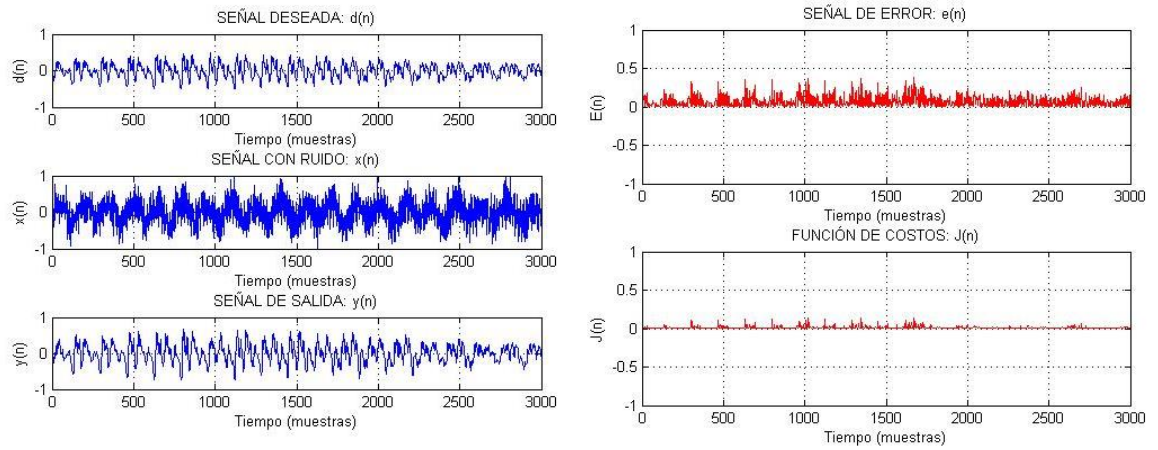
Promedio del error = -3.46e-05
Coeficiente de correlación=0.6898
Tiempo de ejecución=116.175869 [seg]

Prueba 7: M=64, alfa=0.6, ep=0.00000000000000000000000005, ga=0.25



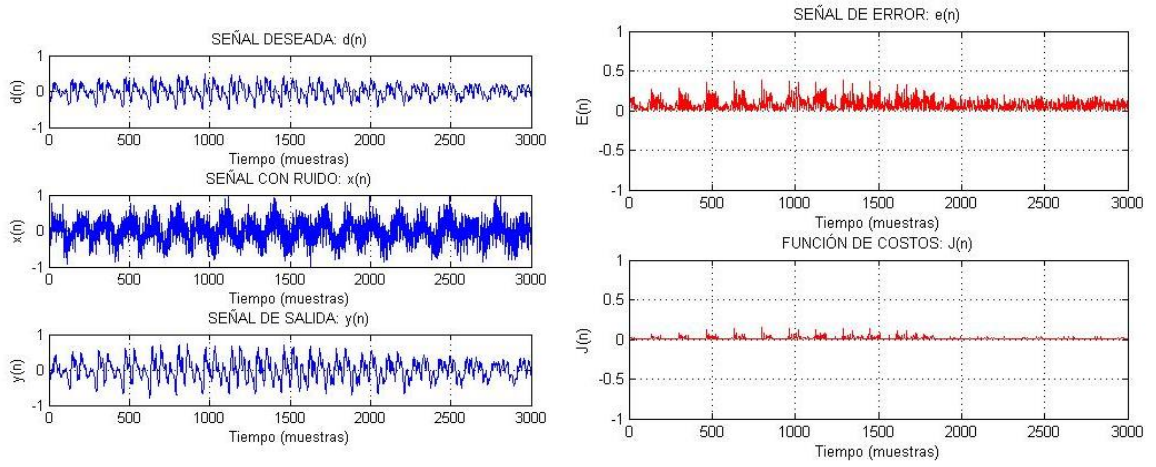
Promedio de la función de costos= 0.003
Promedio del error absoluto =0.0247
SNR d,x=-8.4075[dBc] SNR y',x=-13.0377[dBc]
Promedio del error =-3.6331e-05
Coeficiente de correlación=0.7128
Tiempo de ejecución=117.019808 [seg]

Prueba 8: M=128, alfa=0.6, ep=0.00000000000000000000000005, ga=0.25



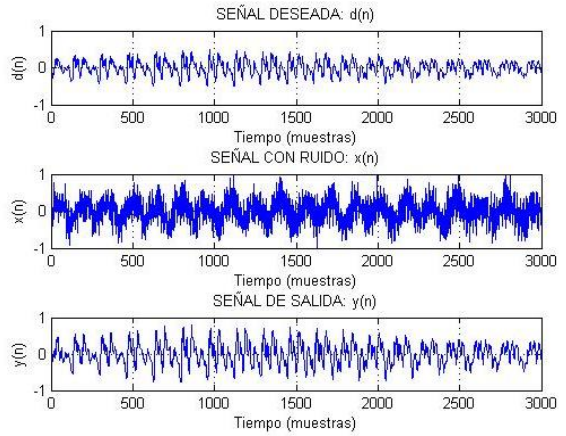
Promedio de la función de costos =0.0029
Promedio del error absoluto =0.0236
SNR d,x=-8.4075[dBc] SNR y',x=-11.9598[dBc]
Promedio del error =7.3618e-06
Coeficiente de correlación=0.7173
Tiempo de ejecución=166.994262 [seg]

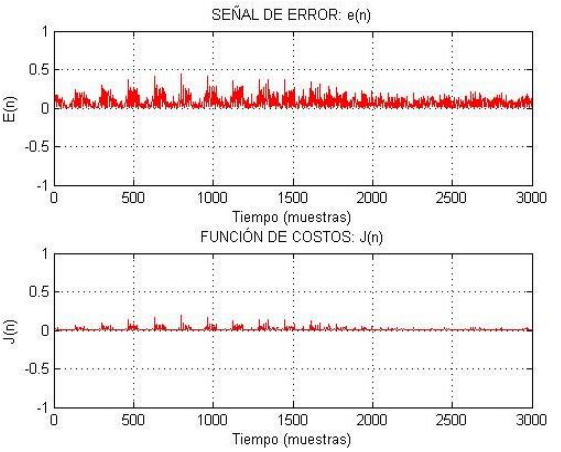
Prueba 9: M=256, alfa=0.6, ep=0.00000000000000000000000005, ga=0.25



Promedio de la función de costos = 0.003
Promedio del error absoluto =0.0234
SNR d,x=-8.4075[dBc] SNR y',x=-10.7577[dBc]
Promedio del error =-4.8443e-05
Coeficiente de correlación=0.6995
Tiempo de ejecución=272.357135 [seg]

Prueba 10: M=512, alfa=0.6, ep=0.0000000000000000000000000005, ga=0.25



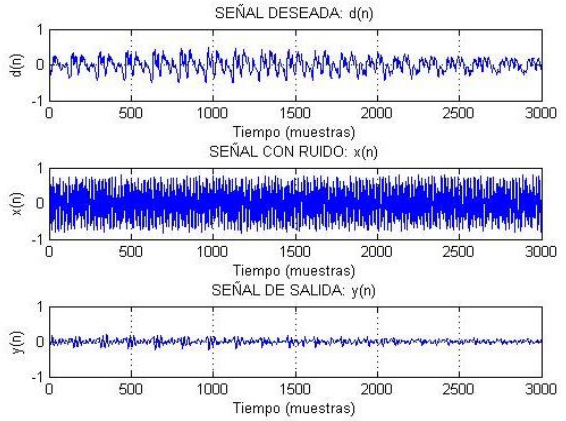


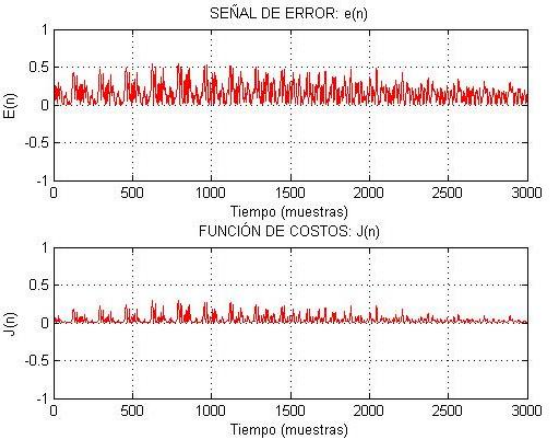
Promedio de la función de costos =0.0033
Promedio del error absoluto = 0.0238
SNR d,x=-8.4075[dBc] SNR y',x=-9.7838[dBc]

Promedio del error =1.5842e-06
Coeficiente de correlación=0.6812
Tiempo de ejecución=467.191561 [seg]

SEÑAL CON RUIDO Y VOZ DE TRASFONDO

Prueba 1: M=64, alfa=0.08, ep=0.000000000000000008, ga=0.1

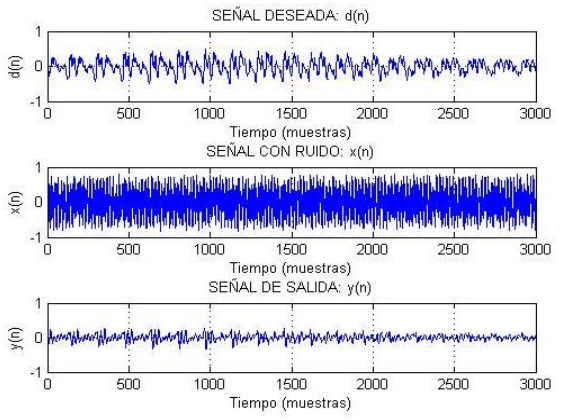


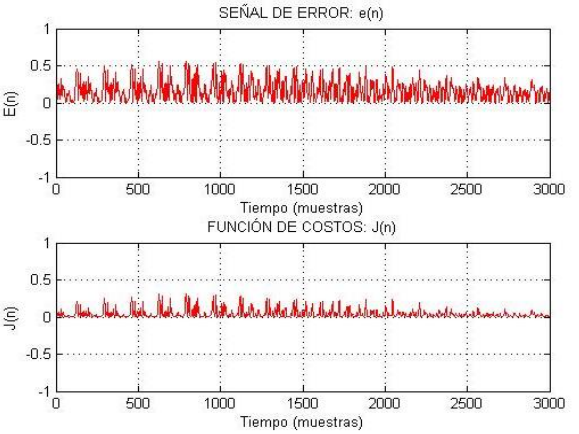


Promedio de la función de costos=0.0066
Promedio del error absoluto =0.0402
SSNR d,x=-9.3079[dBc] SSNR y',x=-23.3454[dBc]

Promedio del error =-7.3533e-05
Coeficiente de correlación=-0.1944
Tiempo de ejecución= 121.586903 [seg]

Prueba 2: M=64, alfa=0.13, ep=0.0000000000000000000008, ga=0.1

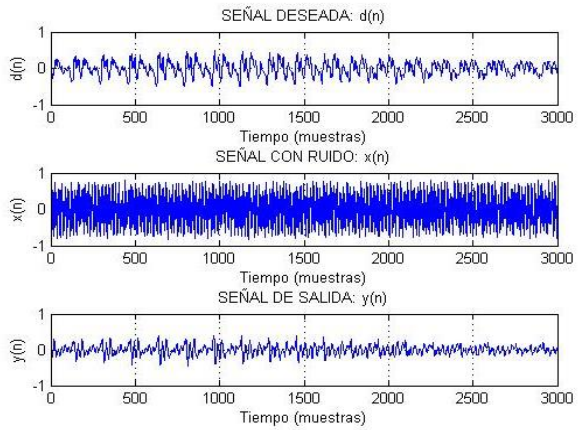


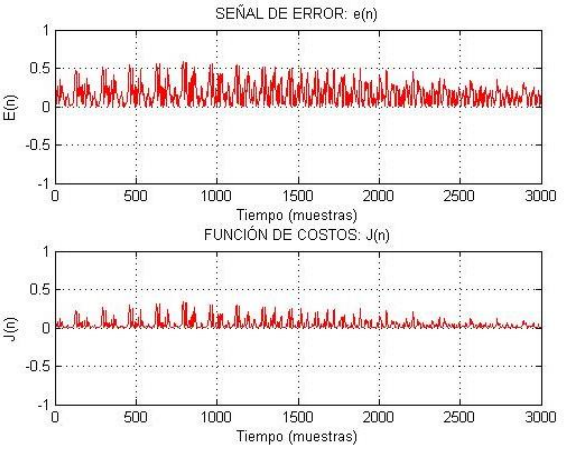


Promedio de la función de costos =0.0071
Promedio del error absoluto =0.0417
SSNR d,x=-9.3079[dBc] SSNR y',x=-20.7283[dBc]

Promedio del error =-7.7648e-05
Coeficiente de correlación=-0.2371
Tiempo de ejecución=115.185604[seg]

Prueba 3: M=64, alfa=0.2, ep=0.00000000000000000008, ga=0.1

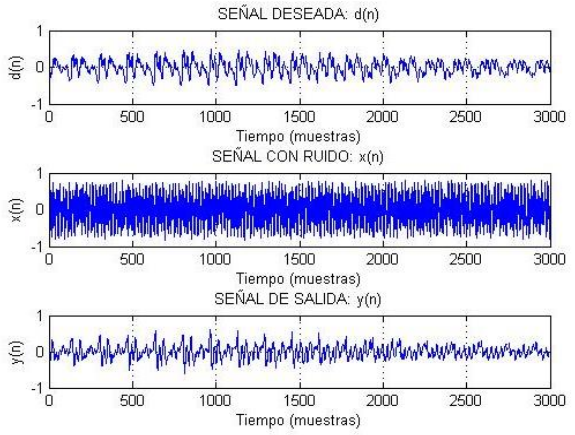


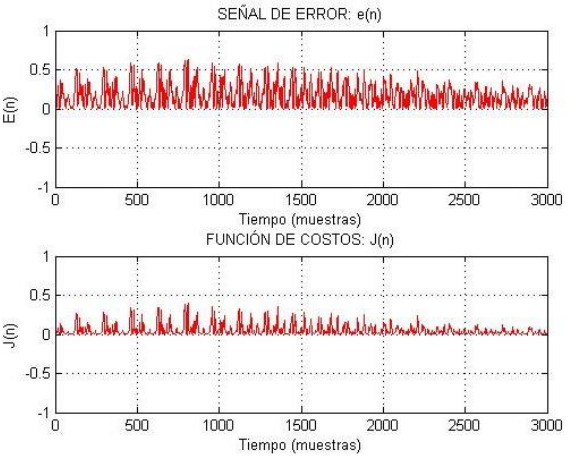


Promedio de la función de costos=0.0078
Promedio del error absoluto =0.0441
SSNR d,x=-9.3079[dBc] SSNR y',x=-18.249[dBc]

Promedio del error =-8.6125e-05
Coeficiente de correlación=-0.2899
Tiempo de ejecución=118.814954[seg]

Prueba 4: M=64, alfa=0.3, ep=0.000000000000000000000005, ga=0.1

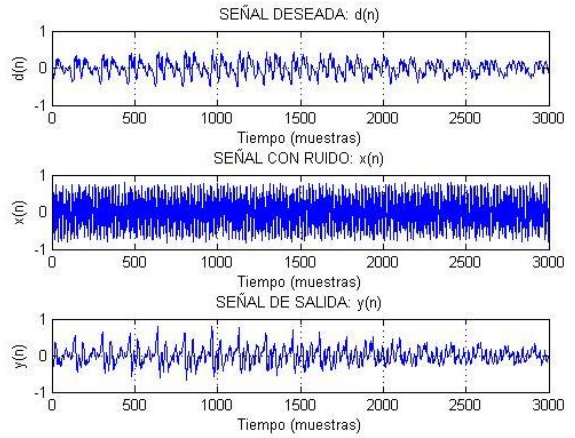


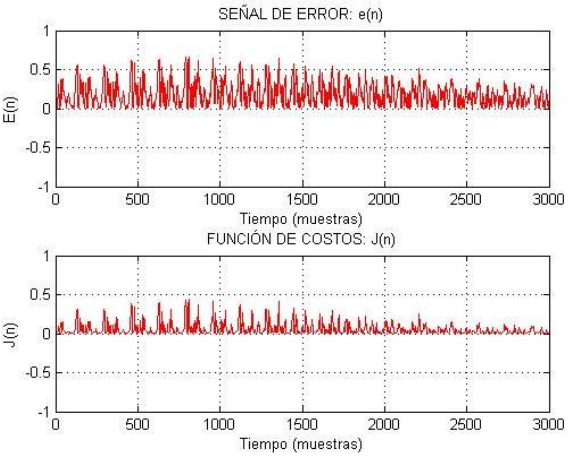


Promedio de la función de costos = 0.0093
Promedio del error absoluto =0.0481
SSNR d,x=-9.3079[dBc] SSNR y',x=-15.5899[dBc]

Promedio del error =-1.0446e-04
Coeficiente de correlación=-0.3511
Tiempo de ejecución=119.192764 [seg]

Prueba 5: M=64, alfa=0.4, ep=0.0000000000000000000000000005, ga=0.1

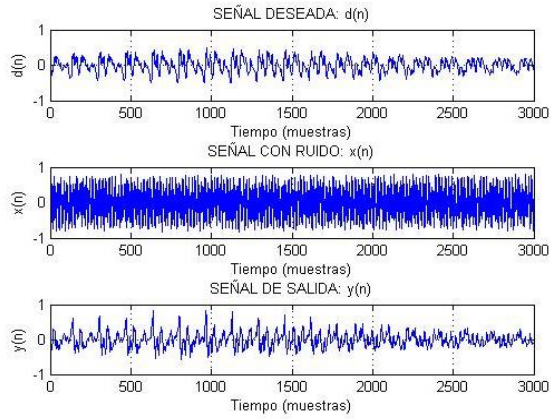


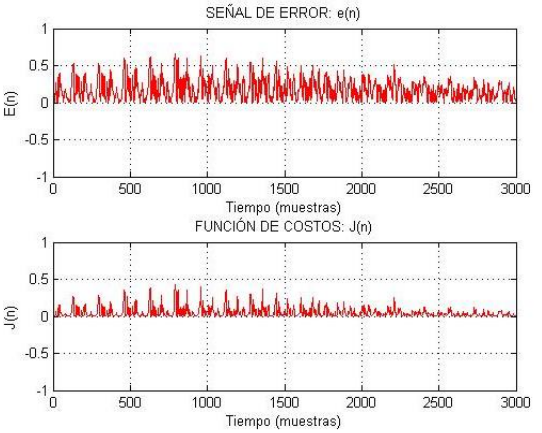


Promedio de la función de costos=0.0111
Promedio del error absoluto =0.0527
SSNR d,x=-9.3079[dBc] SSNR y',x=-13.3428[dBc]

Promedio del error =-1.3072e-04
Coeficiente de correlación=-0.3903
Tiempo de ejecución=118.164595[seg]

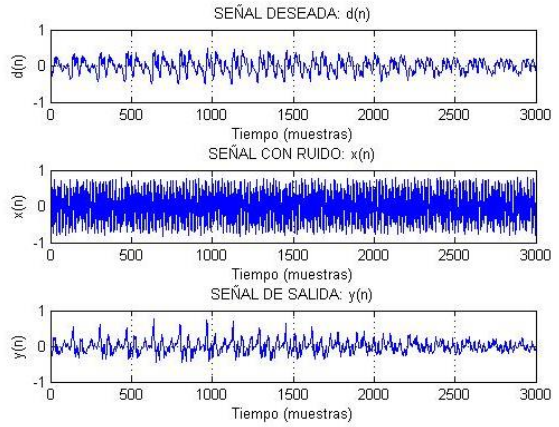
Prueba 6: M=64, alfa=0.5, ep=0.000000000000000000000000005, ga=0.25

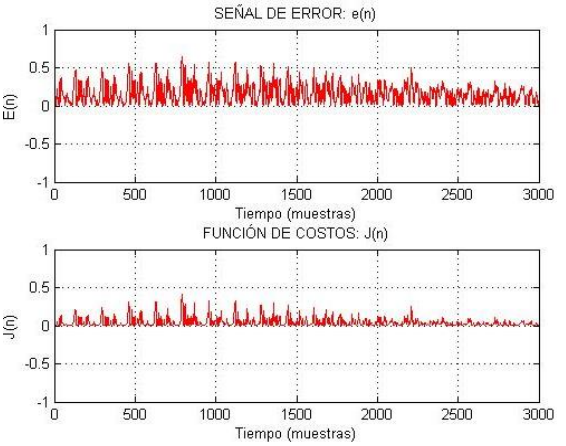




Promedio de la función de costos =0.0110 Promedio del error =-1.5074e-04
Promedio del error absoluto =0.0526 Coeficiente de correlación=-0.3824
SSNR d,x=-9.3079[dBc] SSNR y',x=-13.3289[dBc] Tiempo de ejecución=116.808807 [seg]

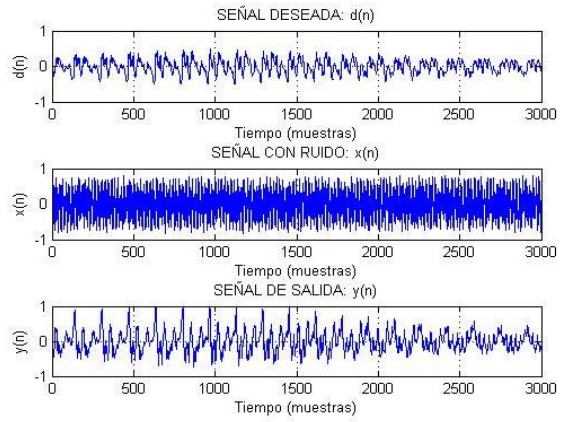
Prueba 7: M=64, alfa=0.6, ep= 0.000000000000000000000000005, ga=0.4

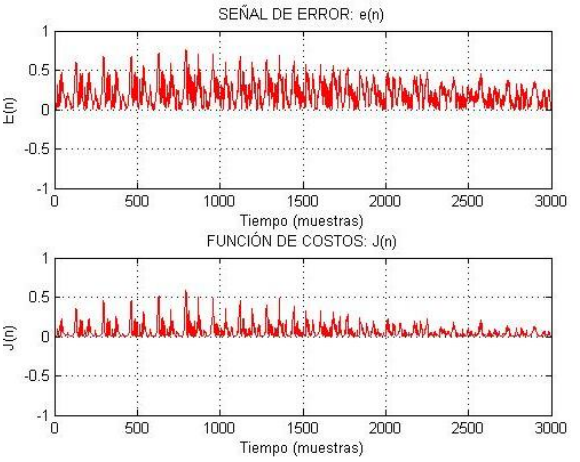




Promedio de la función de costos =0.0095 Promedio del error =-1.5249e-04
Promedio del error absoluto =0.0489 Coeficiente de correlación=-0.3134
SSNR d,x=-9.3079[dBc] SSNR y',x=-14.7403[dBc] Tiempo de ejecución=113.014030 [seg]

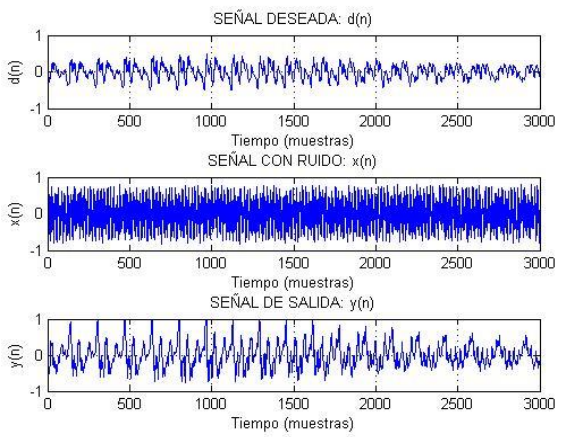
Prueba 8: M=128, alfa=0.6, ep= 0.000000000000000000000000005, ga=0.2

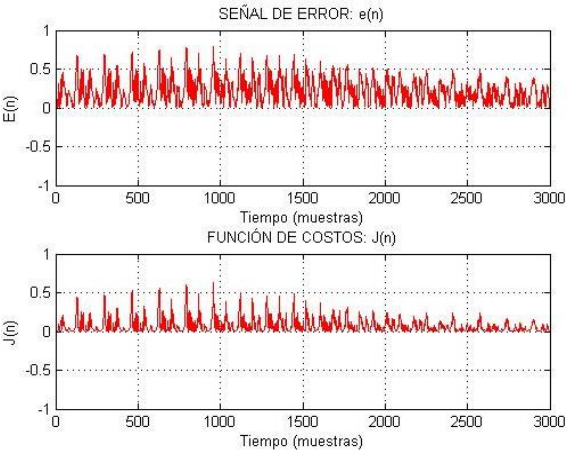




Promedio de la función de costos =0.0163 Promedio del error =-1.9849e-04
Promedio del error absoluto = 0.0641 Coeficiente de correlación=-0.4070
SSNR d,x=-9.3079[dBc] SSNR y',x=-9.4545[dBc] Tiempo de ejecución=170.119605 [seg]

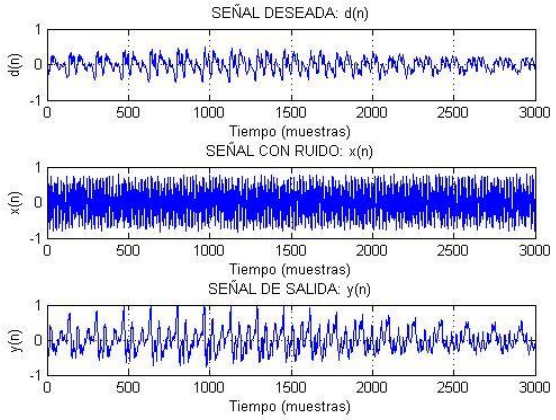
Prueba 9: M=256, alfa=0.6, ep= 0.00000000000000000000000005, ga=0.2

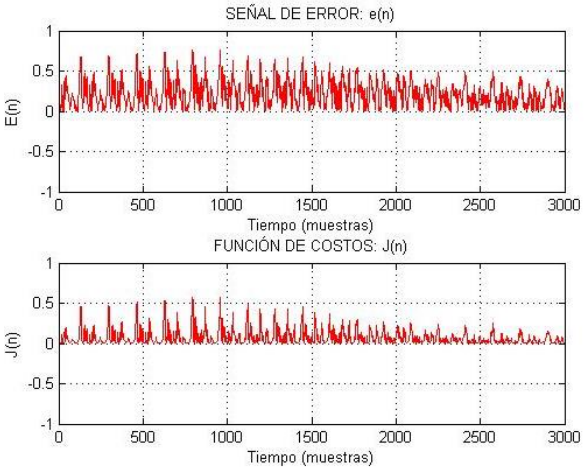




Promedio de la función de costos =0.0182 Promedio del error = -1.4506e-4
Promedio del error absoluto =0.0678 Coeficiente de correlación=-0.4104
SSNR d,x=-9.3079[dBc] SSNR y',x=-8.5238[dBc] Tiempo de ejecución=270.206725 [seg]

Prueba 10: M=512, alfa=0.512, ep= 0.00000000000000000000000005, ga=0.1





Promedio de la función de costos = 0.0159 Promedio del error = -1.2779e-04
Promedio del error absoluto =0.0632 Coeficiente de correlación=-0.4063
SSNR d,x=-9.3079[dBc] SSNR y',x=-9.6467[dBc] Tiempo de ejecución=491.970990[seg]

Señal 3:

Para la tercera señal solo se cuenta con la señal de la conversación con ruido, al código utilizado hasta este momento se le agregan cambios para poder detectar la señal y la deseada se va calculando iteración a iteración.

Los valores de los parámetros que se utilizarán para probar el filtro son:

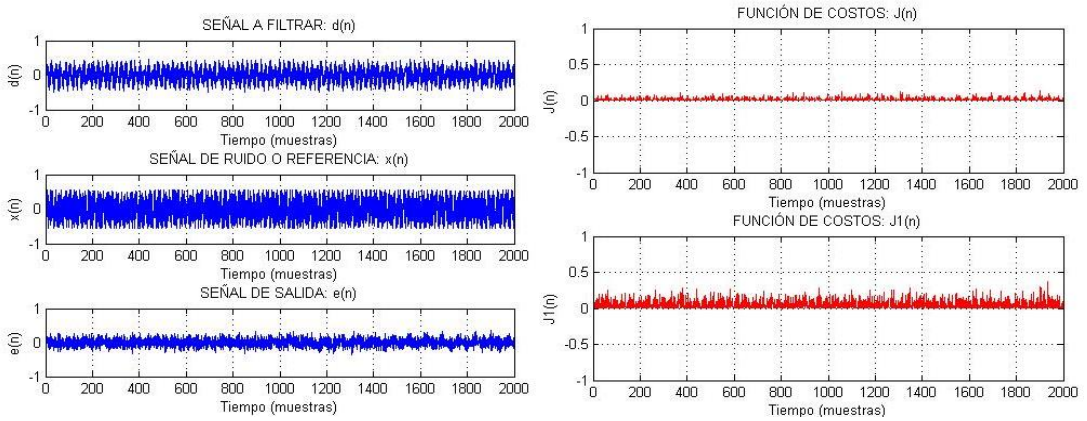
- M=256,alfa=0.6,ep=0.00000000000000000000000008,ga=0.25
- M=256,alfa=0.6,ep= 0.00000000000000000000000008,ga=0.2
- M=256,alfa=0.512,ep=0.00000000000000000000000008,ga=0.1

Para aplicar el filtro se necesitarán dos señales, la señal de voz con ruido y otra señal que contiene el ruido puro, siendo esta la señal de referencia para el filtro y como señal a filtrar se utilizará la señal con ruido y voz de trasfondo, siendo la que requiere de mejor adaptación para dejar la voz clara.

Del programa de MATLAB para la señal 3 (Anexo C), la señal *d* será la señal a filtrar mientras que la señal *x* será el ruido de referencia para el algoritmo, y ahora la señal que va a contener la voz será la señal *e,* que aunque aparentemente es la señal de error es la que se obtendrá quitando el ruido de la señal a filtrar. Así mismo, se obtiene el promedio de la señal *e* y *y* para observar que *e* debe disminuir, mientras que *y* aumentará y de igual manera el J correspondiente a cada una de ellas. La correlación se hará entre la señal a filtrar y la señal que contiene la voz y se debe apreciar que entre mejor se filtre la correlación disminuirá y eso nos ayudará a saber cuál prueba dio mejores resultados.

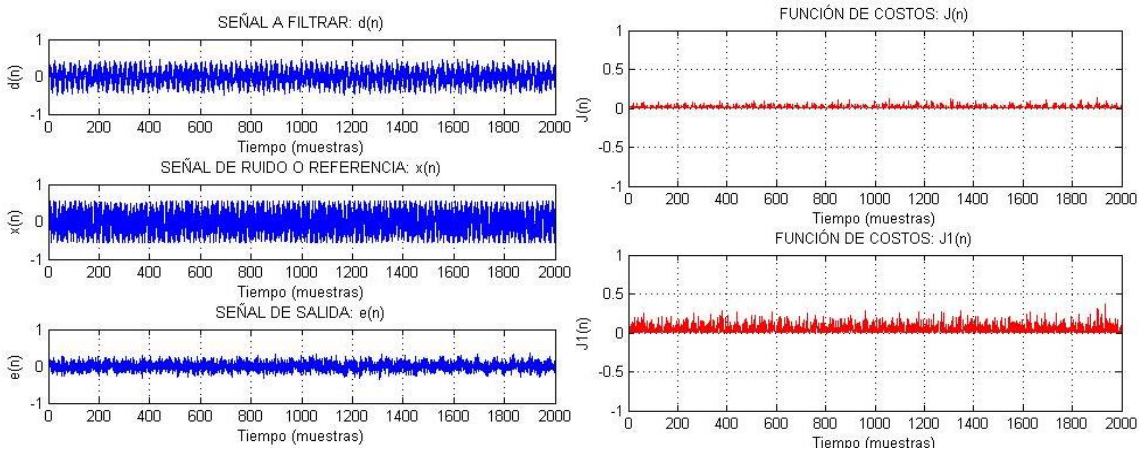
Se realizarán las primeras tres pruebas con los mejores valores de la señal pasada y hasta ese punto se observará lo obtenido para saber si los valores se cambiarán y cómo o si se queda alguna de esas opciones.

Prueba 1: M=256, alfa=0.6, ep=0.00000000000000000000000008, ga=0.25



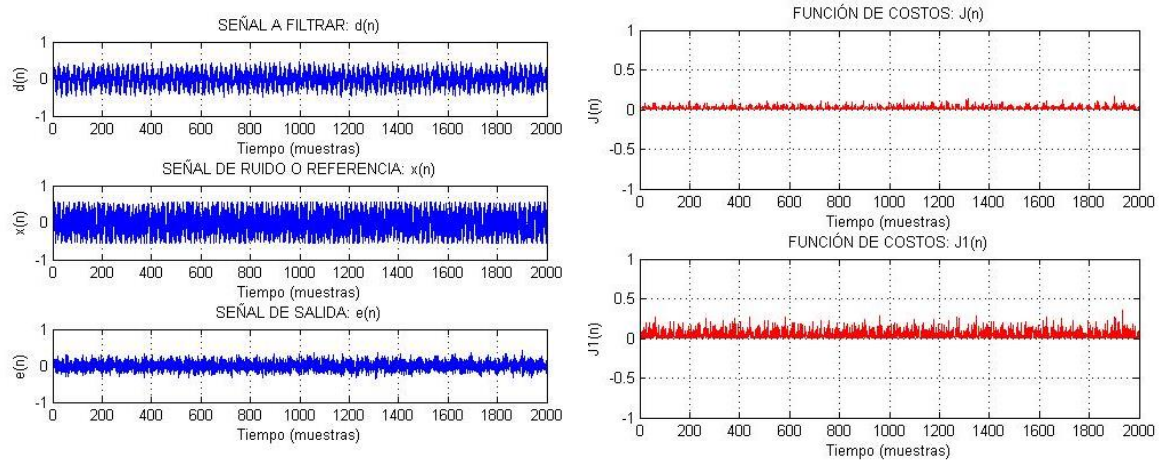
Promedio de la función de costos señal de salida o ruido =0.0495
Promedio de la función de costos señal deseada o error = 0.0162
Promedio de la señal deseada absoluta=0.1064
Promedio de la señal de salida o ruido absoluto = 0.1844
Coeficiente de correlación=0.2961 Tiempo de ejecución= 230.740432 [seg]
SNR x=7.2048 [dBc] SNR d=1.965[dBc] SNR e'=2.678 [dBc]

Prueba 2: M=256, alfa=0.6, ep= 0.00000000000000000000000008, ga=0.2



Promedio de la función de costos señal de salida o ruido =0.0501
Promedio de la función de costos señal deseada o error = 0.0162
Promedio de la señal deseada absoluta = 0.1062
Promedio de la señal de salida o ruido absoluto = 0.1854
Coeficiente de correlación=0.2894 Tiempo de ejecución= 235.642759 [seg]
SNR x=7.2048 [dBc] SNR d=1.965[dBc] SNR e'=2.6694 [dBc]

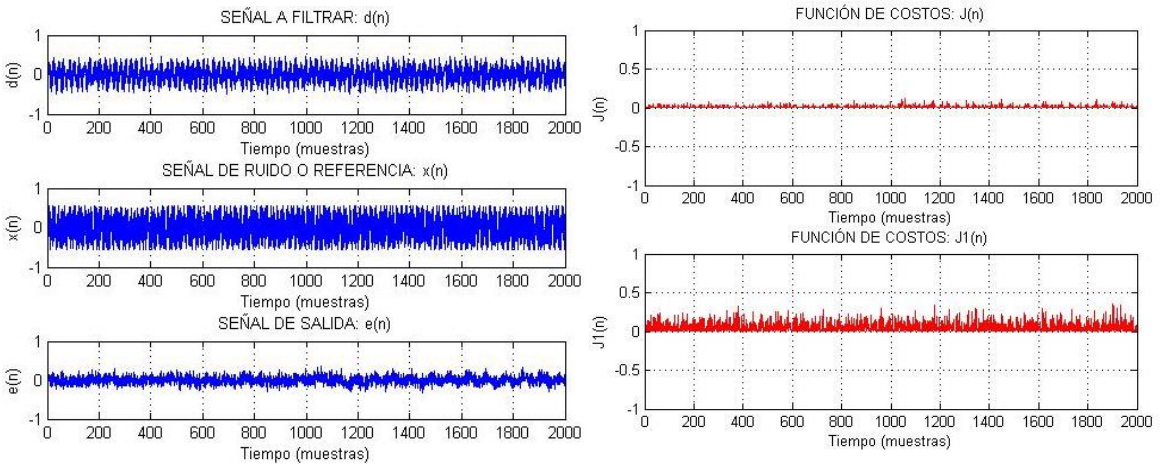
Prueba 3: M=256, alfa=0.512,ep=0.00000000000000000000000008, ga=0.1



Promedio de la función de costos señal de salida o ruido=0.0488
Promedio de la función de costos señal deseada o error = 0.0197
Promedio de la señal deseada absoluta = 0.118
Promedio de la señal de salida o ruido absoluto = 0.1830
Coeficiente de correlación=0.3342 Tiempo de ejecución= 230.315027 [seg]
SNR x=7.2048 [dBc] SNR d=1.965[dBc] SNR e'=3.4446 [dBc]

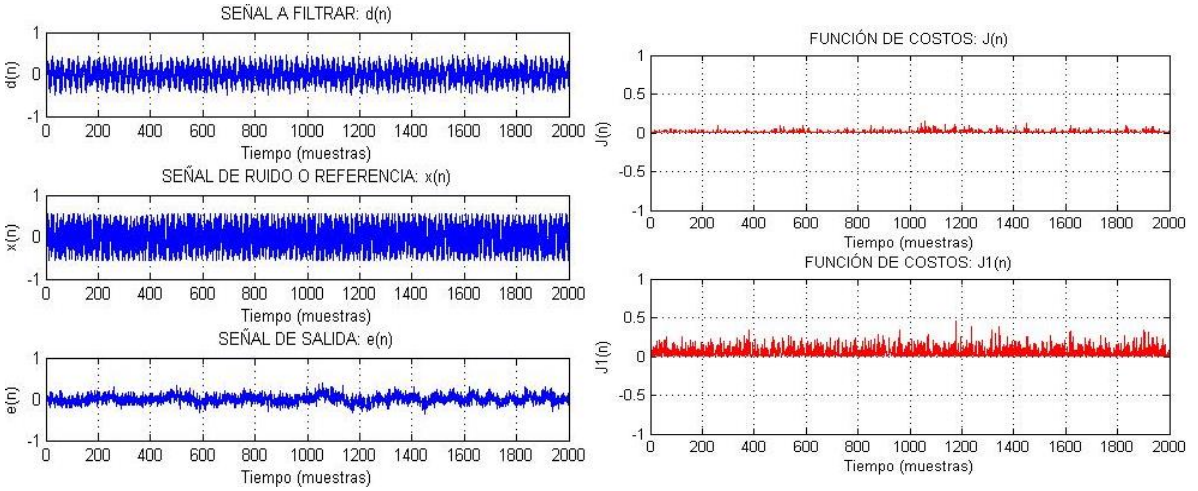
Hasta la prueba número 3 se puede observar que las prueba que arrojó mejores resultados tomando en cuenta que su correlación es más baja y su SNR de e' igual es más bajo, es la número 2 donde el valor de alfa y ga son un poco más grandes. Para las siguiente pruebas se irá subiendo el valor de alfa primero para ver cuánto mejora y si es necesario también se cambiará el valor de ep y ga, se realizarán las pruebas con el orden del filtro de 256 y cuando se obtenga el mejor valor se llevará a 512.

Prueba 4: M=256, alfa=0.8, ep= 0.00000000000000000000000008, ga=0.2



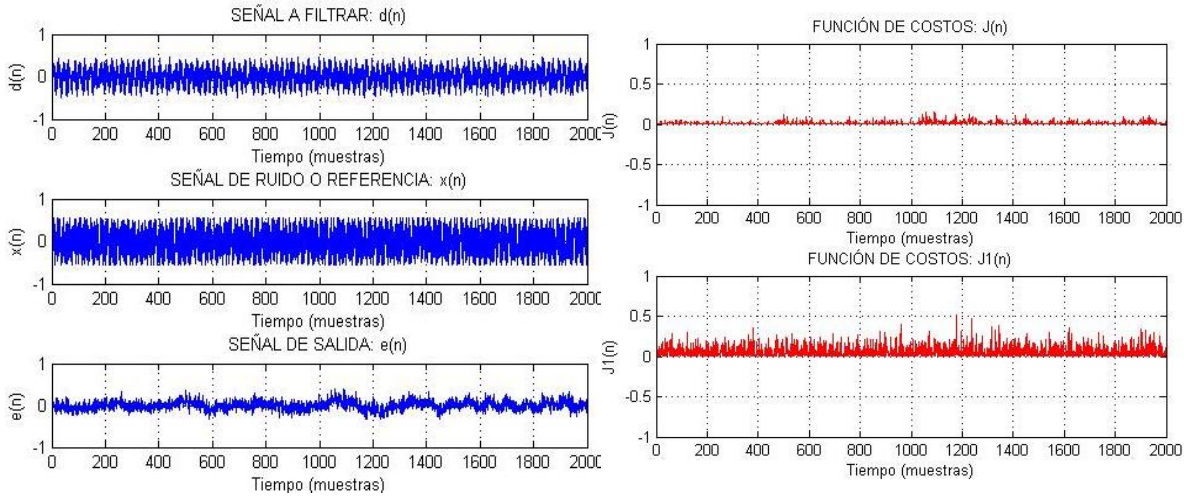
Promedio de la función de costos señal de salida o ruido=0.0536
Promedio de la función de costos señal deseada o error = 0.0114
Promedio de la señal deseada absoluta = 0.0875
Promedio de la señal de salida o ruido absoluto = 0.191
Coeficiente de correlación=0.1663 Tiempo de ejecución= 230.849567 [seg]
SNR x=7.2048 [dBc] SNR d=1.965[dBc] SNR e'=0.6867 [dBc]

Prueba 5: M=256, alfa=0.95, ep= 0.00000000000000000000000008, ga=0.2



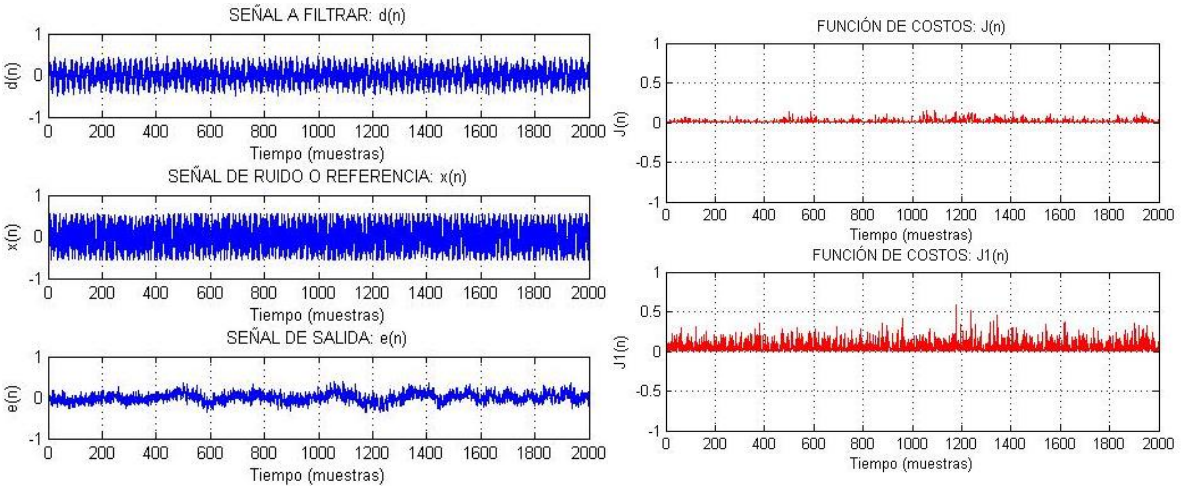
Promedio de la función de costos señal de salida o ruido=0.056
Promedio de la función de costos señal deseada o error = 0.0106
Promedio de la señal deseada absoluta = 0.0826
Promedio de la señal de salida o ruido absoluto = 0.1954
Coeficiente de correlación=0.102 Tiempo de ejecución= 237.473442 [seg]
SNR x=7.2048 [dBc] SNR d=1.965[dBc] SNR e'=-1.8875 [dBc]

Prueba 6: M=256, alfa=0.99, ep= 0.00000000000000000000000008, ga=0.2



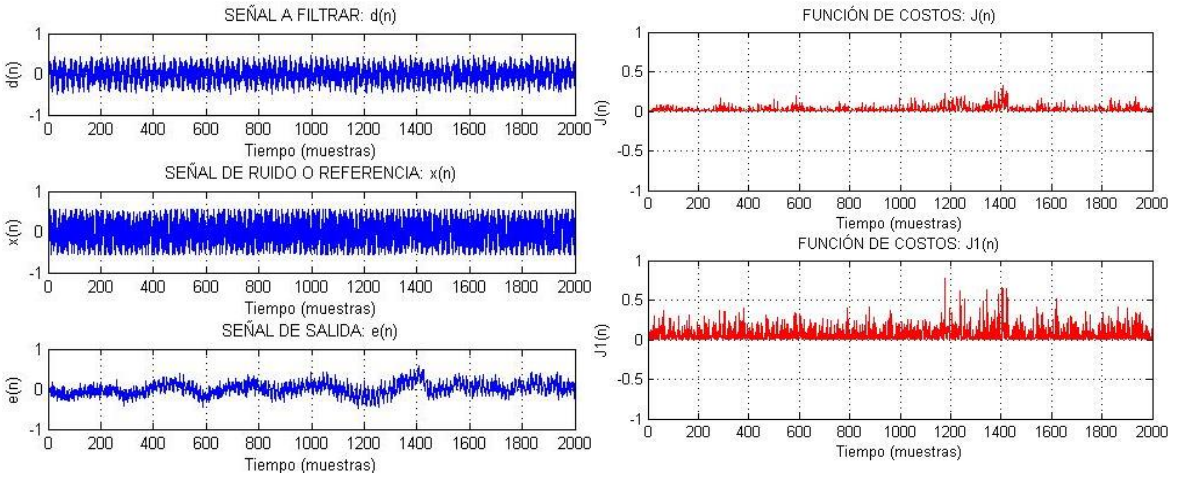
Promedio de la función de costos señal de salida o ruido=0.0577
Promedio de la función de costos señal deseada o error = 0.0116
Promedio de la señal deseada absoluta = 0.0859
Promedio de la señal de salida o ruido absoluto = 0.1984
Coeficiente de correlación=0.084 Tiempo de ejecución= 242.869358 [seg]
SNR x=7.2048 [dBc] SNR d=1.965[dBc] SNR e'=-3.1929 [dBc]

Prueba 7: M=256, alfa=0.99,ep= 0.000000000000000000000000008, ga=0.1



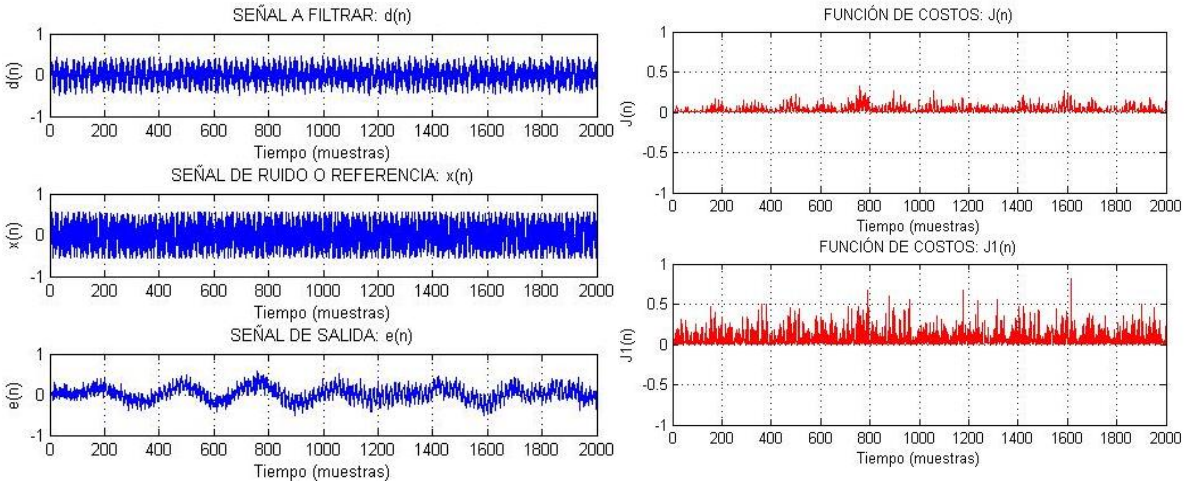
Promedio de la función de costos señal de salida o ruido=0.0601
Promedio de la función de costos señal deseada o error = 0.0127
Promedio de la señal deseada absoluta = 0.0891
Promedio de la señal de salida o ruido absoluto = 0.202
Coeficiente de correlación=0.0537 Tiempo de ejecución= 254.130508 [seg]
SNR x=7.2048 [dBc] SNR d=1.965[dBc] SNR e'=-4.2155 [dBc]

Prueba 8: M=256, alfa=0.99, ep= 0.000000000000000000000000008, ga=0.02



Promedio de la función de costos señal de salida o ruido=0.0674
Promedio de la función de costos señal deseada o error = 0.0185
Promedio de la señal deseada absoluta = 0.1045
Promedio de la señal de salida o ruido absoluto = 0.2121
Coeficiente de correlación=0.0216 Tiempo de ejecución= 248.738759 [seg]
SNR x=7.2048 [dBc] SNR d=1.965[dBc] SNR e'=-7.0766 [dBc]

Prueba 9: M=512, alfa=0.99, ep= 0.00000000000000000000000000008, ga=0.02



Promedio de la función de costos señal de salida o ruido=0.0681
Promedio de la función de costos señal deseada o error = 0.0191
Promedio de la señal deseada absoluta = 0.1048
Promedio de la señal de salida o ruido absoluto = 0.2129
Coeficiente de correlación=0.0175 Tiempo de ejecución= 428.718052 [seg]
SNR x=7.2048 [dBc] SNR d=1.965[dBc] SNR e'=-7.6745 [dBc]

4.3.3 Conversión D/A

Señal 1: Con ayuda de MATLAB se manda a guardar y reproducir la señal de salida del filtro con la misma frecuencia de muestreo que la señal de entrada del filtro

$$fs = 44100\ Hz$$

Señal 2: De igual manera que para la señal 1, se manda a reproducir y guardar la salida del filtro con la frecuencia de muestreo de entrada

$$fs = 44100\ Hz$$

Señal 3: La señal 3 se reproduce y guarda con la frecuencia de muestreo de la señal de entrada

$$fs = 44100\ Hz$$

4.4 Resultados preliminares

Los resultados obtenidos en MATLAB dan una pauta para saber cómo se comporta el algoritmo, cómo influye la selección de parámetros, las señales a filtrar, entre otros aspectos; por lo cual es necesario evaluar lo obtenido hasta el momento y poder llegar a conclusiones útiles para la implementación en tiempo real.

Con el fin de facilitar la visualización y comparación de los resultados obtenidos para cada prueba, los valores se almacenan en tablas donde se puede observar cómo van cambiando a cada prueba.

SEÑAL 1

No. Prueba	Promedio de la función de costos	Promedio del error	Promedio del error absoluto	Tiempo de ejecución [seg]	Coeficiente de correlación	SSNR d,x [dBc]	SSNR y',x [dBc]
1	0.0176	1.4410e-04	0.0948	235.888329	0.5647	4.0961	-2.6968
2	0.0174	1.2693e-04	0.0942	229.223665	0.5735	4.0961	-2.5737
3	0.0203	4.3705e-04	0.1017	226.761399	0.4564	4.0961	-3.9858
4	0.0201	4.2648e-04	0.1013	224.005704	0.4632	4.0961	-3.8408
5	0.0233	4.7174e-04	0.1092	229.52828	0.2887	4.0961	-7.1214
6	0.0233	4.9400e-04	0.1091	233.853612	0.2912	4.0961	-6.9466
7	0.0166	6.4000e-04	0.0919	345.387290	0.5994	4.0961	-1.8927
8	0.0164	6.3612e-04	0.0916	341.003143	0.6036	4.0961	-1.8395
9	0.01892	7.1019e-04	0.0990	356.918351	0.4982	4.0961	-3.1412
10	0.0216	4.4821e-04	0.1045	374.230469	0.3947	4.0961	-5.2857
11	0.0236	2.3461e-04	0.1096	379.387097	0.2719	4.0961	-8.3316
12	0.0226	4.3630e-04	0.1073	374.995640	0.3351	4.0961	-5.9455
13	0.0154	2.0714e-04	0.0882	639.511216	0.6368	4.0961	-1.3045
14	0.0153	2.0636e-04	0.0881	659.819352	0.6387	4.0961	-1.2812
15	0.0181	3.0088e-04	0.0956	632.641323	0.5431	4.0961	-2.4717
16	0.0180	3.0119e-04	0.0955	644.665609	0.5447	4.0961	-2.4465
17	0.0219	2.6953e-04	0.1052	671.321931	0.3731	4.0961	-5.0055
18	0.0219	2.6973e-04	0.1052	643.870748	0.374	4.0961	-4.9747
19	0.0144	3.2007e-04	0.0849	1143.011429	0.6658	4.0961	-0.7394
20	0.0144	3.2032e-04	0.0848	1167.943230	0.6667	4.0961	-0.7291
21	0.0101	3.6562e-04	0.0712	1205.074933	0.7863	4.0961	0.6003
22	0.0076	3.2742e-04	0.0617	1196.266560	0.8476	4.0961	1.3052
23	0.0059	2.6475e-04	0.0547	1155.080497	0.8844	4.0961	1.7610
24	0.0045	1.8431e-04	0.0476	1617.317947	0.9149	4.0961	2.1731

Para realizar el análisis de las muestras obtenidas se presentan gráficas evaluando cada uno de los parámetros obtenidos y sacando conclusiones de cada una de ellas. Cabe mencionar que para las primeras 12 pruebas se mantuvieron ambas señales con sus valores originales de amplitud, mientras que para las otras 12 pruebas se hizo un escalamiento de 1.7 a la señal con ruido para tratar de igualar su amplitud a la deseada y obtener un mejor valor en la toma del SNR.

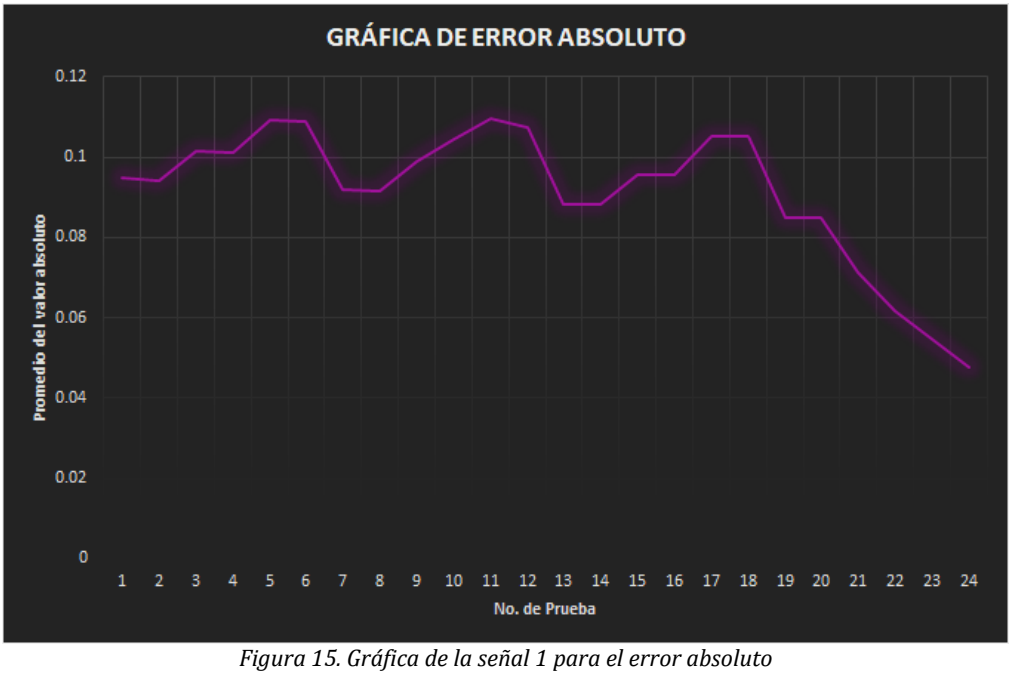


Figura 15. Gráfica de la señal 1 para el error absoluto

En la gráfica de promedio de error absoluto se puede observar que hasta la prueba número 20 los valores más pequeños de error se dieron cuando el valor de alfa era el mayor (0.015) y si ese valor disminuye el error aumenta, por lo cual podemos decir que si el valor de alfa aumenta mejora el rendimiento del filtro; lo anterior se pudo comprobar aumentando el valor de alfa en las últimas cuatro pruebas. Así mismo en los puntos más bajos se ve una ligera inclinación descendente de la prueba 1 a 2, de la 7 a 8, de la 13 a 14 y de la 19 a 20; esta ligera inclinación se da debido a que el valor de ep se hace más pequeño, por lo que otra conclusión es que si el valor de ep se hace más pequeño también mejora el rendimiento del filtro. Finalmente cabe mencionar que los puntos de error más bajos es donde se hizo el cambio del orden del filtro M, lo que permite comprobar que al aumentar el orden mejora el rendimiento.

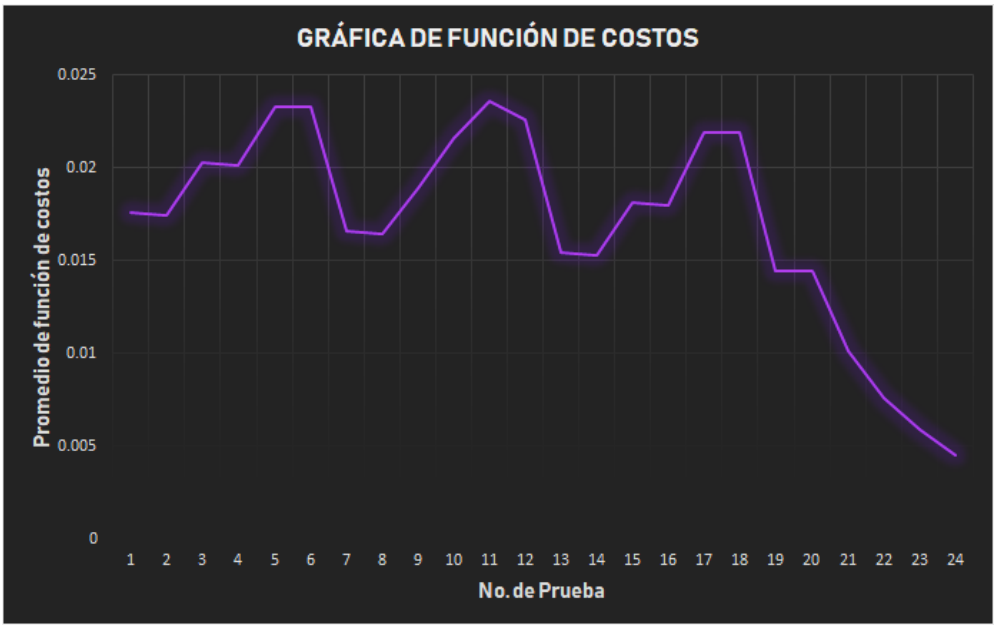


Figura 16. Gráfica de la señal 1 para la función de costos (J)

La gráfica de función de costos refleja prácticamente lo mismo que la gráfica del promedio de error absoluto, de tal manera que comprueba las observaciones hechas anteriormente y permite apreciar de un poco más detalladamente los cambios descendentes y los cambios entre el orden del filtro.

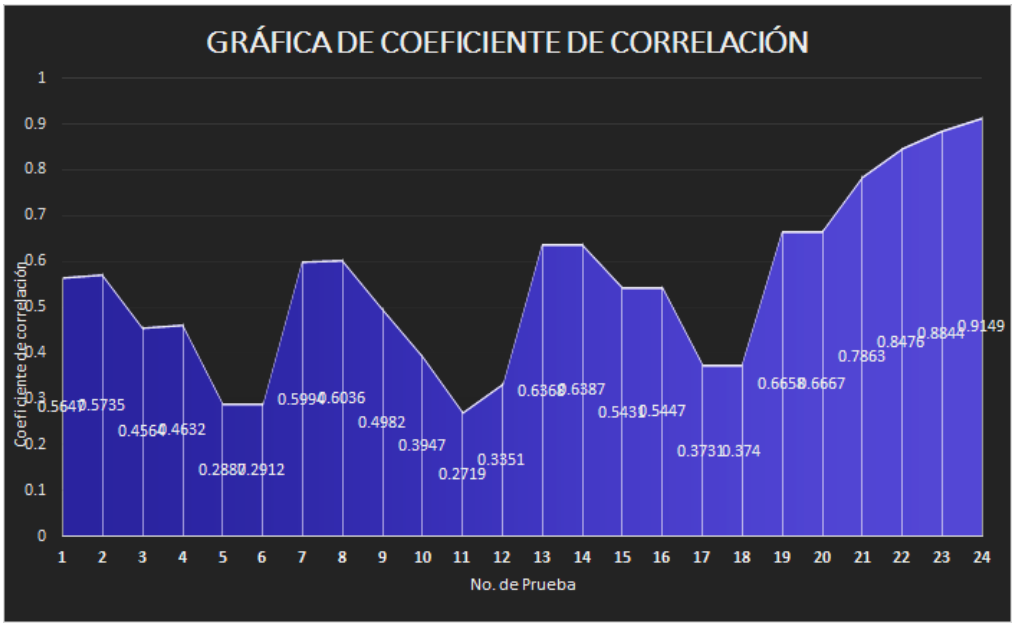


Figura 17. Gráfica de la señal 1 de coeficiente de correlación entre la señal deseada y la salida del filtro.

De todos los parámetros a evaluar uno de los más importantes es el coeficiente de correlación, porque nos arroja la información de similitud entre las señales de entrada y deseada, siendo el valor unitario el valor máximo y que refleja que las señales son exactamente iguales. Las pruebas realizadas reflejaron que las pruebas con mayor coeficiente de correlación fueron las que tenían el valor más alto de alfa y el valor más bajo de ep, y así mismo fue mejorando el coeficiente de correlación con el cambio del orden de filtro. Con el aumento del coeficiente de correlación y escuchando la señal de salida del filtro, se pudo apreciar que entre mayor era el coeficiente del filtro la reverberación que se daba entre las señales se redujo considerablemente hasta que en la última prueba era prácticamente nula y también se pudo observar que antes del coeficiente de 0.6 la reverberación distorsionaba mucho la señal de salida del filtro. Las últimas 3 pruebas fueron las mejores ya que arrojaron valores del coeficiente de correlación arriba de 0.8, llegando a obtener un coeficiente de 0.9149, un valor cercano el unitario.

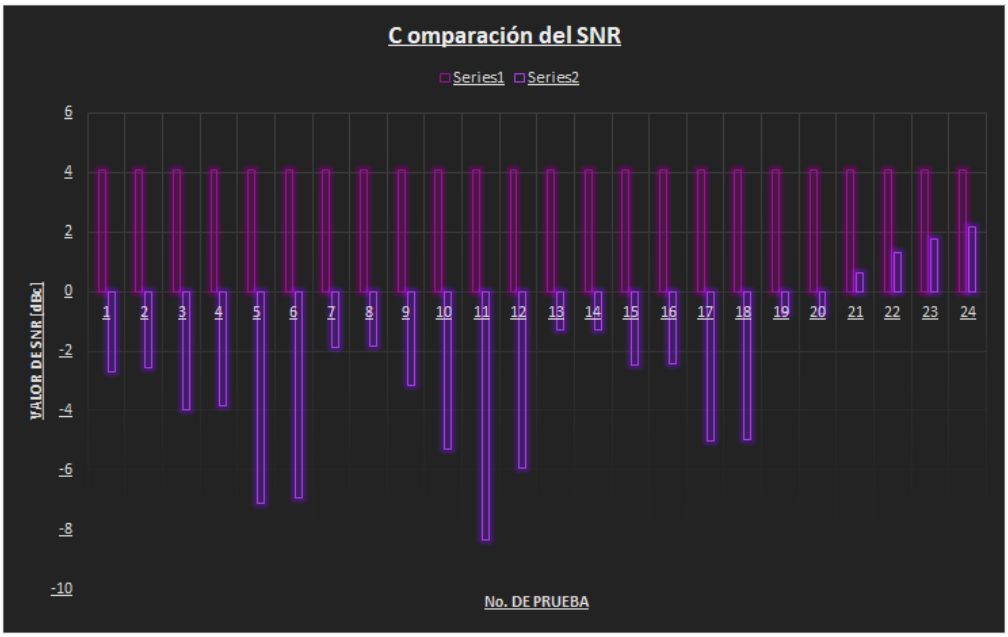


Figura 18. Gráfica de la señal 1 del valor del SNR

El SNR es otro de los parámetros más importantes para analizar el rendimiento del filtro. Hasta la prueba 12 podemos observar que el valor de SNR d,x es un poco mayor a 8 y el valor de SNR y',x debe de igualarse al primero para que el rendimiento sea bueno; se puede observar que los valores en los que más se acercó fueron las pruebas 7 y 8. Posteriormente, para las siguientes 12 pruebas se hizo el escalamiento de la señal con ruido y se pudo observar el cambio en el valor del SNR d,x; en estas últimas 12 pruebas, los mejores valores se obtuvieron en las últimas 4 pruebas, sobre todo en la prueba número 24, confirmando el valor de correlación obtenido previamente.

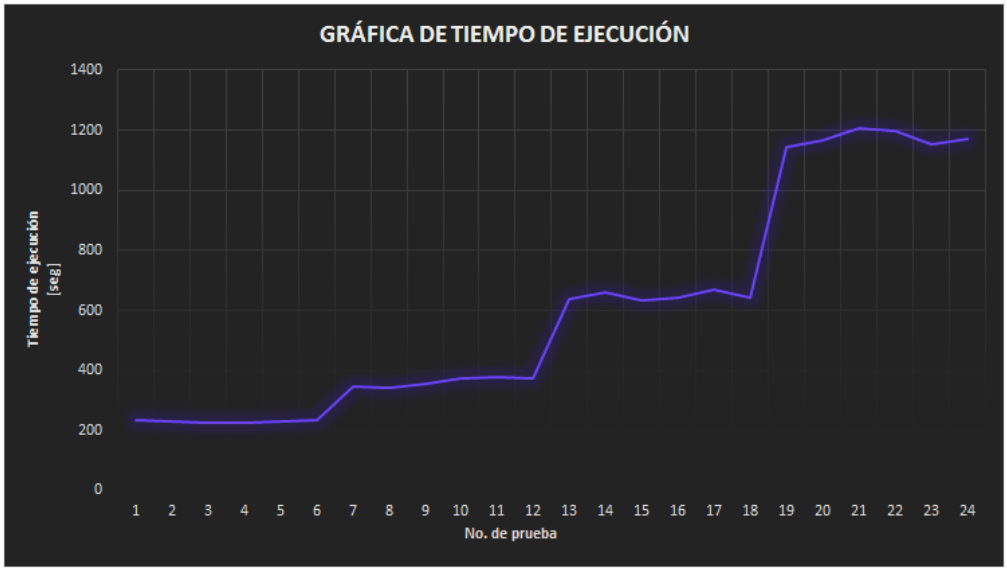


Figura 19. Gráfica de la señal 1 de tiempo de ejecución.

Para la señal 1 si bien el tiempo de ejecución es importante es el parámetro al que se le da menos peso pero deja apreciar que el tiempo de ejecución se mantiene rondando por valores muy cercanos mientras el orden del filtro no cambie y es hasta que cambia cuando el tiempo de ejecución se ve incrementado y el mayor incremento se ve en el cambio de M de 256 a 512. Si bien para la siguiente señal de tomará en cuenta el tiempo de ejecución, aún no será determinante para elegir la mejor combinación de parámetros para el filtro.

SEÑAL 2

Señal con voz y ruido de trasfondo

No. Prueba	Promedio de la función de costos	Promedio del error	Promedio del error absoluto	Coeficiente de correlación	Tiempo de ejecución [seg]	SSNR d,x [dBc]	SSNR y'x [dBc]
1	0.0048	-4.0550e-05	0.0335	0.4238	113.845025	-8.4075	-15.6241
2	0.0045	-2.9868e-05	0.0319	0.4858	117.088688	-8.4075	-14.3870
3	0.0042	-1.6242e-05	0.0303	0.5372	116.215116	-8.4075	-13.4111
4	0.0039	4.2194e-05	0.0287	0.5771	115.821612	-8.4075	-12.5922
5	0.0038	7.4789e-05	0.0277	0.5976	115.156349	-8.4075	-12.0521
6	0.0032	-3.46e-05	0.0256	0.6898	116.175869	-8.4075	-13.299
7	0.003	-3.6331e-05	0.0247	0.7128	117.019808	-8.4075	-13.0377
8	0.0029	7.3618e-06	0.0236	0.7173	166.994262	-8.4075	-11.9598
9	0.003	-4.8443e-05	0.0234	0.6995	272.357135	-8.4075	-10.7577
10	0.0033	1.5842e-06	0.0238	0.6812	467.191561	-8.4075	-9.7838

Hasta la prueba 7 se realizaron pruebas con el fin de mejorar el coeficiente de correlación y se logró, eso trajo consigo que la claridad de la voz mejorara y la distorsión se fuera reduciendo. También se debe tomar en cuenta que la señal a filtrar esta vez fue una señal de voz y se puede apreciar que los valores de los parámetros variaron conforme a las conclusiones obtenidas de la primera señal analizada y permite apreciar mejor los valores que deben tener los parámetros para la implementación en tiempo real. A continuación se grafican los valores de las últimas cuatro pruebas que son las que arrojan información concluyente para la siguiente señal a filtrar, donde el parámetro más importante será el orden del filtro

Figura 20. Gráfica de la señal 2 estado 1 de la función de costos

Se puede apreciar que para las primeras tres pruebas al duplicar el orden del filtro el promedio de función de costos se reduce o se mantiene, sin embargo para la última prueba aumenta un poco y las pruebas con el orden del filtro de 128 y 256 serían las mejores. Sin embargo, la distorsión en la voz pudo dejar de apreciarse hasta la prueba con el orden del filtro más grande: 512.

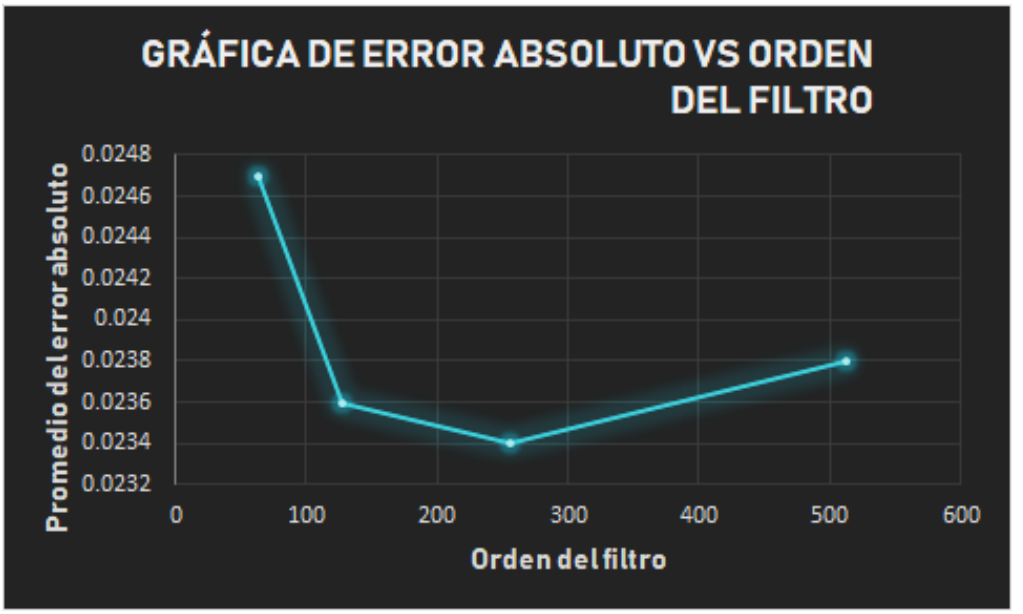


Figura 21. Gráfica de la señal 2 estado 1 del error absoluto

Siguiendo con los resultados de la gráfica anterior, vemos que las mejores tres pruebas fueron con los valores más grandes del orden del filtro. Los valores de los parámetros óptimos hasta este momento son los de la prueba 8, 9 y 10.

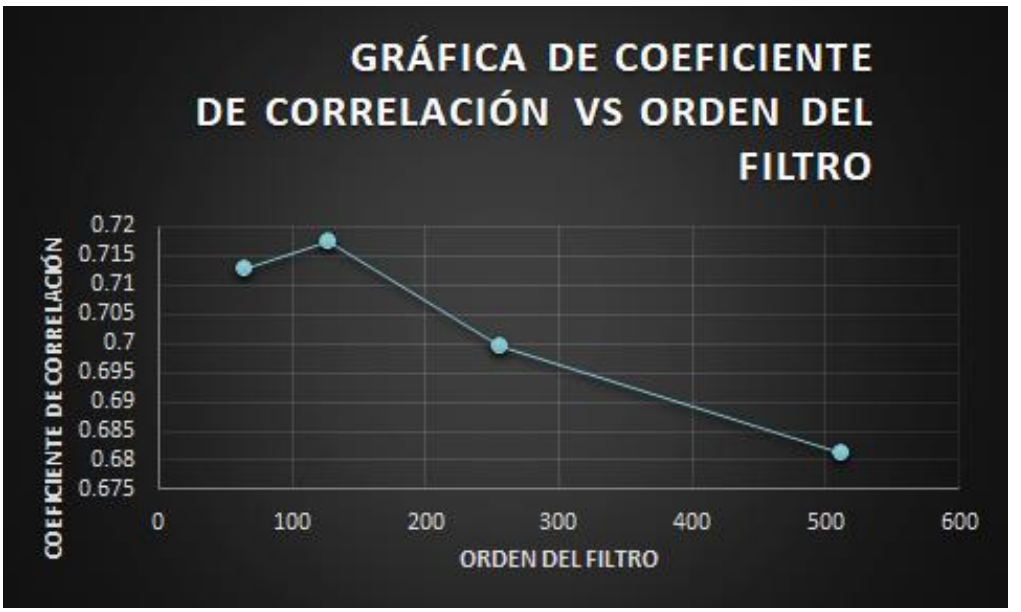


Figura 22. Gráfica de la señal 2 estado 1 del coeficiente de correlación

Se puede apreciar que los valores más altos de coeficiente de correlación son para los órdenes de filtro de 64 y 128. Aunque el coeficiente de correlación se redujo para 256 y 512, la calidad de la voz fue mucho mejor que con las dos primeras pruebas, se pudo apreciar que la distorsión se eliminó casi por completo.

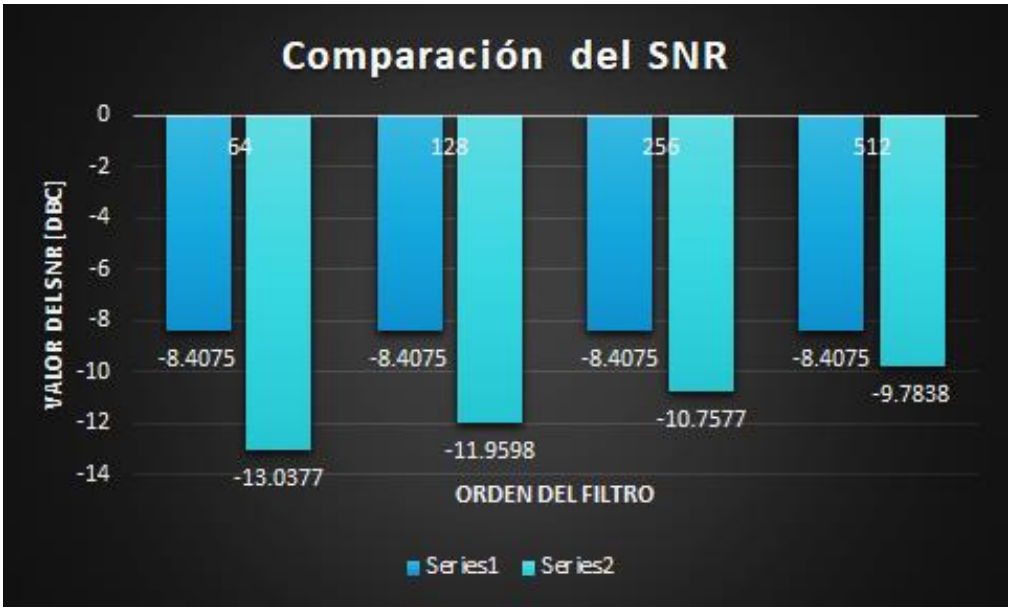


Figura 23. Gráfica de la señal 2 estado 1 de comparación del valor de SNR

En la gráfica del SNR se puede observar que los dos mejores valores obtenidos y los que más se acercan al SNR primario, son las pruebas con el orden del filtro de 256 y 512, lo cual es un factor muy importante a tomar en cuenta ya que el SNR refleja con mayor precisión el rendimiento y la fiabilidad del filtro.

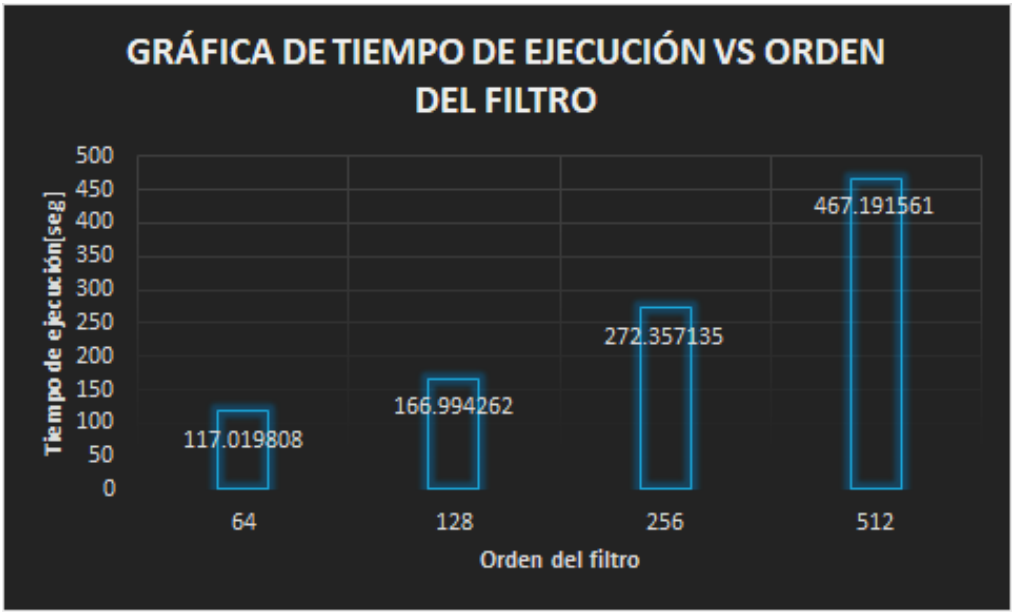


Figura 24. Gráfica de la señal 2 estado 1 del tiempo de ejecución del algoritmo

El tiempo de ejecución aumenta conforme aumenta el orden de filtro, es claro que el tiempo de ejecución es mayor en las últimas pruebas. Para la última prueba es importante saber esto pero el parámetro aún no será decisivo.

Conociendo todo lo anterior de las pruebas realizadas con esta señal se van a tomar los parámetros de las pruebas 9 y 10 como punto de partida para las pruebas de la siguiente señal junto con los parámetros de las pruebas que se seleccionen de la segunda señal ruidosa que se filtró.

Señal con ruido y voz de trasfondo

No. Prueba	Promedio de la función de costos	Promedio del error	Promedio del error absoluto	Coeficiente de correlación	Tiempo de ejecución [seg]	SSNR d,x [dBc]	SSNR y',x [dBc]
1	0.0066	-7.3533e-05	0.0402	-0.1944	121.586903	-9.3079	-23.3454
2	0.0071	-7.7648e-05	0.0417	-0.2371	115.185604	-9.3079	-20.7283
3	0.0078	-8.6125e-05	0.0441	-0.2899	118.814954	-9.3079	-18.249
4	0.0093	-1.0446e-04	0.0481	-0.3511	119.192764	-9.3079	-15.5899
5	0.0111	-1.3072e-04	0.0527	-0.3903	118.164595	-9.3079	-13.3428
6	0.0110	-1.5074e-04	0.0526	-0.3824	116.808807	-9.3079	-13.3289
7	0.0095	-1.5249e-04	0.0489	-0.3134	113.014030	-9.3079	-14.7403
8	0.0163	-1.9849e-04	0.0641	-0.4070	170.119605	-9.3079	-9.4545
9	0.0182	-1.4506e-4	0.0678	-0.4104	270.206725	-9.3079	-8.5238
10	0.0159	-1.2779e-04	0.0632	-0.4063	491.970990	-9.3079	-9.6467

Para la segunda señal filtrada se apreció que la distorsión no era tan grande como en la primera señal pero se podía seguir apreciando un ruido agudo proveniente del ruido a filtrar de la señal. También se apreció que el valor de la correlación era negativo, lo cual llega a suceder cuando las señales se correlacionan en sentido inverso, es decir, que a valores altos de una de ellas le suelen corresponder valor bajos de la otra y viceversa. Entre el valor sea más cercano a -1 la similitud de las señales es mayor y si se llega a -1 se habla de correlación negativa perfecta lo que supone igualdad absoluta entre las dos variables y aunado a esto se apreció que aunque

los valores de coeficiente de correlación no eran tan grandes, la calidad de la voz no era tan mala como con la primera señal a filtrar.

Se presentan las gráficas con los valores obtenidos de las últimas 4 pruebas, ya que hasta antes de ellas se buscó mejorar el coeficiente de correlación y es hasta la séptima prueba cuando se mantiene los parámetros y solo se varía el orden del filtro.

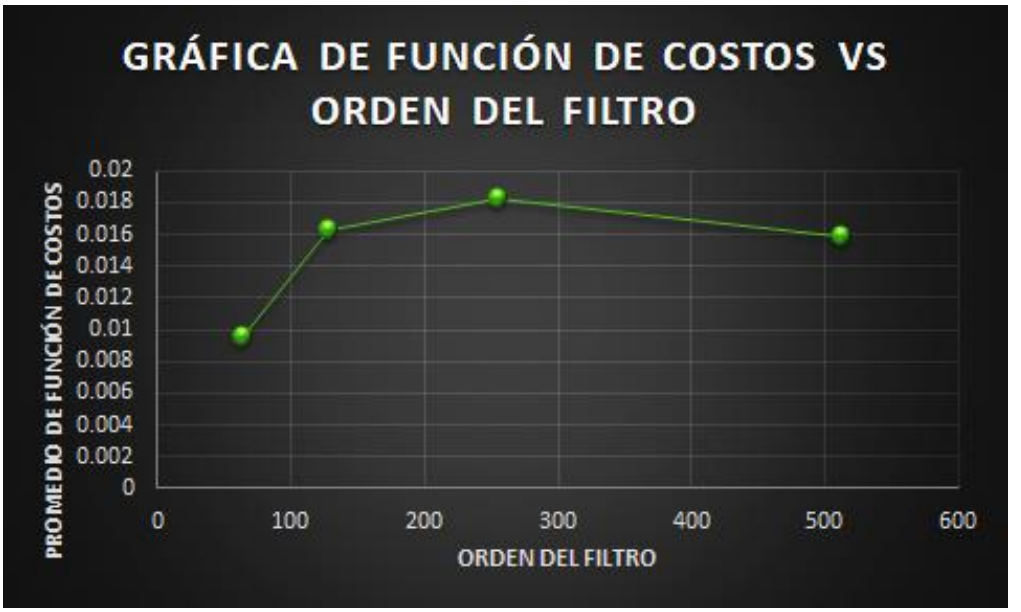


Figura 25. Gráfica de la señal 2 estado 2 de la función de costos

Figura 26. Gráfica de la señal 2 estado 2 del error absoluto.

Para esta señal se puede observar que el valor de la función de costos y del error absoluto aumentan cuando el orden del filtro aumenta exceptuando la última prueba, esto pasa por la correlación negativa que se da en el proceso, por tal motivo estos dos criterios no son válidos para evaluar el rendimiento que tuvo el filtro en estas pruebas.

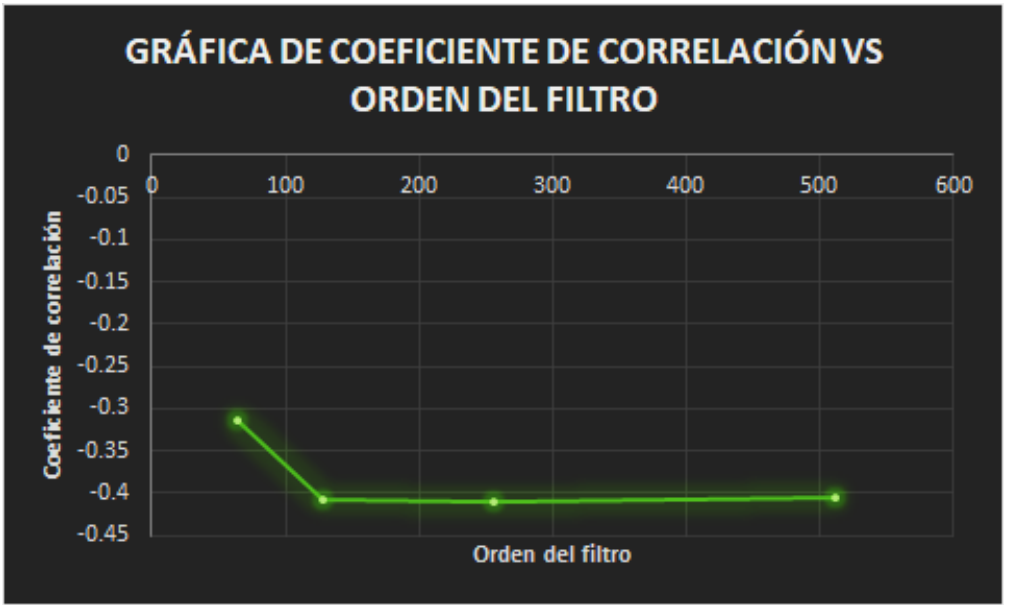


Figura 27. Gráfica de la señal 2 estado 2 de coeficiente de correlación.

La correlación será el criterio que nos dará información válida para evaluar el rendimiento. Podemos observar que hasta la prueba con orden del filtro de 256 el valor de la correlación aumentó si nos referimos a que el valor máximo es de -1; en esta señal mientras mayor era el valor de la correlación, mejor la calidad de la voz que se obtenía. A pesar de que en la última prueba se observa que el coeficiente de correlación cae un poco, la calidad del sonido mejoró y por ese motivo no se descarta la prueba con orden de filtro M=512.

Figura 28. Gráfica de la señal 2 estado 2 de comparación del valor del SNR

De la gráfica anterior se puede observar que el valor del SNR de la señal filtrada prácticamente se iguala en la prueba final, mientras que en las otras se acerca considerablemente. Lo antes señalado nos indica que el mejor valor del orden del filtro es el de 512.

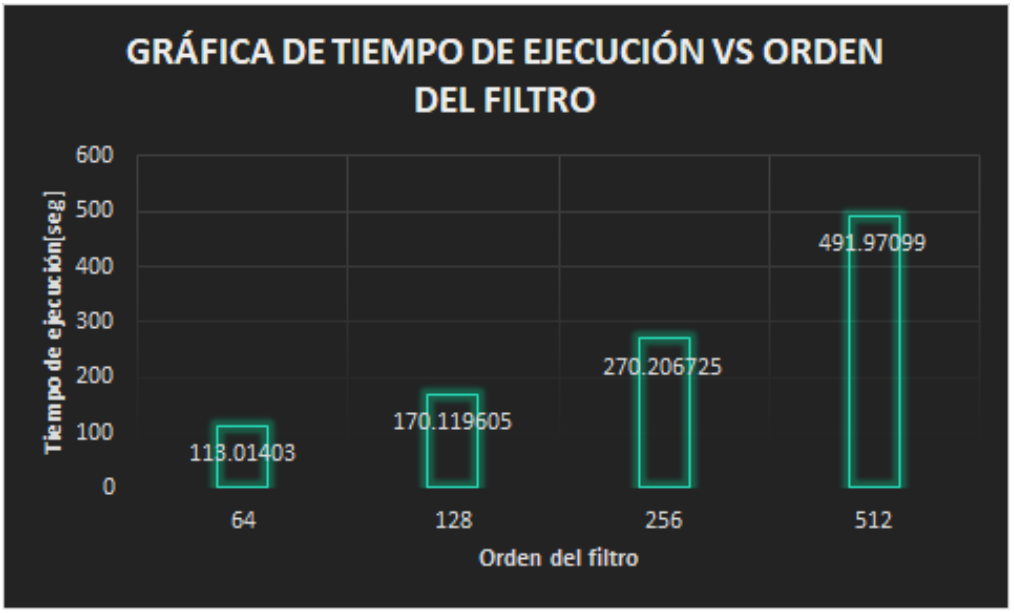


Figura 29. Gráfica de la señal 2 estado 2 de tiempo de ejecución.

De igual manera que la prueba anterior, si bien el tiempo de ejecución es importante y se considera para la tercera prueba, los criterios más valiosos serán la correlación y SNR y con base en ello, se toman los valores de las dos últimas muestras (9 y 10) para llevar a la tercera prueba junto con las dos anterior de la señal con voz y ruido de trasfondo.

En total serán los valores de 3 pruebas los que servirán de punto de partida para evaluar la tercera señal creyendo que van a ser muy acertados ya que tanto en esta señal como en la tercera lo que se filtra es voz.

SEÑAL 3

No. Prueba	Promedio de la función de costos señal de salid o ruido	Promedio de la función de costos señal deseada o error	Promedio de la señal deseada absoluta	Promedio de la señal de salida o ruido absoluto	Coeficiente de correlación	Tiempo de ejecución [seg]	SNR x [dBc]	SNR d [dbC]	SNR e [dBc]
1	0.0495	0.0162	0.1064	0.1844	0.2961	230.740432	7.2048	1.965	2.678
2	0.0501	0.0162	0.1062	0.1854	0.2894	235.642759	7.2048	1.965	2.6694
3	0.0488	0.0197	0.118	0.1830	0.3342	230.315027	7.2048	1.965	3.4446
4	0.0536	0.0114	0.0875	0.191	0.1663	230.849567	7.2048	1.965	0.6867
5	0.056	0.0106	0.0826	0.1954	0.102	237.473442	7.2048	1.965	-1.8875
6	0.0577	0.0116	0.0859	0.1984	0.084	242.869358	7.2048	1.965	-3.1929
7	0.0601	0.0127	0.0891	0.202	0.0537	254.130508	7.2048	1.965	-4.2155
8	0.0674	0.0185	0.1045	0.2121	0.0216	248.738759	7.2048	1.965	-7.0766
9	0.0681	0.0191	0.1048	0.2129	0.0175	428.718052	7.2048	1.965	-7.6745

Para realizar el análisis de los resultados se toma desde la prueba 4, ya que las pruebas anteriores sirvieron para ver cómo debían cambiar los valores y se comprobarán las observaciones que se plantearon se tendrían antes de iniciar las pruebas.

Figura 30. Gráfica de la señal 3 de la función de costos de la señal deseada

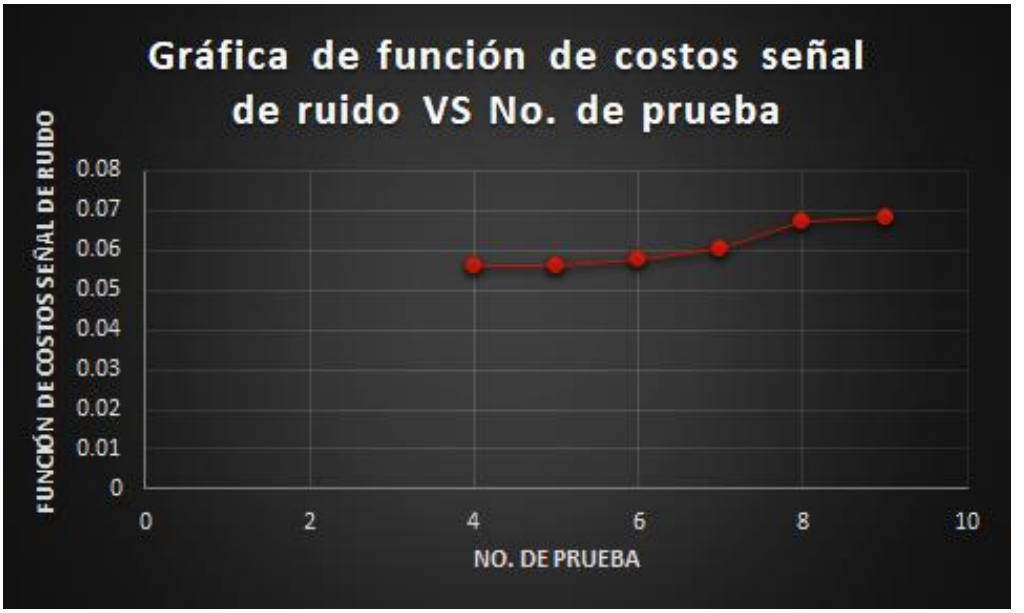


Figura 31. Gráfica de la señal 3 de la función de costos de la señal de ruido.

Como se puede apreciar en las dos gráficas anteriores la función de costos de la señal de ruido aumenta conforme cambian los valores de los parámetros y mejora el filtro y la función de la señal deseada tiende a reducir en las primeras pruebas y aumenta en las pruebas finales debido a los picos que se tienen de la señal de voz, estos resultados son los que se esperaban ya que la señal *e* es la que contiene a la voz y la diferencia entre la señal a filtrar y la señal de ruido de referencia, por lo que a medida que mejora el filtro la voz debe apreciarse más. Sin embargo, en ambas gráficas aumentan de valor por lo cual no son un buen criterio de elección.

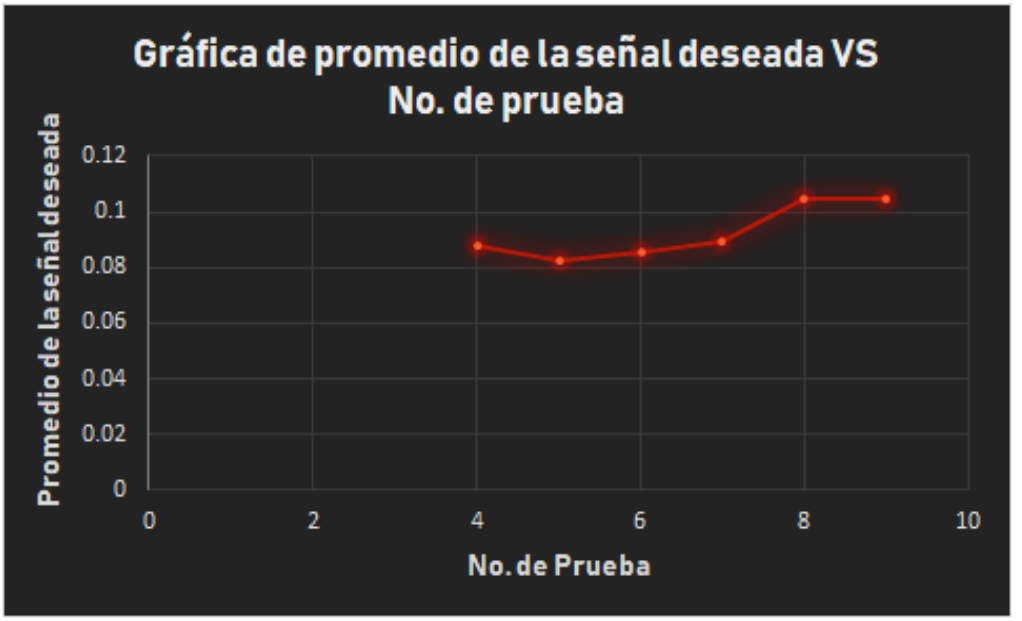


Figura 32. Gráfica de la señal 3 de error absoluto de la señal deseada.

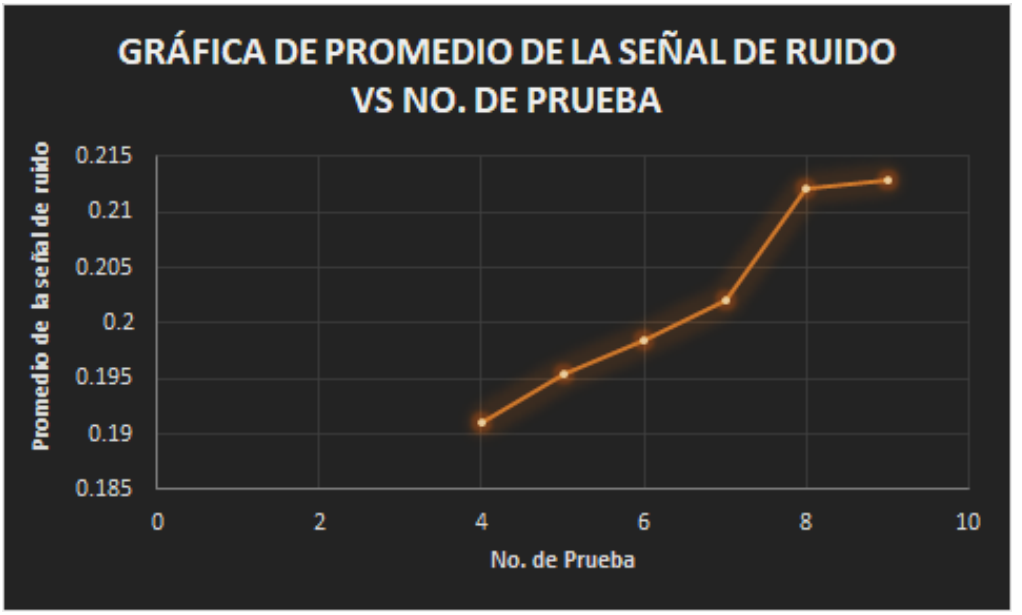


Figura 33. Gráfica de la señal 3 de error absoluto de la señal de ruido.

De igual manera que en las dos gráficas anteriores, se aprecian los resultados esperados, a medida que el filtro mejora el promedio de la señal de salida se reduce y posteriormente aumenta, ya que la señal se va acercando más a la señal de la voz y el ruido que contiene se va haciendo más pequeño por lo que al hacer el promedio hay menos muestras con amplitud grande y esa amplitud reducida se ve agregada a la señal de ruido que se está quitando y se aprecia cómo aumenta su promedio.

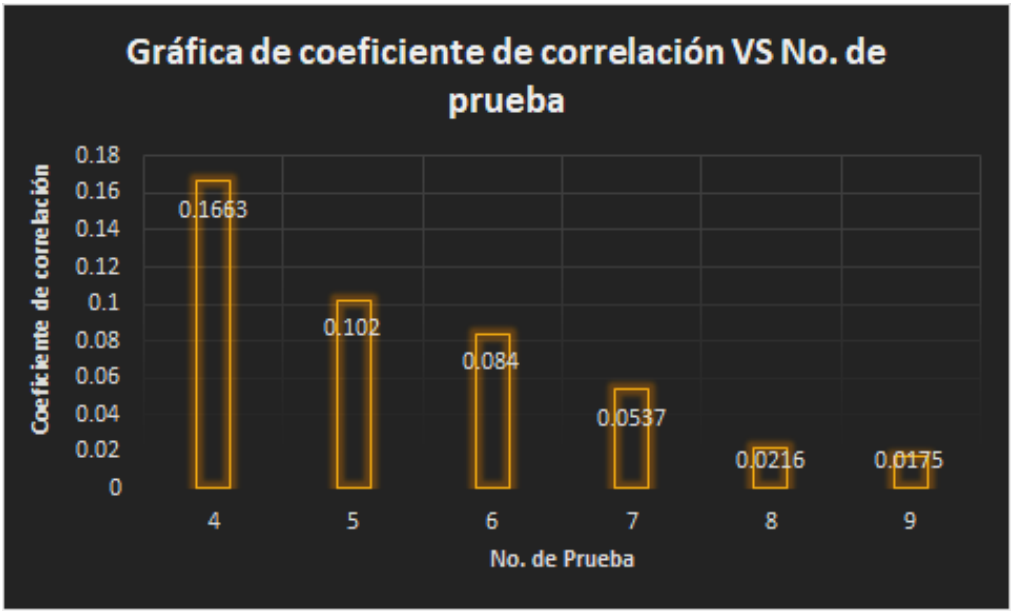


Figura 34. Gráfica de la señal 3 de coeficiente de correlación

Contrario a lo que se buscaba en las pruebas anteriores donde se quería aumentar el coeficiente de correlación, para esta señal se quería lo contrario ya que eso indica que entre menos se parecen las señal a filtrar y la de referencia, la voz se va esclareciendo ya que es la única señal que hace que las dos anteriores difieran, por tanto entre menor sea el coeficiente de correlación, mayor será la claridad de la voz. De la gráfica se puede observar que los mejores coeficientes de correlación los tienen las dos últimas pruebas, sobre todo la última que es en la que el orden del filtro se aumentó a 512 y eso también mejoró la calidad de la voz.

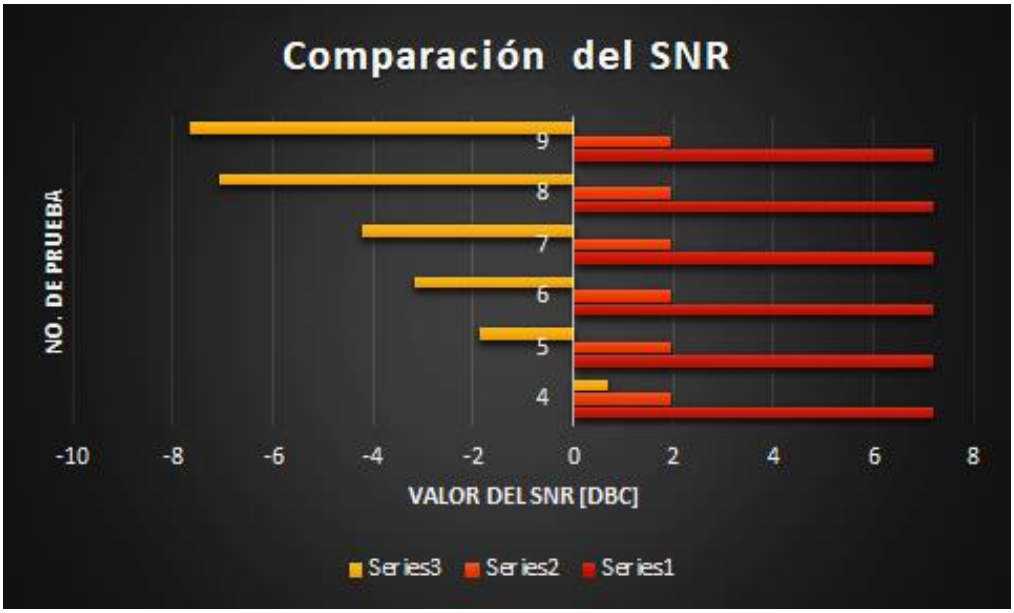


Figura 35. Gráfica de la señal 3 de comparación del valor de SNR.

En la gráfica del SNR lo que se planeaba obtener era que el SNR de e' en cada iteración se fuera alejando tanto del valor del SNR x y SNR d, ya que ello indica que el ruido cada vez es menor en la señal filtrada y se puede apreciar que el valor se aleja más entre mayor es el orden del filtro y en el cambio de las variables.

Hasta el momento las dos últimas pruebas son las que arrojaron mejores resultados, pero esta vez el tiempo de ejecución junto con el coeficiente de correlación serán los criterios de decisión para saber qué parámetros se llevarán a la implementación en tiempo real.

Figura 36. Gráfica de la señal 3 de tiempo de ejecución del algoritmo.

El menor tiempo de ejecución se tuvo en las pruebas 5 y 8, sin embargo en la prueba 9 se dobló el orden del filtro pero no así el tiempo de ejecución. El mejor resultado se obtuvo en la prueba número 9 aunque el tiempo de ejecución aumentó en un 68.34% y al momento de llevar esos parámetros a tiempo real la complejidad computacional se verá aumentada, por ende la velocidad de procesamiento deberá aumentar considerablemente pero hasta este punto también es importante evaluar la calidad de voz obtenida y con el orden de 512 mejora.

Considerando todo lo anterior se quedarán los valores de alfa=0.99, ep=0.00000000000000000000000000008 y ga=0.02, ya que si estos valores aumentan el filtro deja de funcionar adecuadamente. Con respecto al orden del filtro, la calidad de la voz mejora al aumenta el orden del filtro por lo cual se recurre a llegar a un punto intermedio entre 256 y 512, el cual será 384, pero para eso el dispositivo a elegir deberá correr a una velocidad alta de procesamiento para ser capaz de tener el menor retraso posible.

Si bien a lo largo del desarrollo se obtuvieron los parámetros M=512, alfa=0.99, ep=0.00000000000000000000000000008 y ga=0.02 para llevar a tiempo real y que gracias al valor del coeficiente de correlación permiten ver que los resultados obtenidos al filtrar la voz son eficientes, se observó que para la implementación de los filtros adaptativos no hay reglas para elegir los valores de los parámetros, ya que aunque se tienen sugerencias, la mejor manera de llegar es experimentalmente con pruebas como las realizadas anteriormente y van a depender mucho de la señal que se esté filtrando, entre más ruido haya aunado a la voz los parámetros deberán ser más altos para quitar el mayor ruido posible y con menor ruido los parámetros serán más pequeños y reducirán al igual el mayor ruido posible. Dicho lo anterior, aunque se eligieron parámetros para llevar a tiempo real y con base en ellos se elegirá un dispositivo que permita trabajar a una velocidad grande, debemos tomar en cuenta que tal vez

los parámetros tendrán que cambiarse para adaptarse a lo que se filtre pero eso podría ayudar a establecer modos en el prototipo donde se puede establecer si hay poco, medio o mucho ruido y que dependiendo del ambiente se establezca el modo adecuado.

Además de lo anterior, siendo el valor del SNR el parámetro más importante para evaluar la eficiencia del filtro y con ayuda de la literatura [14], se pudo encontrar que con diversas técnicas que se aplican a los auxiliares auditivos para la reducción digital de ruido se tienen los resultados registrados a lo largo de la historia en cuanto a la mejora obtenida en SNR e inteligibilidad con cada una de las técnicas. A continuación se muestran los resultados:

- ❏ Filtrado adaptativo predictivo. Ganancia de aproximadamente 7[dB] en SNR y no se han reportado ganancias en la inteligibilidad.
- ❏ Sustracción espectral. Las pruebas de inteligibilidad fallaron para verificar un incremento en ella. Esta técnica es inefectiva cuando la media del error de fondo es igual o mayor que la de la voz.
- ❏ Cancelación de ruido. La ganancia registrada en el SNR ha sido mayor de 15[dB] con una ganancia en inteligibilidad del 38%.
- ❏ Cancelación de reverberación.

Cabe mencionar que la literatura menciona que con un correcto uso de micrófonos direccionales se puede obtener una ganancia adicional en el SNR de 3 a 4[dB]. [14]

5. Validación del algoritmo en tiempo real

Se planea transferir el algoritmo a software que permita evaluar que el proceso se está llevando a cabo en tiempo real y así mismo, verificar que los parámetros obtenidos de simulación arrojen resultados que permitan validar la futura implementación del algoritmo en tiempo real con audio y voz.

Para considerar que el proceso se realice en tiempo real, según bibliografía [15], el retraso en voz debe ser menor de 125 000[ns] y en audio menor a 22 727[ns].

Considerando que la idea es poder llevar el algoritmo a pruebas reales con audio y voz se propone hacer la selección de un dispositivo que sea capaz de procesar ese tipo de señales y que su nivel de procesamiento permita llevarlo a cabo en tiempo real considerando que las etapas para la implementación serían las siguientes:

- Adquisición de señal de audio a filtrar y señal de audio de referencia
- Conversión analógica a digital de ambas señales
- Filtrado adaptativo
- Conversión digital a analógica de la señal filtrada
- Envío de la señal filtrada a un audífono

En la siguiente sección se presentan los dispositivos entre los que se hará la elección y las características de cada uno de ellos.

5.1 Selección de dispositivos

En la actualidad, los cuatro grandes fabricantes de DSP son Texas Instruments, con la serie TMS320; Motorola, con las series DSP56000, DSP56100, DSP56300, DSP56600 y DSP96000; Lucent Technologies (anteriormente AT&T), con las series DSP1600 y DSP3200; y Analog Devices, con las series ADSP2100 y ADSP21000. Sin embargo, los que han dado mejores resultados en el mercado son los de las marcas Motorola y Texas Instruments, por lo cual analizan dispositivos pertenecientes a estar marcas para realizar la elección de dispositivo.

Tabla 6. Tabla de dispositivos con sus características

DISPOSITIVO	PROPIEDADES
DSP56002EVM	❖ Opera a 40 MHz ❖ Realiza más de 20 millones de instrucciones por segundo. ❖ Tiene un ciclo de instrucción de 50ns a 40 MHz ❖ Realiza más de 120 millones de operaciones por segundo a 40 MHz
DSP56603EVM	❖ Opera a 60 MHz
TMS320C6748 DSP Development Kit (LCDK)	❖ Opera a 375 y 456 MHz ❖ DSP de punto fijo y punto flotante ❖ Arquitectura de memoria cache de 2 niveles ❖ Controlador LCD

	❖ Comunicación SPI, I^2C, UART, SATA ❖ Puerto multicanal de audio con comunicación I2S ❖ 2 Puertos multicanal de buffer serial ❖ Permite realizar más de 8 operaciones por ciclo.
TMS320C6713 DSP Starter Kit(DSK)	❖ Opera a 300, 225, 200, 167 MHz ❖ DSP de punto fijo y punto flotante ❖ Comunicación SPI, I^2C

A continuación se muestra una matriz de decisión con los principales criterios a tomar en cuenta para la elección del dispositivo y el resultado que se obtuvo con esa matriz, tomando en cuenta que se evalúa con una escala del 0 al 10 para cada criterio, con la siguiente especificación para cada criterio:

- Criterio 1: 0→Costo muy elevado, 10→Costo reducido
- Criterio 2: 0 ó 10 si no cuenta o cuenta con puertos de audio respectivamente
- Criterio 3: 0→Más de 10 dispositivos auxiliares, 10→Ningún dispositivo auxiliar
- Criterio 4: 0→Menor a 30MHz, 10→Mayor a 300MHz

Tabla 7. Matriz de decisiones para la selección de dispositivo

ALTERNATIVAS DE DISPOSITIVOS	DSP56002EVM	DSP56603EVM	TMS320C6748 DSP Development Kit (LCDK)	TMS320C6713 DSP Starter Kit(DSK)
CRITERIOS				
Costo del dispositivo	8	8	8	5
Cuenta o no con puertos para audio	10	10	10	10
Necesidad de conectar a otros dispositivos auxiliares o no (CODEC, etc.)	6	6	9	9
Velocidad de procesamiento	2	4	10	10
TOTAL	26	28	37	34

De la matriz anterior podemos observar que el dispositivo que ofrece mayores ventajas es el TMS320C6748 DSP Development Kit de Texas Instruments, así que se decide desarrollar la parte de validación es este dispositivo.

Para poder comenzar a desarrollar la aplicación en la tarjeta TMS320C6748, debido a sus características, fue necesario utilizar un emulador de comunicación JTAG para poder

conectar con los puertos USB de la computadora. El emulador es el TI XDS100V2 de la compañía Spectrum Digital.

5.2 Implementación en software del algoritmo en tiempo real

Para la implementación en tiempo real se decidió utilizar el ambiente de desarrollo de Texas Instruments, Code Composer versión 5.5, que cuenta con un sistema operativo para operaciones en tiempo real (RTOS) pudiendo monitorear el tiempo de ejecución y el cambio de valores en las variables de importancia en el kit de desarrollo.

La implementación en tiempo real consistió en generar una señal sinusoidal con las siguientes características, que será considerada como la señal deseada (d) y simulará la voz:

- Frecuencia: f=44100
- Periodo: p=1/f
- Amplitud: a=1

También se generará la señal a filtrar (x) como un vector de números aleatorios que van a simular ser el ruido ambiental junto con la voz, ya que en el ambiente los sonidos no son uniformes en frecuencia ni en amplitud y simula adecuadamente la no estacionariedad de las señales reales.

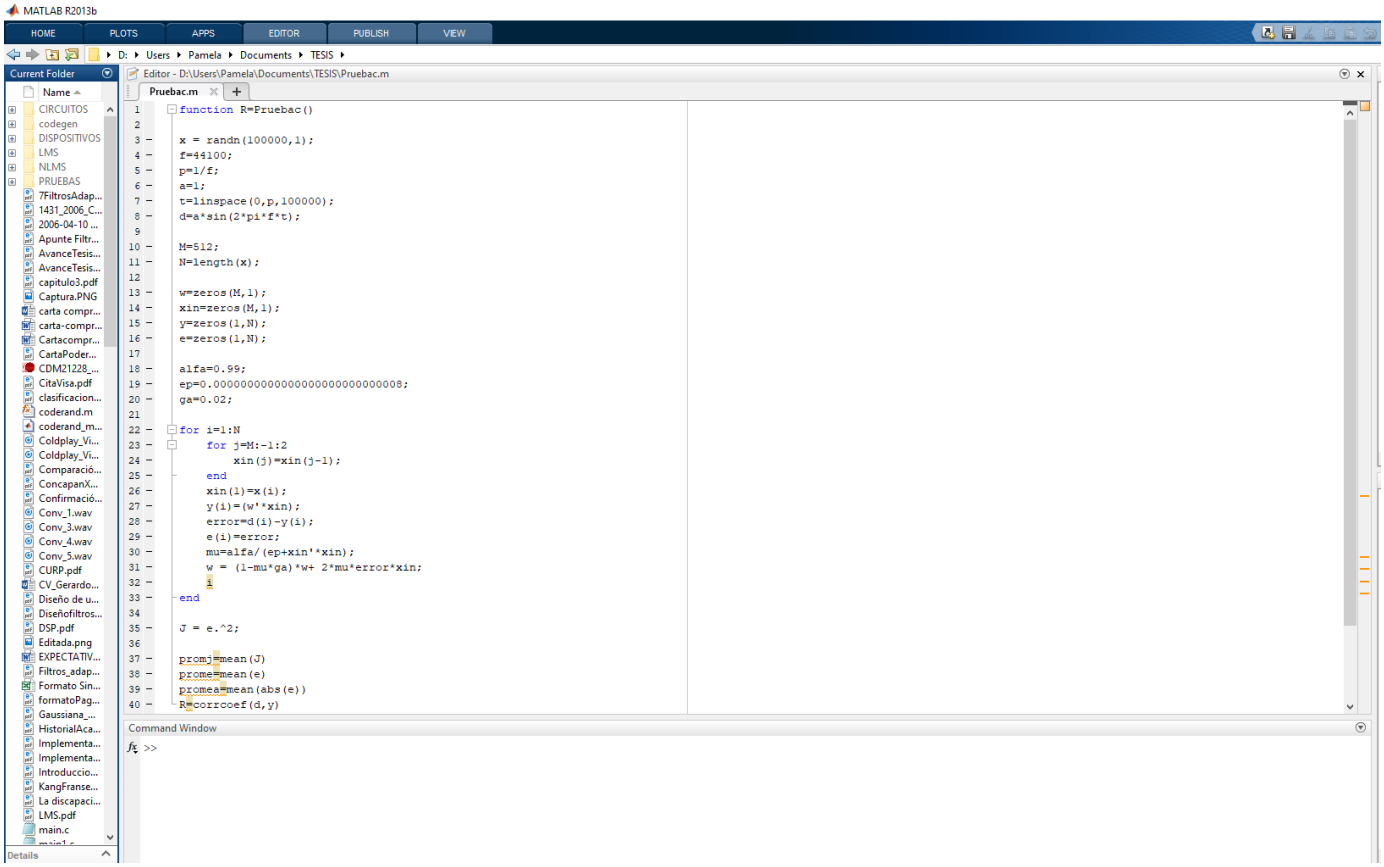
Finalmente, con las señales previamente generadas se evaluarán los parámetros finales que se obtuvieron en la simulación con MATLAB:

- M=512
- alfa=0.99
- ep=0.00000000000000000000000000008
- ga=0.02

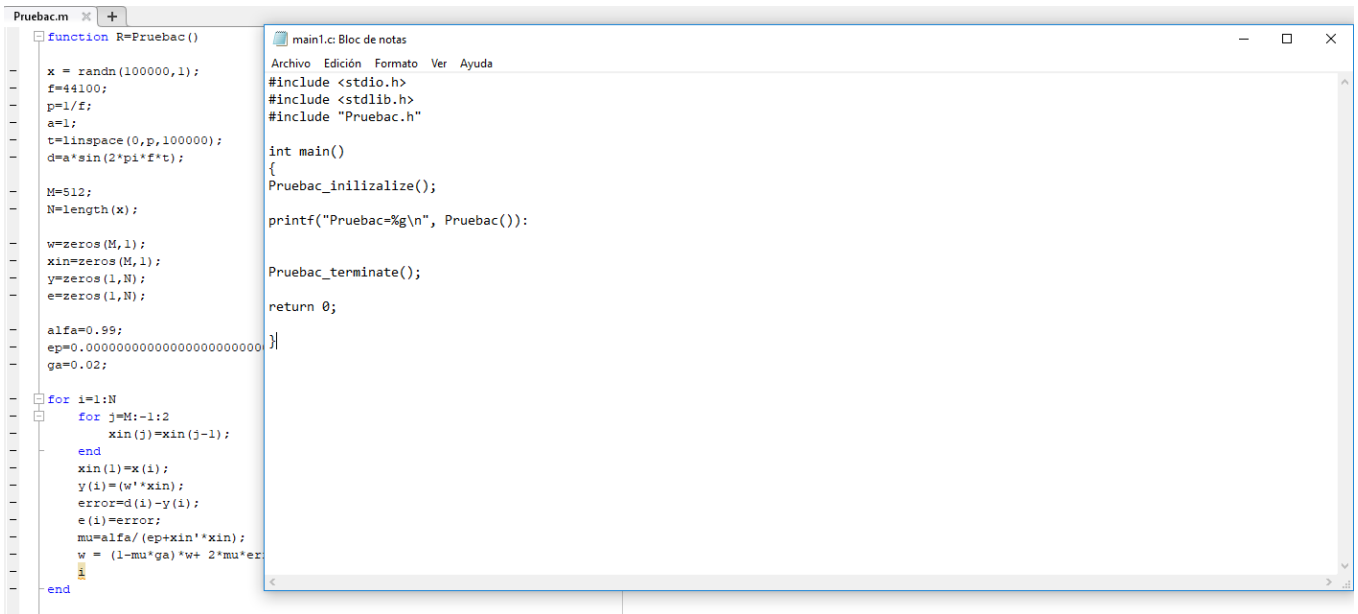
6. Resultados

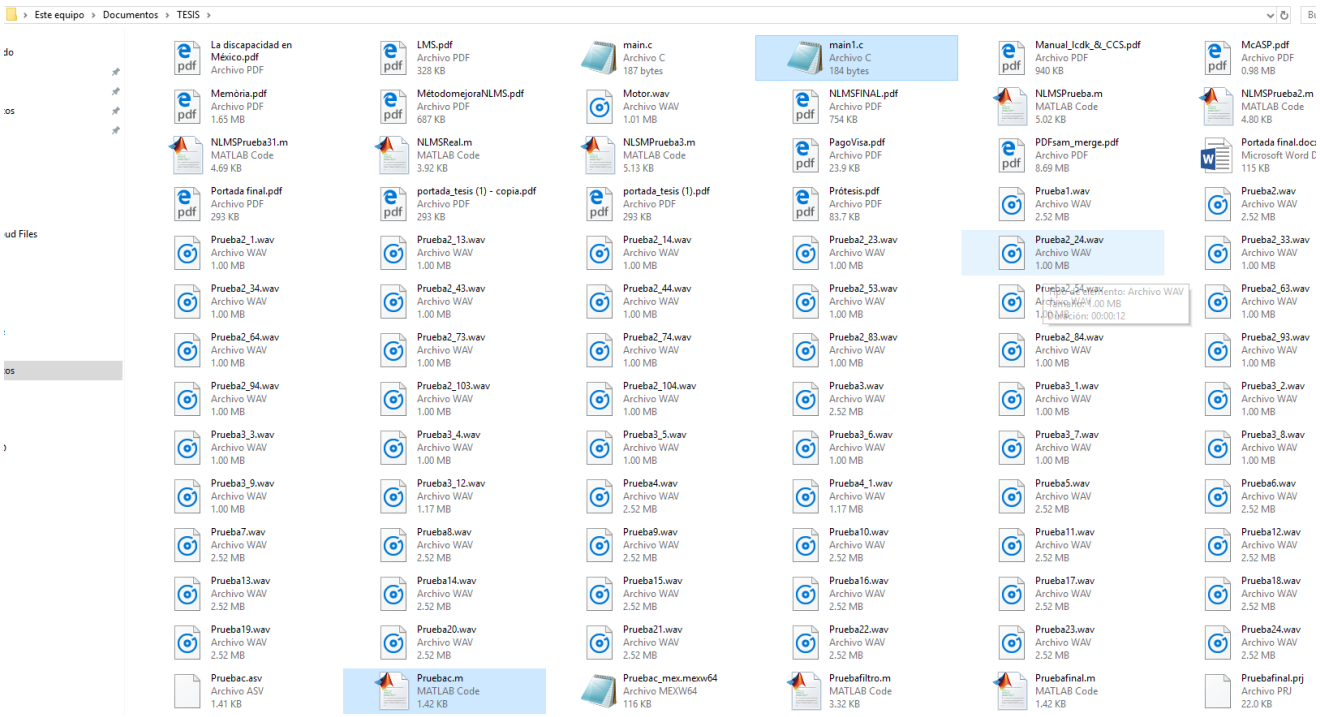
Para facilitar la implementación en tiempo real se utilizó la función CODEGEN que es parte del software MATLAB, con el fin de migrar el código de prueba a lenguaje C teniendo los mismo parámetros del filtro adaptativo. La migración se hizo de la siguiente manera:

I. Se tiene el código en MATLAB

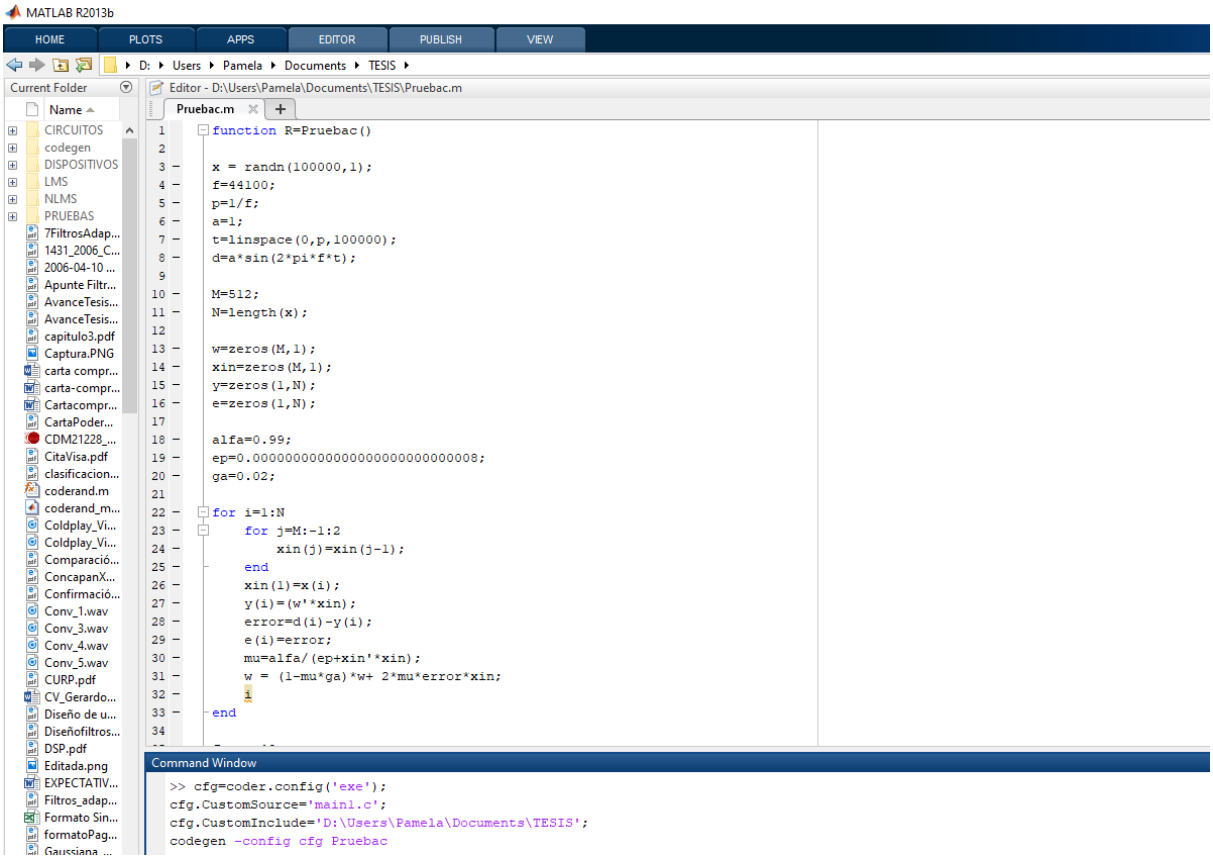


II. Se crea un archivo de texto main.c que será guardado con ese nombre y extensión en la misma carpeta donde está almacenado el problema de MATLAB, en el cual se inicializará y terminará la función como se aprecia a continuación:





III. A continuación se ejecutarán los comandos para codegen en MATLAB, específicamente en command window



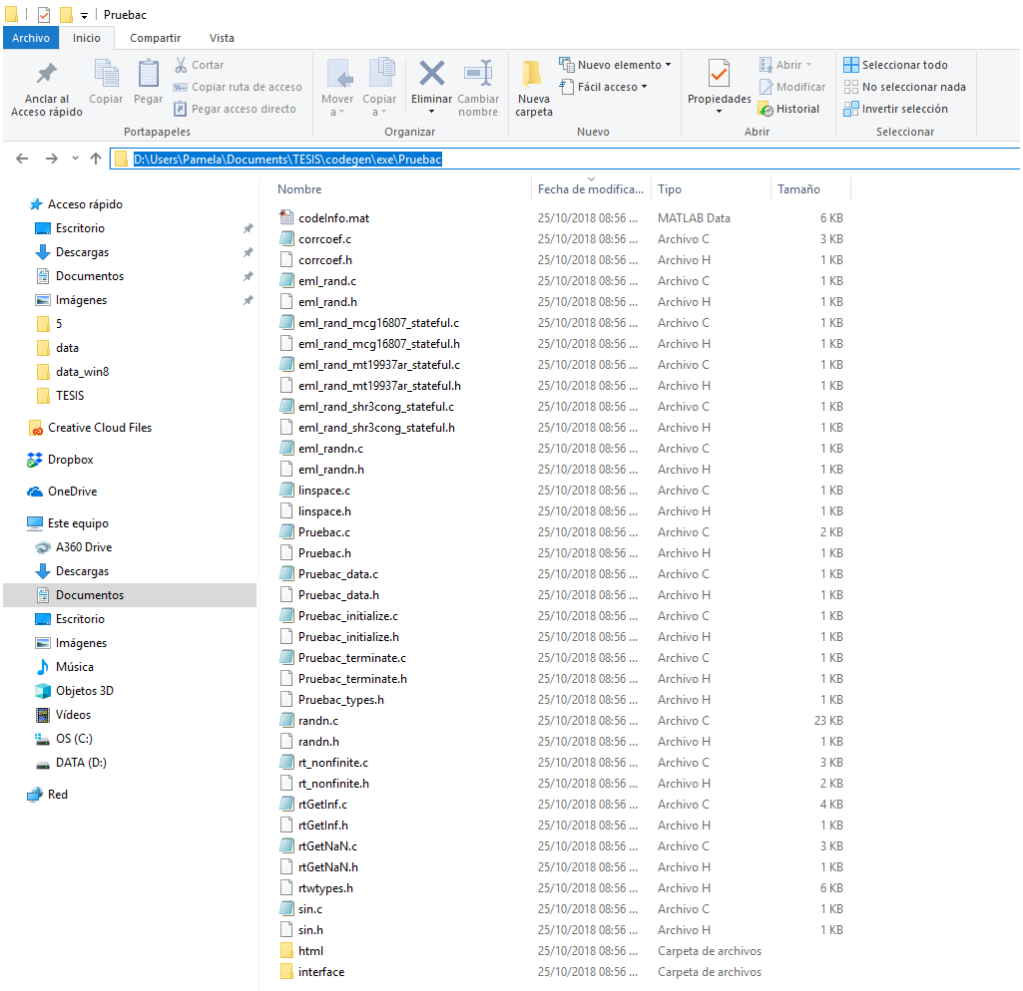
La primera línea configura para que se genere un ejecutable

La segunda línea toma el archivo donde se inicializa y termina la función

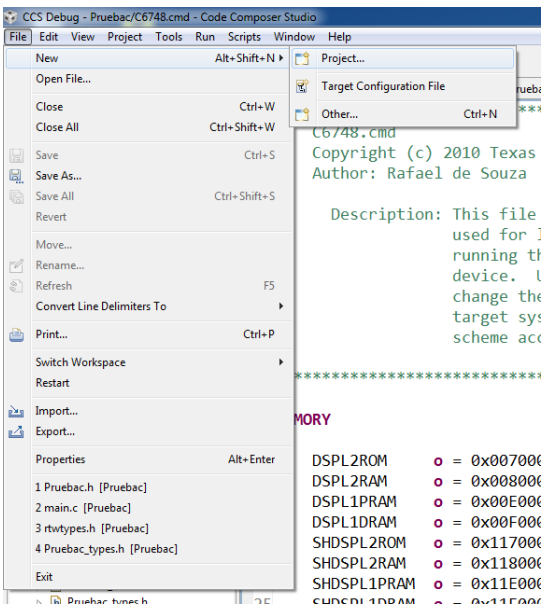
La tercera línea indica la ruta donde se encuentra el archivo main.

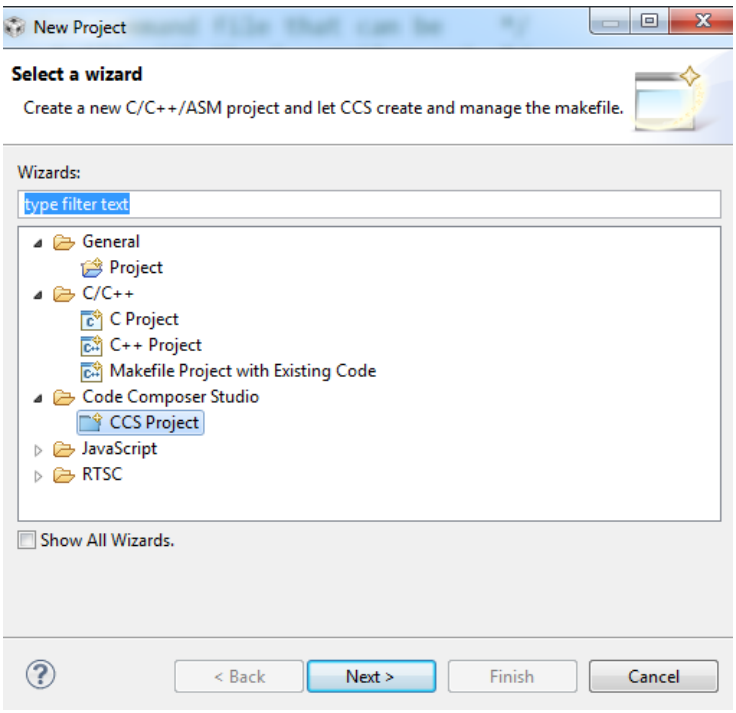
La cuarta línea genera el código del programa en MATLAB, en este caso Pruebac

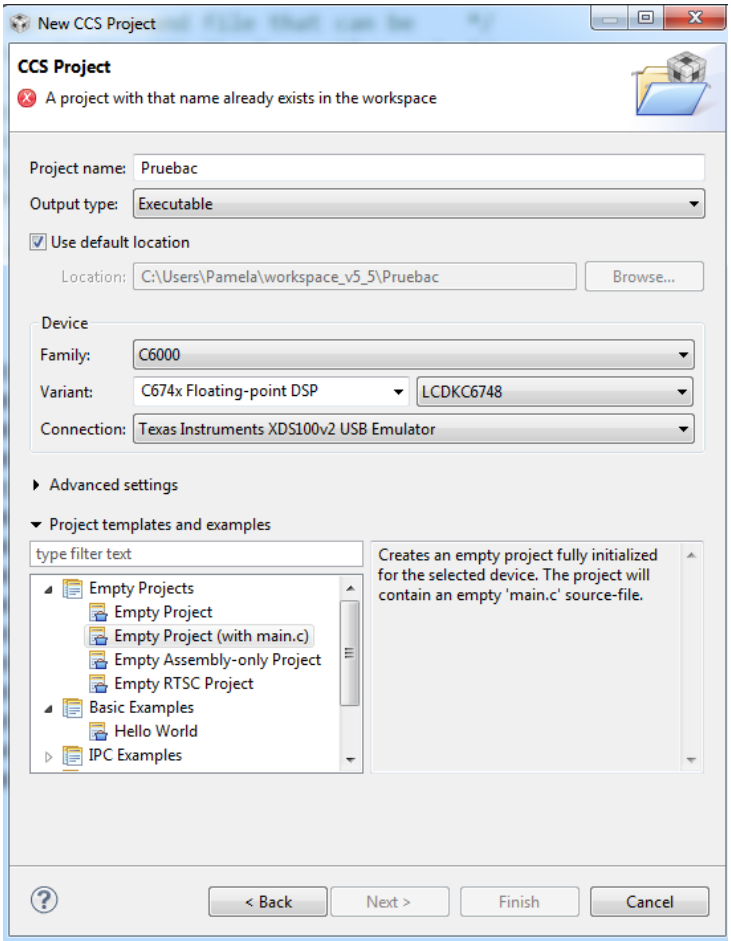
IV. Los archivos generados con guardados en una carpeta llamada: codegen que se guardará en la ruta donde se encuentra el archivo main. c, dentro de dicha carpeta existirá otra con el nombre exe que es la de interés y dentro de ella la carpeta con el nombre del programa de MATLAB: Pruebac donde estarán guardados todos los archivos necesarios para a Code Composer.



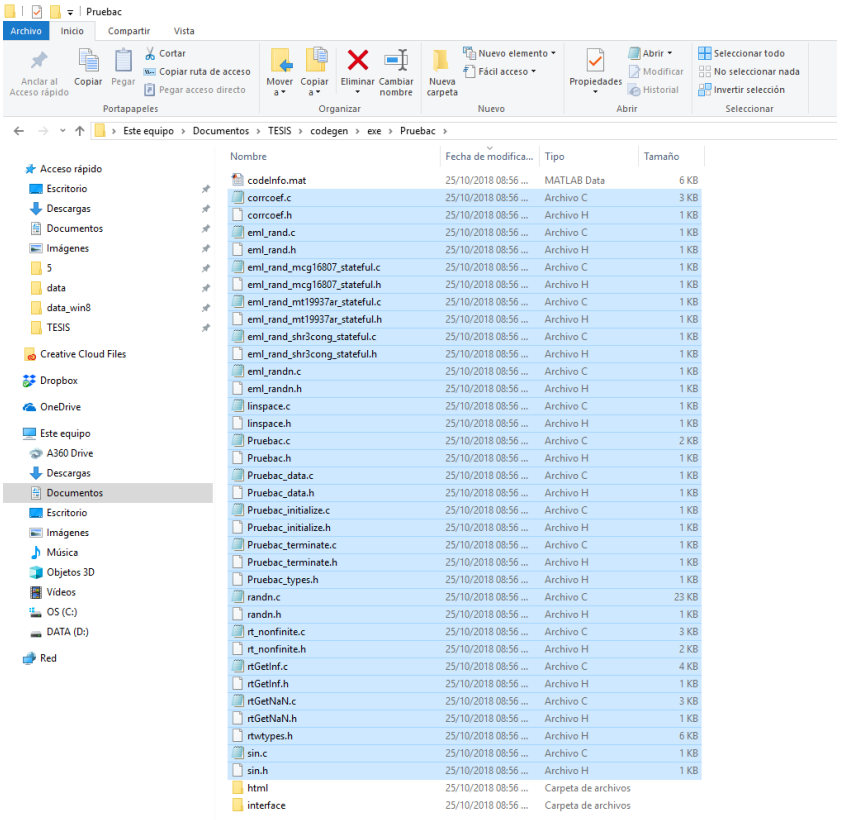
V. Se crea un nuevo proyecto en Code Composer de la siguiente manera:

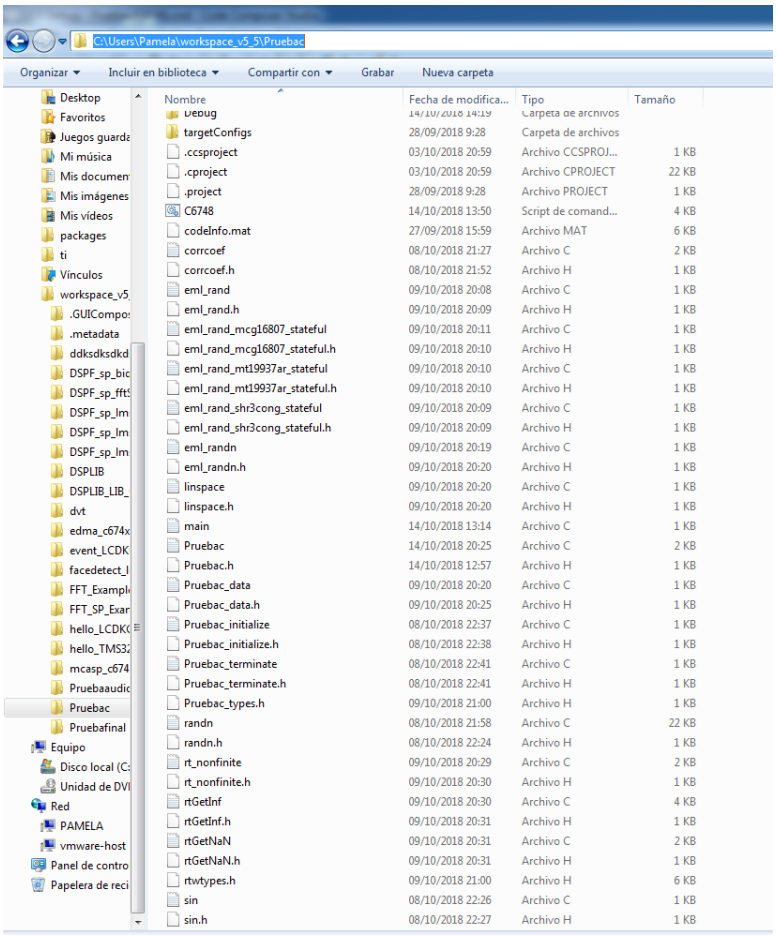



New Project
Select a wizard
Create a new C/C++/ASM project and let CCS create and manage the makefile.
Wizards:
type filter text
General
Project
C/C++
C Project
C++ Project
Makefile Project with Existing Code
Code Composer Studio
CCS Project
JavaScript
RTSC
Show All Wizards.
< Back
Next >
Finish
Cancel

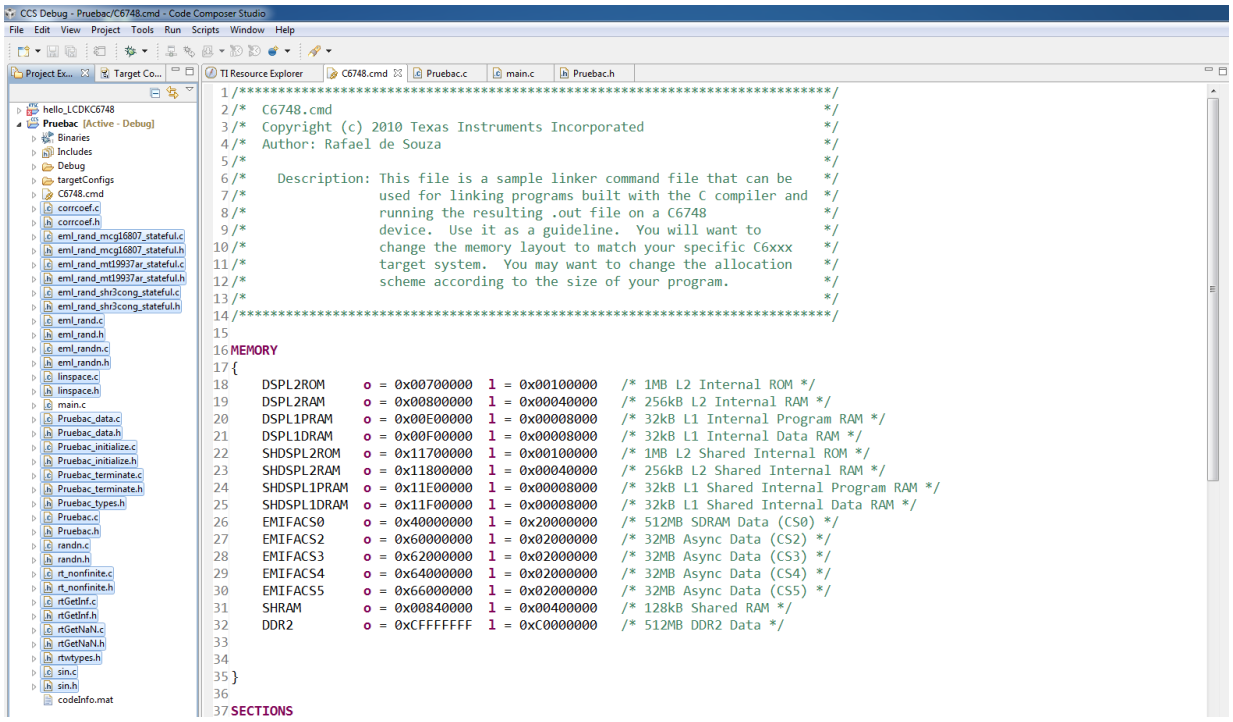

New CCS Project
CCS Project
A project with that name already exists in the workspace
Project name: Pruebac
Output type: Executable
Use default location
Location: C:\Users\Pamela\workspace_v5_5\Pruebac
Browse...
Device
Family: C6000
Variant: C674x Floating-point DSP
LCDKC6748
Connection: Texas Instruments XDS100v2 USB Emulator
Advanced settings
Project templates and examples
type filter text
Empty Projects
Empty Project
Empty Project (with main.c)
Empty Assembly-only Project
Empty RTSC Project
Basic Examples
Hello World
IPC Examples
Creates an empty project fully initialized for the selected device. The project will contain an empty 'main.c' source-file.
< Back
Next >
Finish
Cancel

VI. Finalmente, una vez creado el proyecto se agregan todos los archivos con extensión .c y .h generados de MATLAB, se seleccionan todos los archivos a agregar y se copian para después pegarse en la carpeta donde se encuentra el proyecto de Code Composer.





Y eso se verá reflejado en Code Composer



Para validar los resultados obtenidos en la prueba de la implementación en tiempo real en la tarjeta de desarrollo, se obtuvo el valor de correlación para la misma señal con los mismos parámetros en MATLAB, en donde el resultado obtenido fue:

$$R = -0.2010$$

Mientras que los valores obtenidos en la tarjeta de desarrollo son:

$$R = -0.1794777592717047$$

Como error entre los dos resultados mostrados, si te toma el resultado de MATLAB como referencia, se tiene:

$$E_{relativo} = \frac{R_{MATLAB} - R_{CC}}{R_{MATLAB}} = \frac{-0.201 - (-0.1794777592717047)}{-0.201} = 0.107$$

Evaluando lo anterior se puede deducir que existen varias razones por las cuales los resultados entre ambos software fueron diferentes:

- El redondeo en cada uno es diferente, MATLAB redondea a menos dígitos mientras que en Code Composer se mantienen más dígitos para los datos flotantes. MATLAB redondea el número a cuatro dígitos después del punto decimal, lo que reduce la cantidad de cifras almacenadas en la memoria y la precisión del cálculo, mientras que Code Composer redondea a dieciséis cifras después del punto decimal; si se analiza lo anterior se puede deducir que a lo largo de la ejecución del código y de las operaciones algebraicas, en MATLAB el error de redondeo era mayor y se iba acumulando en cada iteración

mientras que en Code Composer los cálculos fueron más precisos y exactos. La decisión de la precisión del número depende de la exactitud requerida para los cálculos, la cantidad de pasos de procesamiento matemático y las tolerancias usadas durante la recolección de los datos. [20]

- La librería con la que cada software cuenta para discretizar la función seno a lo largo del total de muestras, ya que como sea sabe la función seno es una función continua en el tiempo, puede ir variando en algunos dígitos debido al redondeo o por el propio proceso de muestreo, pero por la cantidad de muestras el error se va acumulando.

Tomando en cuenta lo mencionado anteriormente en cuanto a la diferencia de valores entre ambos softwares, se inclina a pensar que el valor más acertado lo tiene Code Composer debido a que acarrea información hasta 16 dígitos después del punto decimal y si MATLAB acarrea información hasta solo 4 dígitos, en cada operación y en cada iteración se irá perdiendo mayor información que en Code Composer, además de que el número de iteraciones o muestras (100000) habla de que el error acumulado fue incluso mayor debido a ese número, por lo cual se puede garantizar que el algoritmo en Code Composer funciona correctamente y una señal que se obtenga del ambiente la procesará adecuadamente. Pero es importante ver que con los parámetros obtenidos para la prueba en Code Composer, el resultado de la correlación para la función seno no fue muy elevado en comparación con las señales de audio pregrabadas, lo cual permite ver que los parámetros deben adaptarse a cada una de una de las señales a procesar.

Conclusiones

Al abordar el problema que se plantea en esta tesis se quería dar una solución que ofreciera una mejora a los usuarios de auxiliares auditivos. Con la decisión de implementar algoritmos adaptativos para darle solución al problema se observan ciertos comportamientos que tienen este tipo de algoritmos y se pueden obtener las siguientes conclusiones:

- ✓ Los filtros adaptativos permiten que la atención se focalice en la señal deseada, ya que a pesar de que antes de filtrarse dicha señal esté combinada con ruido que contenga las mismas frecuencias que la señal deseada, el filtro mediante la captación del ruido de referencia puede diferenciar entre qué parte de la señal ruidosa corresponde al ruido puro y cuál a la señal deseada; comportamiento que no podría obtenerse con filtros de coeficientes fijos.
- ✓ A pesar de que todos los algoritmos adaptativos puedan filtrar una señal, algunos ofrecen mejoras en cuanto a velocidad de convergencia y costo computacional, ya que con ciertos cambios permiten que el algoritmo adapte los parámetros para una señal en específico y puede entregar más fácilmente la solución óptima, como lo fue en este caso el algoritmo NLMS.
- ✓ Pocos son los algoritmos que existen que permitan controlar o asegurar hasta cierto punto su convergencia a la solución óptima, en el caso de los filtros adaptativos el parámetro que permite asegurar dicha convergencia es el coeficiente de fuga, parámetro que se implementó en este trabajo con el fin de optimizar el algoritmo.
- ✓ Cada una de las señales que existen en el mundo real posee características que las hace únicas y que cambian sus valores estadísticos, por tal motivo para cada una de ellas los parámetros necesarios para el algoritmo adaptativo, como son: coeficiente de fuga, orden del filtro, valor de alfa para el cálculo específico del paso de adaptación muestra a muestra, serán diferentes y deben buscarse bajo varias iteraciones evaluando los criterios más fuertes que son: Correlación y SNR, como se observa en la parte de simulación del trabajo.
- ✓ No se debe de perder de vista que el comportamiento en simulación que tenga una señal y el filtro adaptativo puede cambiar radicalmente al llevarse a tiempo real ya que no existe control sobre factores que puedan interferir, por ello es necesario reevaluar los parámetros al llevar el algoritmo a tiempo real.
- ✓ Si bien se hizo la validación del algoritmo en cuanto a precisión y tiempo de ejecución en una tarjeta de desarrollo, para poder ser validado con señales reales tanto de voz como de ruido ambiental hay otros factores a tomar en consideración como la precisión con que digitalice las señales antes mencionadas, la calidad de audio de salida de la tarjeta, el

tiempo de retraso entre que la señal con ruido entre al sistema hasta que salga filtrada a un audífono, factores que permitirán observar la viabilidad de diseñar un sistema que pueda ser probado con personas.

Como parte del trabajo a futuro se consideran ciertas etapas para poder llegar al alcance final, un producto que se pueda comercializar. Las etapas antes mencionadas son:

- Implementación del algoritmo adaptativo en tiempo real con señales de audio pregrabadas.
- Implementación del algoritmo adaptativo en tiempo real con señales de audio captadas mediante micrófonos
- Establecer los parámetros finales para tiempo real y señales reales
- Crear una interfaz de control de volumen y parámetros del algoritmo adaptativo para cada usuario
- Miniaturización del dispositivo

Referencias bibliográficas

[1] *INEGI. La discapacidad en México, datos al 2014.* México,2016. Revisado en 2017.

[2] *Historia y Evolución de los aparatos auditivos.* Saber de ciencias. Revisado en 2017 en la página: https://www.saberdeciencias.com/noticias-de-ciencias/192-historia-y-evolucion-de-los-aparatos-auditivos

[3] *Características de los Aparatos Auditivos.* Aparato Auditivo.com.mx. Revisado en 2017 en la página: https://aparatoauditivo.com.mx/blog/consejos/125-caracteristicas-de-los-aparatos-auditivos

[4] J. G. Proakis, D. G. Manolakis. *Tratamiento digital de señales.* 4ª edición. Pearson Educación S.A., Madrid, 2007. Revisado en 2017.

[5] J. Rodrigo Mendoza Pérez, "*JAVAFilters: Cálculo y Diseño de Filtros Analógicos y Digitales.",* Escuela de Ingeniería Electrónica, UDLA-P. Primavera 2004. Revisado en 2017 en la página: http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/mendoza_p_jr/capitulo3.pdf

[6] D. Millán Domingo, "*Estudio y comparativa de diferentes algoritmos adaptativos para la identificación de sistemas*", Escola Politécnica Superior d'Enginyeria de Vilanova i la Geltrú, Universitat Politécnica de Catalunya. 5 julio de 2012. Revisado en 2017.

[7].*What is Stochastic Gradient Descent?* Quora. Revisado en 2017 en la página: https://www.quora.com/What-is-Stochastic-Gradient-Descent

[8] A.D.Poularikas. *Adaptive Filtering. Fundamentals of Least Mean Squares with MATLAB.* CRC Press. Taylor & Francis Group. 2015. Revisado en 2017

[9] J.R. Treichler. *Theory and Design of Adaptive Filters.* Texas Instruments, Inc. 1987. John Wiley & Sons.Revisado en 2017.

[10] Sayed, Ali. H. *Fundamentals of Adaptive Filtering.* IEEE Press. John Wiley & Sons. 2003. Revisado en 2017.

[11] *Coeficiente de correlación.* Revisado en 2017 en la página: https://www.uv.es/ceaces/base/descriptiva/coefcorre.htm

[12] Simón Bendezú, Giovani. *Filtro adaptativo LMS y su aplicación en el reconocimiento de palabras aisladas para el control de un equipo de sonido por medio de la voz.* Facultad de Ciencias e Ingeniería. Pontificia Universidad Católica del Perú/ Lima, Perú. 2004. Revisado en 2017 en la página: http://tesis.pucp.edu.pe/repositorio/bitstream/handle/123456789/280/SIMON_BENDEZU_GIOVANI_FILTRO_ADAPTIVO_LMS_APLICACI%C3%93N.pdf?sequence=1

[13]Ortiz P., David. Quintero M., Olga L. *Una aproximación al filtrado adaptativo para la cancelación de ruidos en señales de voz monofónicas.* Departamento de Ciencias Básicas, Universidad EAFIT. Medellín, Antioquia. Revisado en 2018 en la página:

https://repository.eafit.edu.co/bitstream/handle/10784/4611/28%20UnaAproximacionFiltradoAdaptativoCancelacionRuidos.pdf?sequence=1

[14] Welch B, Thad. Wright,Cameron H.G. Morrow, Michael G. *Real-Time Digital Signal Processing from MATLAB to C with the TMS320C6x DSPs.* CRC Press. Taylor & Francis Group. 2012. Revisado en 2018.

[15] Davis, Gillian M. *Noise Reduction in Speech Applications.* CRC Press LLC. 2002. Revisado en 2018.

[16]Kates, J.M. *Feedback cancellation in hearing aids: Results from a computer simulation.* IEEE Trans. ASSP, 39(3), 553-562, 1999. Revisado en 2018.

[17] Durán Villalobos, Carlos Alberto. *Implementación del algoritmo NLMS en un DSP.* Escuela Superior de Ingeniería Mecánica y Eléctrica. Sección de estudio de posgrado e investigación. Instituto Politécnico Nacional. Mayo 2010. Revisado en 2018.

[18] Hung Ngoc Nguyen, Majid Dowlatnia y Azhar. *Implementation of the LMS and NLMS algorithms for Acoustic Echo Cancellation in teleconference system using MATLAB.* School of Mathematics and Systems Engineering. Vaxjo University. Diciembre 2009. Revisado en 2018.

[19] *TMS320C6748™ Fixed- and Floating-Point DSP.* Texas Instruments. SPRS590G –JUNE 2009–REVISED JANUARY 2017. Revisado en 2018

[20] *Cómo incrementar los decimales en MATLAB.* Michael Peter. Revisado en 2018 en la página: https://techlandia.com/incrementar-decimales-matlab-como_187199/

[21] *Algoritmo de Mínimos Cuadrados con Error Codificado para Filtrado Adaptivo*. Revisado en 2018 en la página: https://scielo.conicyt.cl/pdf/infotec/v19n5/art10.pdf

[22] *TMS320C6748 Fixed- and Floating-Point DPS.* Texas Instruments. SPRS590G –JUNE 2009–REVISED JANUARY 2017. Revisado en 2018 en la página: http://www.ti.com/lit/ds/sprs590g/sprs590g.pdf

Anexos

A. Programa de simulación en MATLAB para la señal 1

```
%Filtro adaptativo ep-NLMS prueba
%Realizar pruebas con 64, 128, 256 y con alfa 0.015, 0.009 y 0.003;
%ep de 0.02 y 0.008; ga 0.1 y tomar el tiempo de ejecución de cada una de
%las pruebas y checar el error y la función de costos

tic                                                     %Inicia conteo de tiempo de ejecución

%Sección de código a medir

clear all                                               %Limpia todas las variables
clc                                                     %Limpia la ventana de comando

%Cargar las señales de entrada
[x,Fs,nbits]= wavread('Coldplay_Viva_La_Vida.wav');      %Señal de entrada con ruido
wavplay(x,Fs)                                           %Se reproduce la señal con ruido
[d,Fs,nbits]= wavread('Coldplay_Viva_La_Vida_Karaoke_Lyric_Video.wav');  %Señal deseada
wavplay(d,Fs)                                           %Se reproduce la señal deseada
x=x*1.7;
x=x(:);                                                 %Se convierte la señal x a vector
d=d(:);                                                 %Se convierte la señal d a vector

%Declarar las variables necesarias
M=64;                                                   %Número de orden del filtro
N=length(x);                                            %Número de iteraciones

%Inicializar variables en 0
w=zeros(M,1);                                           %Vector inicial de coeficientes del filtro
xin=zeros(M,1);                                         %Señal de entrada inicial

%Valores para calcular el paso de adaptación
alfa=0.015;
ep=0.02;

%Coeficiente de fuga
ga=0.1;

%---------------ALGORITMO 3-NLMS PARA FULTRO ADAPTATIVO-------------%
for i=1:N
    for j=M:-1:2
        xin(j)=xin(j-1);
    end
    xin(1)=x(i);                                        %Insertar nueva muestra al inicio de la señal
    y(i)=(w'*xin);                                      %Señal de salida del filtro adaptativo
    error=d(i)-y(i);                                    %Error
    e(i)=error;                                         %Almacenar el error estimado
    mu=alfa/(ep+xin'*xin);                              %Calcula el paso de adaptación
    w = (1-mu*ga)*w+ 2*mu*error*xin;                    %Actualización de coeficientes del filtro
    i
end

%Función de costo del filtro
J = e.^2;                                               %Curva de adaptación

%-----------------------------------------------------------
%GRAFICACIÓN DE LAS SEÑALES NECESARIAS
%-----------------------------------------------------------


%----------Señal deseada d(n)----------%
figure(1)
subplot(3,1,1)
plot(d)
xlabel('Tiempo (muestras)');
ylabel('d(n)');
title('SEÑAL DESEADA: d(n)')
grid on
axis([0 N -1 1]);


%---------Señal con ruido x(n)--------%
subplot(3,1,2)
plot(1.7*x)
xlabel('Tiempo (muestras)');
ylabel('x(n)');
title('SEÑAL CON RUIDO: x(n)')
grid on
axis([0 N -1 1]);
```

```
77 -    axis([0 N -1 1]);
78
79      %--------Señal de salida y(n)---------%
80 -    subplot(3,1,3)
81 -    plot(1.7*y)
82 -    xlabel('Tiempo (muestras)');
83 -    ylabel('y(n)');
84 -    title('SEÑAL DE SALIDA: y(n)')
85 -    grid on
86 -    axis([0 N -1 1]);
87
88      %---------Señal de error-----------%
89 -    figure(2)
90 -    subplot(2,1,1)
91 -    plot(abs(e),'red')
92 -    xlabel('Tiempo (muestras)');
93 -    ylabel('E(n)');
94 -    title('SEÑAL DE ERROR: e(n)')
95 -    axis([0 N -1 1]);
96 -    grid on
97
98      %--------Función de costos-------%
99 -    subplot(2,1,2)
100 -   plot(J,'red')
101 -   xlabel('Tiempo (muestras)');
102 -   ylabel('J(n)');
103 -   title('FUNCIÓN DE COSTOS: J(n)')
104 -   axis([0 N -1 1]);
105 -   grid on
106
107 -   promj=mean(J)                          %Se obtiene el promedio de J
108 -   prome=mean(e)                          %Se obtiene el promedio del error
109 -   promea=mean(abs(e))                    %Se obtiene el promedio del error abs
110 -   sound(y,Fs)                            %Se obtiene el audio de la salida del filtro
111 -   audiowrite('Pruebal.wav',y,Fs);        %Se manda a escribir el archivo de audio con la salida y
112 -   R=corrcoef(d,y)                        %Se obtienen los coeficientes de correlación entre "d" y "y"
113 -   r=snr(d,x)                             %Valor más alto porque es voz sobre canción
114 -   r1=snr(y',x)                           %Valor de la salida del filtro contra la señal con ruido
115 -   toc                                    %Finaliza el conteo del tiempo de ejecución
```

B. Programa de simulación en MATLAB para la señal 2

```
%Filtro adaptativo ep-NLMS prueba
%Realizar pruebas con 64, 128, 256 y con alfa 0.015, 0.009 y 0.003;
%ep de 0.02 y 0.008; ga 0.1 y tomar el tiempo de ejecución de cada una de
%las pruebas y checar el error y la función de costos

tic                                                     %Inicia conteo de tiempo de ejecución

%Sección de código a medir

clear all                                               %Limpia todas las variables
clc                                                     %Limpia la ventana de comando

%Cargar las señales de entrada
[x,Fs,nbits]= wavread('Conv_4.wav');                    %Señal de entrada con ruido
wavplay(x,Fs)                                           %Se reproduce la señal con ruido
[d,Fs,nbits]= wavread('Conv_1.wav');                    %Señal deseada
wavplay(d,Fs)                                           %Se reproduce la señal deseada
x=x(:);                                                 %Se convierte la señal x a vector
d=d(:);                                                 %Se convierte la señal d a vector

%Declarar las variables necesarias
M=512;                                                  %Número de orden del filtro
N=length(x);                                            %Número de iteraciones

%Inicializar variables en 0
w=zeros(M,1);                                           %Vector inicial de coeficientes del filtro
xin=zeros(M,1);                                         %Señal de entrada inicial

%Valores para calcular el paso de adaptación
alfa=0.5268;
ep=0.00000000000000000000000008;

%Coeficiente de fuga
ga=0.1 ;

%---------------ALGORITMO 3-NLMS PARA FULTRO ADAPTATIVO--------------%
for i=1:N
    for j=M:-1:2
        xin(j)=xin(j-1);
        xin(j)=xin(j-1);
    end
    xin(1)=x(i);                                        %Insertar nueva muestra al inicio de la señal
    y(i)=(w'*xin);                                      %Señal de salida del filtro adaptativo
    error=d(i)-y(i);                                    %Error
    e(i)=error;                                         %Almacenar el error estimado
    mu=alfa/(ep+xin'*xin);                              %Calcula el paso de adaptación
    w = (1-mu*ga)*w+ 2*mu*error*xin;                    %Actualización de coeficientes del filtro
    i
end

%Función de costo del filtro
J = e.^2;                                               %Curva de adaptación

%-------------------------------------------------------------
%GRAFICACIÓN DE LAS SEÑALES NECESARIAS
%-------------------------------------------------------------


%----------Señal deseada d(n)----------%
figure(1)
subplot(3,1,1)
plot(d)
xlabel('Tiempo (muestras)');
ylabel('d(n)');
title('SEÑAL DESEADA: d(n)')
grid on
axis([0 N -1 1]);


%---------Señal con ruido x(n)--------%
subplot(3,1,2)
plot(1.7*x)
xlabel('Tiempo (muestras)');
ylabel('x(n)');
title('SEÑAL CON RUIDO: x(n)')
grid on
axis([0 N -1 1]);

```

```
76 -    axis([0 N -1 1]);
77
78      %--------Señal de salida y(n)---------%
79 -    subplot(3,1,3)
80 -    plot(1.7*y)
81 -    xlabel('Tiempo (muestras)');
82 -    ylabel('y(n)');
83 -    title('SEÑAL DE SALIDA: y(n)')
84 -    grid on
85 -    axis([0 N -1 1]);
86
87      %---------Señal de error-----------%
88 -    figure(2)
89 -    subplot(2,1,1)
90 -    plot(abs(e),'red')
91 -    xlabel('Tiempo (muestras)');
92 -    ylabel('E(n)');
93 -    title('SEÑAL DE ERROR: e(n)')
94 -    axis([0 N -1 1]);
95 -    grid on
96
97      %--------Función de costos-------%
98 -    subplot(2,1,2)
99 -    plot(J,'red')
100 -   xlabel('Tiempo (muestras)');
101 -   ylabel('J(n)');
102 -   title('FUNCIÓN DE COSTOS: J(n)')
103 -   axis([0 N -1 1]);
104 -   grid on
105
106 -   promj=mean(J)                                   %Se obtiene el promedio de J
107 -   prome=mean(e)                                   %Se obtiene el promedio del error
108 -   promea=mean(abs(e))                             %Se obtiene el promedio del error abs
109 -   sound(y*1.7,Fs)                                 %Se obtiene el audio de la salida del filtro
110 -   audiowrite('Prueba2_104.wav',y,Fs);             %Se manda a escribir el archivo de audio con la salida y
111 -   R=corrcoef(d,y)                                 %Se obtienen los coeficientes de correlación entre "d" y "y"
112 -   r=snr(d,x)                                      %SNR de señal deseada contra señal con ruido
113 -   rl=snr(y',x)                                    %SNR de señal salida contra señal con ruido
114 -   toc                                             %Finaliza el conteo del tiempo de ejecución
```

C. Programa de simulación en MATLAB para la señal 3

```
%Filtro adaptativo ep-NLMS prueba para señal 3
tic                                                     %Inicia conteo de tiempo de ejecución

%Sección de código a medir

clear all                                               %Limpia todas las variables
clc                                                     %Limpia la ventana de comando

%Cargar las señales de entrada
[d,Fs,nbits]= wavread('Conv_4.wav');                    %Señal de entrada con ruido a filtrar
wavplay(d,Fs)                                           %Se reproduce la señal con ruido a filtrar
[x,Fs,nbits]= wavread('Motor.wav');                     %Señal de ruido de referencia
wavplay(x,Fs)                                           %Se reproduce la señal de ruido
x=0.55*x;
x=x(:);                                                 %Se convierte la señal x a vector
d=d(:);                                                 %Se convierte la señal d a vector


%Declarar las variables necesarias
M=512;                                                  %Número de orden del filtro
N=length(x);                                            %Número de iteraciones

%Inicializar variables en 0
w=zeros(M,1);                                           %Vector inicial de coeficientes del filtro
xin=zeros(M,1);                                         %Señal de entrada inicial

%Valores para calcular el paso de adaptación
alfa=0.99;
ep=0.00000000000000000000000000008;

%Coeficiente de fuga
ga=0.02;


%---------------ALGORITMO 3-NLMS PARA FULTRO ADAPTATIVO--------------%
for i=1:N
    for j=M:-1:2
        xin(j)=xin(j-1);
    end
    end
    xin(1)=x(i);                                        %Insertar nueva muestra al inicio de la señal
    y(i)=(w'*xin);                                      %Señal de salida del filtro adaptativo
    error=d(i)-y(i);                                    %Error
    e(i)=error;                                         %Almacenar el error estimado
    mu=alfa/(ep+xin'*xin);                              %Calcula el paso de adaptación
    w =(1-mu*ga)*w + 2*mu*error*xin;                    %Actualización de coeficientes del filtro
    i

end

%Función de costo del filtro
J = e.^2;                                               %Curva de adaptación
J1= y.^2;

%-------------------------------------------------------------
%GRAFICACIÓN DE LAS SEÑALES NECESARIAS
%-------------------------------------------------------------


%----------Señal a filtrar d(n)----------%
figure(1)
subplot(3,1,1)
plot(d)
xlabel('Tiempo (muestras)');
ylabel('d(n)');
title('SEÑAL A FILTRAR: d(n)')
grid on
axis([0 N -1 1]);


%---------Señal de ruido x(n)--------%
subplot(3,1,2)
plot(x)
xlabel('Tiempo (muestras)');
ylabel('x(n)');
title('SEÑAL DE RUIDO O REFERENCIA: x(n)')
grid on
axis([0 N -1 1]);
```

```
77 -    axis([0 N -1 1]);
78
79      %--------Señal de salida e(n)---------%
80 -    subplot(3,1,3)
81 -    plot(e)
82 -    xlabel('Tiempo (muestras)');
83 -    ylabel('e(n)');
84 -    title('SEÑAL DE SALIDA: e(n)')
85 -    grid on
86 -    axis([0 N -1 1]);
87
88      %--------Función de costos e-------%
89 -    figure(2)
90 -    subplot(2,1,1)
91 -    plot(J,'red')
92 -    xlabel('Tiempo (muestras)');
93 -    ylabel('J(n)');
94 -    title('FUNCIÓN DE COSTOS: J(n)')
95 -    axis([0 N -1 1]);
96 -    grid on
97
98      %--------Función de costos y-------%
99 -    figure(2)
100 -   subplot(2,1,2)
101 -   plot(J1,'red')
102 -   xlabel('Tiempo (muestras)');
103 -   ylabel('J1(n)');
104 -   title('FUNCIÓN DE COSTOS: J1(n)')
105 -   axis([0 N -1 1]);
106 -   grid on
107
108 -   promj=mean(J)                                   %Se obtiene el promedio de J
109 -   promj1=mean(J1)                                 %Se obtiene el promedio de J1
110 -   promea=mean(abs(e))                             %Promedio de la señal de e absoluto
111 -   promya=mean(abs(y))                             %Promedio de la señal de y absoluto
112 -   sound(e,Fs)                                     %Se obtiene el audio de la salida del filtro
113 -   audiowrite('Prueba3_9.wav',e,Fs);               %Se manda a escribir el archivo de audio con la salida e
114 -   R=corrcoef(d,e)                                 %Correlación entre la señal a filtrar y la señal de salida
115 -   r=snr(x)                                        %SNR de la señal de ruido de referencia
```

```
115 -    r=snr(x)                                       %SNR de la señal de ruido de referencia
116 -    r1=snr(d)                                      %SNR de la señal a filtrar
117 -    r2=snr(e')                                     %SNR de la señal de salida del filtro
118 -    toc                                            %Finaliza el conteo del tiempo de ejecución
119
```

D. Programas de verificación del algoritmo en Code Composer

D.1 Correlación

corrcoef.c

```c
/*
 *  corrcoef.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "corrcoef.h"

/* Function Definitions */
void corrcoef(const double x[100000], const double varargin_1[100000], double r
              [4])
{
  int k;
  static double b_x[200000];
  int j;
  double absrij;
  double d[2];
  for (k = 0; k < 100000; k++) {
    b_x[k] = x[k];
    b_x[100000 + k] = varargin_1[k];
  }

  for (k = 0; k < 4; k++) {
    r[k] = 0.0;
  }

  for (j = 0; j < 2; j++) {
    absrij = 0.0;
    for (k = 0; k < 100000; k++) {
      absrij += b_x[k + 100000 * j];
    }

    absrij /= 100000.0;
    for (k = 0; k < 100000; k++) {
      b_x[k + 100000 * j] -= absrij;
    }
  }

  for (j = 0; j < 2; j++) {
    absrij = 0.0;
    for (k = 0; k < 100000; k++) {
      absrij += b_x[k + 100000 * j] * b_x[k + 100000 * j];
    }

    r[j + (j << 1)] = absrij / 99999.0;
    k = j + 2;
    while (k < 3) {
      absrij = 0.0;
      for (k = 0; k < 100000; k++) {
        absrij += b_x[100000 + k] * b_x[k + 100000 * j];
      }

      r[1 + (j << 1)] = absrij / 99999.0;
      k = 3;
    }
  }

  for (k = 0; k < 2; k++) {
    d[k] = sqrt(r[k + (k << 1)]);
  }

  for (j = 0; j < 2; j++) {
    k = j + 2;
    while (k < 3) {
      r[1 + (j << 1)] = r[1 + (j << 1)] / d[1] / d[j];
      k = 3;
    }

    k = j + 2;
    while (k < 3) {
      absrij = fabs(r[1 + (j << 1)]);
      if (absrij > 1.0) {
        r[1 + (j << 1)] /= absrij;
      }

      r[2 + j] = r[1 + (j << 1)];
```

```
      r[2 + j] = r[1 + (j << 1)];
      k = 3;
    }

    if (r[j + (j << 1)] > 0.0) {
      if (r[j + (j << 1)] < 0.0) {
        r[j + (j << 1)] = -1.0;
      } else if (r[j + (j << 1)] > 0.0) {
        r[j + (j << 1)] = 1.0;
      } else {
        if (r[j + (j << 1)] == 0.0) {
          r[j + (j << 1)] = 0.0;
        }
      }
    } else {
      r[j + (j << 1)] = rtNaN;
    }
  }
}
```

corrcoef.h

```
/*
 * corrcoef.h
 */

#ifndef __CORRCOEF_H__
#define __CORRCOEF_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void corrcoef(const double x[100000], const double varargin_1[100000], double r[4]);
#endif
```

D.2 Generador de números aleatorios

randn.c

```
/*
 * randn.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "randn.h"
#include "Pruebac_data.h"

/* Variable Definitions */
static unsigned int d_state[625];

/* Function Declarations */
static double b_genrandu(unsigned int mt[625]);
static double eml_rand_mt19937ar(unsigned int e_state[625]);
static double eml_rand_shr3cong(unsigned int e_state[2]);
static void genrand_uint32_vector(unsigned int mt[625], unsigned int u[2]);
static void genrandu(unsigned int s, unsigned int *e_state, double *r);
static void twister_state_vector(unsigned int mt[625], double seed);

/* Function Definitions */
static double b_genrandu(unsigned int mt[625])
{
  double r;
  int32_T exitg1;
  unsigned int u[2];
  boolean_T isvalid;
  int k;
  boolean_T exitg2;

  /* ========================= COPYRIGHT NOTICE ============================ */
  /*  This is a uniform (0,1) pseudorandom number generator based on:        */
  /*                                                                         */
  /*  A C-program for MT19937, with initialization improved 2002/1/26.       */
  /*  Coded by Takuji Nishimura and Makoto Matsumoto.                        */
  /*                                                                         */
  /*  Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,      */
  /*  All rights reserved.                                                   */

  /*  All rights reserved.                                                   */
  /*                                                                         */
  /*  Redistribution and use in source and binary forms, with or without     */
  /*  modification, are permitted provided that the following conditions     */
  /*  are met:                                                               */
  /*                                                                         */
  /*    1. Redistributions of source code must retain the above copyright    */
  /*       notice, this list of conditions and the following disclaimer.     */
  /*                                                                         */
  /*    2. Redistributions in binary form must reproduce the above copyright */
  /*       notice, this list of conditions and the following disclaimer      */
  /*       in the documentation and/or other materials provided with the     */
  /*       distribution.                                                     */
  /*                                                                         */
  /*    3. The names of its contributors may not be used to endorse or       */
  /*       promote products derived from this software without specific      */
  /*       prior written permission.                                         */
  /*                                                                         */
  /*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS    */
  /*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT      */
  /*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  */
  /*  A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT  */
  /*  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  */
  /*  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT       */
  /*  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  */
  /*  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  */
  /*  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT    */
  /*  (INCLUDING  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE */
  /*  OF THIS  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  */
  /*                                                                         */
  /* ============================   END   ================================= */
  do {
    exitg1 = 0;
    genrand_uint32_vector(mt, u);
    r = 1.1102230246251565E-16 * ((double)(u[0] >> 5U) * 6.7108864E+7 + (double)
                                  (u[1] >> 6U));
    if (r == 0.0) {
      if ((mt[624] >= 1U) && (mt[624] < 625U)) {
        isvalid = TRUE;
```

```
        isvalid = TRUE;
      } else {
        isvalid = FALSE;
      }

      if (isvalid) {
        isvalid = FALSE;
        k = 1;
        exitg2 = FALSE;
        while ((exitg2 == FALSE) && (k < 625)) {
          if (mt[k - 1] == 0U) {
            k++;
          } else {
            isvalid = TRUE;
            exitg2 = TRUE;
          }
        }
      }

      if (!isvalid) {
        twister_state_vector(mt, 5489.0);
      }
    } else {
      exitg1 = 1;
    }
  } while (exitg1 == 0);

  return r;
}

static double eml_rand_mt19937ar(unsigned int e_state[625])
{
  double r;
  int32_T exitg1;
  unsigned int u32[2];
  int i;
  int ip1;
  static const double dv1[257] = { 0.0, 0.215241895984875, 0.286174591792068,
    0.335737519214422, 0.375121332878378, 0.408389134611989, 0.43751840220787,
    0.335737519214422, 0.375121332878378, 0.408389134611989, 0.43751840220787,
    0.46363433679088, 0.487443966139235, 0.50942332960209, 0.529909720661557,
    0.549151702327164, 0.567338257053817, 0.584616766106378, 0.601104617755991,
    0.61689699000775, 0.63207223638606, 0.646695714894993, 0.660822574244419,
    0.674499822837293, 0.687767892795788, 0.700661841106814, 0.713212285190975,
    0.725446140909999, 0.737387211434295, 0.749056662017815, 0.760473406430107,
    0.771654424224568, 0.782615023307232, 0.793369058840623, 0.80392911698997,
    0.814306670135215, 0.824512208752291, 0.834555354086381, 0.844444954909153,
    0.854189171008163, 0.863795545553308, 0.87327106808886, 0.882622229585165,
    0.891855070732941, 0.900975224461221, 0.909987953496718, 0.91889818364959,
    0.927710533401999, 0.936429340286575, 0.945058684468165, 0.953602409881086,
    0.96206414322304, 0.970447311064224, 0.978755155294224, 0.986990747099062,
    0.99515699963509, 1.00325667954467, 1.01129241744, 1.01926671746548,
    1.02718196603564, 1.03504043983344, 1.04284431314415, 1.05059566459093,
    1.05829648333067, 1.06594867476212, 1.07355406579244, 1.0811144097034,
    1.08863139065398, 1.09610662785202, 1.10354167942464, 1.11093804601357,
    1.11829717411934, 1.12562045921553, 1.13290924865253, 1.14016484436815,
    1.14738850542085, 1.15458145035993, 1.16174485944561, 1.16887987673083,
    1.17598761201545, 1.18306914268269, 1.19012551542669, 1.19715774787944,
    1.20416683014438, 1.2111537262437, 1.21811937548548, 1.22506469375653,
    1.23199057474614, 1.23889789110569, 1.24578749554863, 1.2526602218949,
    1.25951688606371, 1.26635828701823, 1.27318520766536, 1.27999841571382,
    1.28679866449324, 1.29358669373695, 1.30036323033084, 1.30712898903073,
    1.31388467315022, 1.32063097522106, 1.32736857762793, 1.33409815321936,
    1.3408203658964, 1.34753587118059, 1.35424531676263, 1.36094934303328,
    1.36764858359748, 1.37434366577317, 1.38103521107586, 1.38772383568998,
    1.39441015092814, 1.40109476367925, 1.4077782768464, 1.41446128977547,
    1.42114439867531, 1.42782819703026, 1.43451327600589, 1.44120022484872,
    1.44788963128058, 1.45458208188841, 1.46127816251028, 1.46797845861808,
    1.47468355569786, 1.48139403962819, 1.48811049705745, 1.49483351578049,
    1.50156368511546, 1.50830159628131, 1.51504784277671, 1.521803020761,
    1.52856772943771, 1.53534257144151, 1.542128153229, 1.54892508547417,
    1.55573398346918, 1.56255546753104, 1.56939016341512, 1.57623870273591,
    1.58310172339603, 1.58997987002419, 1.59687379442279, 1.60378415602609,
    1.61071162236983, 1.61765686957301, 1.62462058283303, 1.63160345693487,
    1.63860619677555, 1.64562951790478, 1.65267414708306, 1.65974082285818,
    1.66683029616166, 1.67394333092612, 1.68108070472517, 1.68824320943719,
    1.69543165193456, 1.70264685479992, 1.7098896570713, 1.71716091501782,
    1.72446150294804, 1.73179231405296, 1.73915426128591, 1.74654827828172,
```

```
    1.72446150294804, 1.73179231405296, 1.73915426128591, 1.74654827828172,
    1.75397532031767, 1.76143636531891, 1.76893241491127, 1.77646449552452,
    1.78403365954944, 1.79164098655216, 1.79928758454972, 1.80697459135082,
    1.81470317596628, 1.82247454009388, 1.83028991968276, 1.83815058658281,
    1.84605785028518, 1.8540130597602, 1.86201760539967, 1.87007292107127,
    1.878180486293, 1.88634182853678, 1.8945585256707, 1.90283220855043,
    1.91116456377125, 1.91955733659319, 1.92801233405266, 1.93653142827569,
    1.94511656000868, 1.95376974238465, 1.96249306494436, 1.97128869793366,
    1.98015889690048, 1.98910600761744, 1.99813247135842, 2.00724083056053,
    2.0164337349062, 2.02571394786385, 2.03508435372962, 2.04454796521753,
    2.05410793165065, 2.06376754781173, 2.07353026351874, 2.0833996939983,
    2.09337963113879, 2.10347405571488, 2.11368715068665, 2.12402331568952,
    2.13448718284602, 2.14508363404789, 2.15581781987674, 2.16669518035431,
    2.17772146774029, 2.18890277162636, 2.20024554661128, 2.21175664288416,
    2.22344334009251, 2.23531338492992, 2.24737503294739, 2.25963709517379,
    2.27210899022838, 2.28480080272449, 2.29772334890286, 2.31088825060137,
    2.32430801887113, 2.33799614879653, 2.35196722737914, 2.36623705671729,
    2.38082279517208, 2.39574311978193, 2.41101841390112, 2.42667098493715,
    2.44272531820036, 2.4592083743347, 2.47614993967052, 2.49358304127105,
    2.51154444162669, 2.53007523215985, 2.54922155032478, 2.56903545268184,
    2.58957598670829, 2.61091051848882, 2.63311639363158, 2.65628303757674,
    2.68051464328574, 2.70593365612306, 2.73268535904401, 2.76094400527999,
    2.79092117400193, 2.82287739682644, 2.85713873087322, 2.89412105361341,
    2.93436686720889, 2.97860327988184, 3.02783779176959, 3.08352613200214,
    3.147889289518, 3.2245750520478, 3.32024473383983, 3.44927829856143,
    3.65415288536101, 3.91075795952492 };

double u;
static const double dv2[257] = { 1.0, 0.977101701267673, 0.959879091800108,
  0.9451989534423, 0.932060075959231, 0.919991505039348, 0.908726440052131,
  0.898095921898344, 0.887984660755834, 0.878309655808918, 0.869008688036857,
  0.860033621196332, 0.851346258458678, 0.842915653112205, 0.834716292986884,
  0.826726833946222, 0.818929191603703, 0.811307874312656, 0.803849483170964,
  0.796542330422959, 0.789376143566025, 0.782341832654803, 0.775431304981187,
  0.768637315798486, 0.761953346836795, 0.755373506507096, 0.748892447219157,
  0.742505296340151, 0.736207598126863, 0.729995264561476, 0.72386453346863,
  0.717811932630722, 0.711834248878248, 0.705928501332754, 0.700091918136512,
  0.694321916126117, 0.688616083004672, 0.682972161644995, 0.677388036218774,
  0.671861719897082, 0.66639134390875, 0.660975147776663, 0.655611470579697,
  0.650298743110817, 0.645035480820822, 0.639820277453057, 0.634651799287624,
  0.629528779924837, 0.624450015547027, 0.619414360605834, 0.614420723888914,
  0.609468064925773, 0.604555390697468, 0.599681752619125, 0.594846243767987,
  0.590047996332826, 0.585286179263371, 0.580559996100791, 0.575868682972354,
  0.571211506735253, 0.566587763256165, 0.561996775814525, 0.557437893618766,
  0.552910490425833, 0.548413963255266, 0.543947731190026, 0.539511234256952,
  0.535103932380458, 0.530725304403662, 0.526374847171684, 0.522052074672322,
  0.517756517229756, 0.513487720747327, 0.509245245995748, 0.505028667943468,
  0.500837575126149, 0.49667156905249, 0.492530263643869, 0.488413284705458,
  0.484320269426683, 0.480250865909047, 0.476204732719506, 0.47218153846773,
  0.468180961405694, 0.464202689048174, 0.460246417812843, 0.456311852678716,
  0.452398706861849, 0.448506701507203, 0.444635565395739, 0.440785034665804,
  0.436954852547985, 0.433144769112652, 0.429354541029442, 0.425583931338022,
  0.421832709229496, 0.418100649837848, 0.414387534040891, 0.410693148270188,
  0.407017284329473, 0.403359739221114, 0.399720314980197, 0.396098818515832,
  0.392495061459315, 0.388908860018789, 0.385340034840077, 0.381788410873393,
  0.378253817245619, 0.374736087137891, 0.371235057668239, 0.367750569779032,
  0.364282468129004, 0.360830600989648, 0.357394820145781, 0.353974980800077,
  0.350570941481406, 0.347182563956794, 0.343809713146851, 0.340452257044522,
  0.337110066637006, 0.333783015830718, 0.330470981379163, 0.327173842813601,
  0.323891482376391, 0.320623784956905, 0.317370638029914, 0.314131931596337,
  0.310907558126286, 0.307697412504292, 0.30450139197665, 0.301319396100803,
  0.298151326696685, 0.294997087799962, 0.291856585617095, 0.288729728482183,
  0.285616426815502, 0.282516593083708, 0.279430141761638, 0.276356989295668,
  0.273297054068577, 0.270250256365875, 0.267216518343561, 0.264195763997261,
  0.261187919132721, 0.258192911337619, 0.255210669954662, 0.252241126055942,
  0.249284212418529, 0.246339863501264, 0.24340801542275, 0.240488605940501,
  0.237581574431238, 0.23468686187233, 0.231804410824339, 0.228934165414681,
  0.226076071322381, 0.223230075763918, 0.220396127480152, 0.217574176724331,
  0.214764175251174, 0.211966076307031, 0.209179834621125, 0.206405406397881,
  0.203642749310335, 0.200891822494657, 0.198152586545776, 0.195425003514135,
  0.192709036903589, 0.190004651670465, 0.187311814223801, 0.1846304924268,
  0.181960655599523, 0.179302274522848, 0.176655321443735, 0.174019770081839,
  0.171395595637506, 0.168782774801212, 0.166181285764482, 0.163591108232366,
  0.161012223437511, 0.158444614155925, 0.15588826472448, 0.153343161060263,
  0.150809290681846, 0.148286642732575, 0.145775208005994, 0.143274978973514,
  0.140785949814445, 0.138308116448551, 0.135841476571254, 0.133386029691669,
  0.130941777173644, 0.12850872228, 0.126086870220186, 0.123676228201597,
```

```
    0.130941777173644, 0.12850872228, 0.126086870220186, 0.123676228201597,
    0.12127680548479, 0.11888861344291, 0.116511665625611, 0.114145977827839,
    0.111791568163838, 0.109448457146812, 0.107116667774684, 0.104796225622487,
    0.102487158941935, 0.10018949876881, 0.0979032790388625, 0.095628536713009,
    0.093365311912691, 0.0911136480663738, 0.0888735920682759,
    0.0866451944505581, 0.0844285095703535, 0.082223595813203,
    0.0800305158146631, 0.0778493367020961, 0.0756801303589272,
    0.0735229737139814, 0.0713779490588905, 0.0692451443970068,
    0.0671246538277886, 0.065016577971243, 0.0629210244377582, 0.06083810834954,
    0.0587679529209339, 0.0567106901062031, 0.0546664613248891,
    0.0526354182767924, 0.0506177238609479, 0.0486135532158687,
    0.0466230949019305, 0.0446465522512946, 0.0426841449164746,
    0.0407361106559411, 0.0388027074045262, 0.0368842156885674,
    0.0349809414617162, 0.0330932194585786, 0.0312214171919203,
    0.0293659397581334, 0.0275272356696031, 0.0257058040085489,
    0.0239022033057959, 0.0221170627073089, 0.0203510962300445,
    0.0186051212757247, 0.0168800831525432, 0.0151770883079353,
    0.0134974506017399, 0.0118427578579079, 0.0102149714397015,
    0.00861658276939875, 0.00705087547137324, 0.00552240329925101,
    0.00403797259336304, 0.00260907274610216, 0.0012602859304986,
    0.000477467764609386 };

  double x;
  do {
    exitg1 = 0;
    genrand_uint32_vector(e_state, u32);
    i = (int)((u32[1] >> 24U) + 1U);
    ip1 = (int)(i + 1U) - 1;
    r = (((double)(u32[0] >> 3U) * 1.6777216E+7 + (double)((int)u32[1] &
           16777215)) * 2.2204460492503131E-16 - 1.0) * dv1[ip1];
    if (fabs(r) <= dv1[i - 1]) {
      exitg1 = 1;
    } else if (i < 256) {
      u = b_genrandu(e_state);
      if (dv2[ip1] + u * (dv2[i - 1] - dv2[ip1]) < exp(-0.5 * r * r)) {
        exitg1 = 1;
      }
    } else {
      do {
        do {
          u = b_genrandu(e_state);
          x = log(u) * 0.273661237329758;
          u = b_genrandu(e_state);
        } while (!(-2.0 * log(u) > x * x));

        if (r < 0.0) {
          r = x - 3.65415288536101;
        } else {
          r = 3.65415288536101 - x;
        }

        exitg1 = 1;
      }
    }
  } while (exitg1 == 0);

  return r;
}

static double eml_rand_shr3cong(unsigned int e_state[2])
{
  double r;
  unsigned int icng;
  unsigned int jsr;
  unsigned int ui;
  int j;
  int jp1;
  static const double dv0[65] = { 0.340945, 0.4573146, 0.5397793, 0.6062427,
    0.6631691, 0.7136975, 0.7596125, 0.8020356, 0.8417227, 0.8792102, 0.9148948,
    0.9490791, 0.9820005, 1.0138492, 1.044781, 1.0749254, 1.1043917, 1.1332738,
    1.161653, 1.189601, 1.2171815, 1.2444516, 1.2714635, 1.298265, 1.3249008,
    1.3514125, 1.3778399, 1.4042211, 1.4305929, 1.4569915, 1.4834527, 1.5100122,
    1.5367061, 1.5635712, 1.5906454, 1.617968, 1.6455802, 1.6735255, 1.7018503,
    1.7306045, 1.7598422, 1.7896223, 1.8200099, 1.851077, 1.8829044, 1.9155831,
    1.9492166, 1.9839239, 2.0198431, 2.0571356, 2.095993, 2.136645, 2.1793713,
    2.2245175, 2.2725186, 2.3239338, 2.3795008, 2.4402218, 2.5075117, 2.5834658,
    2.6713916, 2.7769942, 2.7769942, 2.7769942, 2.7769942 };

  double x;
```

```
  double x;
  double y;
  double s;
  icng = 69069U * e_state[0] + 1234567U;
  jsr = e_state[1] ^ e_state[1] << 13;
  jsr ^= jsr >> 17;
  jsr ^= jsr << 5;
  ui = icng + jsr;
  j = (int)((ui & 63U) + 1U) - 1;
  jp1 = (int)((j + 1) + 1U) - 1;
  r = (double)(int)ui * 4.6566128730773926E-10 * dv0[jp1];
  if (fabs(r) <= dv0[j]) {
  } else {
    x = (fabs(r) - dv0[j]) / (dv0[jp1] - dv0[j]);
    icng = 69069U * icng + 1234567U;
    jsr ^= jsr << 13;
    jsr ^= jsr >> 17;
    jsr ^= jsr << 5;
    y = (double)(int)(icng + jsr) * 2.328306436538696E-10;
    s = x + (0.5 + y);
    if (s > 1.301198) {
      if (r < 0.0) {
        r = 0.4878992 * x - 0.4878992;
      } else {
        r = 0.4878992 - 0.4878992 * x;
      }
    } else if (s <= 0.9689279) {
    } else {
      s = 0.4878992 * x;
      x = 0.4878992 - 0.4878992 * x;
      if (0.5 + y > 12.67706 - 12.37586 * exp(-0.5 * (0.4878992 - s) * x)) {
        if (r < 0.0) {
          r = -(0.4878992 - s);
        } else {
          r = 0.4878992 - s;
        }
      } else if (exp(-0.5 * dv0[jp1] * dv0[jp1]) + (0.5 + y) * 0.01958303 /
                 dv0[jp1] <= exp(-0.5 * r * r)) {
      } else {
      } else {
        do {
          icng = 69069U * icng + 1234567U;
          jsr ^= jsr << 13;
          jsr ^= jsr >> 17;
          jsr ^= jsr << 5;
          x = log(0.5 + (double)(int)(icng + jsr) * 2.328306436538696E-10) /
            2.776994;
          icng = 69069U * icng + 1234567U;
          jsr ^= jsr << 13;
          jsr ^= jsr >> 17;
          jsr ^= jsr << 5;
        } while (!(-2.0 * log(0.5 + (double)(int)(icng + jsr) *
                              2.328306436538696E-10) > x * x));

        if (r < 0.0) {
          r = x - 2.776994;
        } else {
          r = 2.776994 - x;
        }
      }
    }
  }

  e_state[0] = icng;
  e_state[1] = jsr;
  return r;
}

static void genrand_uint32_vector(unsigned int mt[625], unsigned int u[2])
{
  int i;
  unsigned int mti;
  int kk;
  unsigned int y;
  unsigned int b_y;
  unsigned int c_y;
  unsigned int d_y;
  for (i = 0; i < 2; i++) {
```

```
  for (i = 0; i < 2; i++) {
    u[i] = 0U;
  }

  for (i = 0; i < 2; i++) {
    mti = mt[624] + 1U;
    if (mti >= 625U) {
      for (kk = 0; kk < 227; kk++) {
        y = (mt[kk] & 2147483648U) | (mt[1 + kk] & 2147483647U);
        if ((int)(y & 1U) == 0) {
          b_y = y >> 1U;
        } else {
          b_y = y >> 1U ^ 2567483615U;
        }

        mt[kk] = mt[397 + kk] ^ b_y;
      }

      for (kk = 0; kk < 396; kk++) {
        y = (mt[227 + kk] & 2147483648U) | (mt[228 + kk] & 2147483647U);
        if ((int)(y & 1U) == 0) {
          c_y = y >> 1U;
        } else {
          c_y = y >> 1U ^ 2567483615U;
        }

        mt[227 + kk] = mt[kk] ^ c_y;
      }

      y = (mt[623] & 2147483648U) | (mt[0] & 2147483647U);
      if ((int)(y & 1U) == 0) {
        d_y = y >> 1U;
      } else {
        d_y = y >> 1U ^ 2567483615U;
      }

      mt[623] = mt[396] ^ d_y;
      mti = 1U;
    }

    y = mt[(int)mti - 1];
    mt[624] = mti;
    y ^= y >> 11U;
    y ^= y << 7U & 2636928640U;
    y ^= y << 15U & 4022730752U;
    y ^= y >> 18U;
    u[i] = y;
  }
}

static void genrandu(unsigned int s, unsigned int *e_state, double *r)
{
  int hi;
  unsigned int test1;
  unsigned int test2;
  hi = (int)(s / 127773U);
  test1 = 16807U * (s - hi * 127773U);
  test2 = 2836U * hi;
  if (test1 < test2) {
    *e_state = (test1 - test2) + 2147483647U;
  } else {
    *e_state = test1 - test2;
  }

  *r = (double)*e_state * 4.6566128752457969E-10;
}

static void twister_state_vector(unsigned int mt[625], double seed)
{
  unsigned int r;
  int mti;
  if (seed < 4.294967296E+9) {
    if (seed >= 0.0) {
      r = (unsigned int)seed;
    } else {
      r = 0U;
    }
```

```
    }
  } else if (seed >= 4.294967296E+9) {
    r = MAX_uint32_T;
  } else {
    r = 0U;
  }

  mt[0] = r;
  for (mti = 0; mti < 623; mti++) {
    r = (r ^ r >> 30U) * 1812433253U + (1 + mti);
    mt[1 + mti] = r;
  }

  mt[624] = 624U;
}

void randn(double r[100000])
{
  int k;
  double b_r;
  unsigned int e_state;
  double t;
  double c_r;
  unsigned int f_state;
  if (method == 0U) {
    if (b_method == 4U) {
      for (k = 0; k < 100000; k++) {
        do {
          genrandu(b_state, &e_state, &b_r);
          genrandu(e_state, &b_state, &t);
          c_r = 2.0 * b_r - 1.0;
          t = 2.0 * t - 1.0;
          t = t * t + c_r * c_r;
        } while (!(t <= 1.0));

        r[k] = (2.0 * b_r - 1.0) * sqrt(-2.0 * log(t) / t);
      }
    } else if (b_method == 5U) {
      for (k = 0; k < 100000; k++) {
```

```
      for (k = 0; k < 100000; k++) {
        r[k] = eml_rand_shr3cong(c_state);
      }
    } else {
      if (!state_not_empty) {
        memset(&d_state[0], 0, 625U * sizeof(unsigned int));
        twister_state_vector(d_state, 5489.0);
        state_not_empty = TRUE;
      }

      for (k = 0; k < 100000; k++) {
        r[k] = eml_rand_mt19937ar(d_state);
      }
    }
  } else if (method == 4U) {
    for (k = 0; k < 100000; k++) {
      e_state = state[0];
      do {
        genrandu(e_state, &f_state, &b_r);
        genrandu(f_state, &e_state, &t);
        c_r = 2.0 * b_r - 1.0;
        t = 2.0 * t - 1.0;
        t = t * t + c_r * c_r;
      } while (!(t <= 1.0));

      state[0] = e_state;
      r[k] = (2.0 * b_r - 1.0) * sqrt(-2.0 * log(t) / t);
    }
  } else {
    for (k = 0; k < 100000; k++) {
      r[k] = eml_rand_shr3cong(state);
    }
  }
}
```

randn.h

```c
/*
 * randn.h
 */

#ifndef __RANDN_H__
#define __RANDN_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void randn(double r[100000]);
#endif
```

D.3 Función seno

sin.c

```c
/*
 * sin.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "sin.h"

/* Function Definitions */
void b_sin(double x[100000])
{
  int k;
  for (k = 0; k < 100000; k++) {
    x[k] = sin(x[k]);
  }
}
```

sin.h

```c
/*
 * sin.h
 */

#ifndef __SIN_H__
#define __SIN_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void b_sin(double x[100000]);
#endif
```

D.4 Inicialización de la función

Pruebac_initialize.c

```c
/*
 * Pruebac_initialize.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "Pruebac_initialize.h"
#include "eml_randn.h"
#include "eml_rand.h"
#include "eml_rand_shr3cong_stateful.h"
#include "eml_rand_mcg16807_stateful.h"
#include "eml_rand_mt19937ar_stateful.h"

/* Function Definitions */
void Pruebac_initialize(void)
{
  rt_InitInfAndNaN(8U);
  state_not_empty_init();
  eml_rand_mcg16807_stateful_init();
  eml_rand_shr3cong_stateful_init();
  eml_rand_init();
  eml_randn_init();
}
```

Pruebac_initialize.h

```c
/*
 * Pruebac_initialize.h
 */

#ifndef __PRUEBAC_INITIALIZE_H__
#define __PRUEBAC_INITIALIZE_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void Pruebac_initialize(void);
#endif
```

D.5 Finalización de la función

Pruebac_terminate.c

```c
/*
 * Pruebac_terminate.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "Pruebac_terminate.h"

/* Function Definitions */
void Pruebac_terminate(void)
{
  /* (no terminate code required) */
}
```

Pruebac_terminate.h

```c
/*
 * Pruebac_terminate.h
 */

#ifndef __PRUEBAC_TERMINATE_H__
#define __PRUEBAC_TERMINATE_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void Pruebac_terminate(void);
#endif
```

D.6 Códigos complementarios para el generador de números aleatorios

eml_rand.c

```c
/*
 * eml_rand.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "eml_rand.h"
#include "Pruebac_data.h"

/* Function Definitions */
void eml_rand_init(void)
{
  b_method = 7U;
}
```

eml_rand.h

```c
/*
 * eml_rand.h
 */

#ifndef __EML_RAND_H__
#define __EML_RAND_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void eml_rand_init(void);
#endif
```

eml_rand_shr3cong_stateful.c

```c
/*
 * eml_rand_shr3cong_stateful.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "eml_rand_shr3cong_stateful.h"
#include "Pruebac_data.h"

/* Function Definitions */
void eml_rand_shr3cong_stateful_init(void)
{
  int i0;
  for (i0 = 0; i0 < 2; i0++) {
    c_state[i0] = 362436069U + 158852560U * i0;
  }
}
```

eml_rand_shr3cong_stateful.h

```c
/*
 * eml_rand_shr3cong_stateful.h
 */

#ifndef __EML_RAND_SHR3CONG_STATEFUL_H__
#define __EML_RAND_SHR3CONG_STATEFUL_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void eml_rand_shr3cong_stateful_init(void);
#endif
```

eml_rand_mt19937_stateful.c

```c
/*
 * eml_rand_mt19937ar_stateful.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "eml_rand_mt19937ar_stateful.h"
#include "Pruebac_data.h"

/* Function Definitions */
void state_not_empty_init(void)
{
  state_not_empty = FALSE;
}
```

eml_rand_mt19937_stateful.h

```c
/*
 * eml_rand_mt19937ar_stateful.h
 */

#ifndef __EML_RAND_MT19937AR_STATEFUL_H__
#define __EML_RAND_MT19937AR_STATEFUL_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void state_not_empty_init(void);
#endif
```

eml_rand_mcg16807_stateful.c

```c
/*
 * eml_rand_mcg16807_stateful.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "eml_rand_mcg16807_stateful.h"
#include "Pruebac_data.h"

/* Function Definitions */
void eml_rand_mcg16807_stateful_init(void)
{
  b_state = 1144108930U;
}
```

eml_rand_mt19937_stateful.h

```c
/*
 * eml_rand_mcg16807_stateful.h
 */

#ifndef __EML_RAND_MCG16807_STATEFUL_H__
#define __EML_RAND_MCG16807_STATEFUL_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void eml_rand_mcg16807_stateful_init(void);
#endif
```

eml_rand.c

```c
/*
 * eml_randn.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "eml_randn.h"
#include "Pruebac_data.h"

/* Function Definitions */
void eml_randn_init(void)
{
  int i;
  static const unsigned int uv0[2] = { 362436069U, 0U };

  method = 0U;
  for (i = 0; i < 2; i++) {
    state[i] = uv0[i];
  }

  if (state[1] == 0U) {
    state[1] = 521288629U;
  }
}
```

eml_rand.h

```c
/*
 * eml_randn.h
 */

#ifndef __EML_RANDN_H__
#define __EML_RANDN_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void eml_randn_init(void);
#endif
```

D.7 Linspace

linspace.c

```c
/*
 * linspace.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "linspace.h"

/* Function Definitions */
void linspace(double d1, double d2, double y[100000])
{
  double delta1;
  double delta2;
  int k;
  y[99999] = d2;
  y[0] = d1;
  if ((((d1 < 0.0) != (d2 < 0.0)) && (fabs(d1) > 8.9884656743115785E+307)) ||
      (fabs(d2) > 8.9884656743115785E+307)) {
    delta1 = d1 / 99999.0;
    delta2 = d2 / 99999.0;
    for (k = 0; k < 99998; k++) {
      y[1 + k] = (d1 + delta2 * (1.0 + (double)k)) - delta1 * (1.0 + (double)k);
    }
  } else {
    delta1 = (d2 - d1) / 99999.0;
    for (k = 0; k < 99998; k++) {
      y[1 + k] = d1 + (1.0 + (double)k) * delta1;
    }
  }
}
```

linspace.h

```c
/*
 * linspace.h
 */

#ifndef __LINSPACE_H__
#define __LINSPACE_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void linspace(double d1, double d2, double y[100000]);
#endif
```

D.8 Obtención de datos de la función principal Pruebac

Pruebac_data.c

```c
/*
 * Pruebac_data.c
 */

/* Include files */
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "Pruebac_data.h"

/* Variable Definitions */
unsigned int method;
unsigned int state[2];
unsigned int b_method;
unsigned int b_state;
unsigned int c_state[2];
boolean_T state_not_empty;
```

Pruebac_data.h

```c
/*
 * Pruebac_data.h
 */

#ifndef __PRUEBAC_DATA_H__
#define __PRUEBAC_DATA_H__

/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Variable Declarations */
extern unsigned int method;
extern unsigned int state[2];
extern unsigned int b_method;
extern unsigned int b_state;
extern unsigned int c_state[2];
extern boolean_T state_not_empty;
#endif
```

D.9 Función para inicializar no-finitos (Inf, NaN and -Inf) y funciones para cada uno de ellos

rt_nonfinite.c

```c
/*
 * rt_nonfinite.c
 */

/*
 * Abstract:
 *      MATLAB for code generation function to initialize non-finites,
 *      (Inf, NaN and -Inf).
 */
#include "rt_nonfinite.h"
#include "rtGetNaN.h"
#include "rtGetInf.h"

real_T rtInf;
real_T rtMinusInf;
real_T rtNaN;
real32_T rtInfF;
real32_T rtMinusInfF;
real32_T rtNaNF;

/* Function: rt_InitInfAndNaN ==========================================================
 * Abstract:
 * Initialize the rtInf, rtMinusInf, and rtNaN needed by the
 * generated code. NaN is initialized as non-signaling. Assumes IEEE.
 */
void rt_InitInfAndNaN(size_t realSize)
{
  (void) (realSize);
  rtNaN = rtGetNaN();
  rtNaNF = rtGetNaNF();
  rtInf = rtGetInf();
  rtInfF = rtGetInfF();
  rtMinusInf = rtGetMinusInf();
  rtMinusInfF = rtGetMinusInfF();
}

/* Function: rtIsInf ===================================================
 * Abstract:
 * Test if value is infinite
 * Test if value is infinite
 */
boolean_T rtIsInf(real_T value)
{
  return ((value==rtInf || value==rtMinusInf) ? 1U : 0U);
}

/* Function: rtIsInfF ==================================================
 * Abstract:
 * Test if single-precision value is infinite
 */
boolean_T rtIsInfF(real32_T value)
{
  return(((value)==rtInfF || (value)==rtMinusInfF) ? 1U : 0U);
}

/* Function: rtIsNaN ===================================================
 * Abstract:
 * Test if value is not a number
 */
boolean_T rtIsNaN(real_T value)
{
#if defined(_MSC_VER) && (_MSC_VER <= 1200)
  return _isnan(value)? TRUE:FALSE;
#else
  return (value!=value)? 1U:0U;
#endif
}

/* Function: rtIsNaNF ==================================================
 * Abstract:
 * Test if single-precision value is not a number
 */
boolean_T rtIsNaNF(real32_T value)
{
#if defined(_MSC_VER) && (_MSC_VER <= 1200)
  return _isnan((real_T)value)? true:false;
#else
  return (value!=value)? 1U:0U;
```

```
  return (value!=value)? 1U:0U;
#endif
}
```

rt_nonfinite.h

```
/*
 * rt_nonfinite.h
 */

#ifndef __RT_NONFINITE_H__
#define __RT_NONFINITE_H__

#if defined(_MSC_VER) && (_MSC_VER <= 1200)
#include <float.h>
#endif
#include <stddef.h>
#include "rtwtypes.h"

extern real_T rtInf;
extern real_T rtMinusInf;
extern real_T rtNaN;
extern real32_T rtInfF;
extern real32_T rtMinusInfF;
extern real32_T rtNaNF;
extern void rt_InitInfAndNaN(size_t realSize);
extern boolean_T rtIsInf(real_T value);
extern boolean_T rtIsInfF(real32_T value);
extern boolean_T rtIsNaN(real_T value);
extern boolean_T rtIsNaNF(real32_T value);

typedef struct {
  struct {
    uint32_T wordH;
    uint32_T wordL;
  } words;
} BigEndianIEEEDouble;

typedef struct {
  struct {
    uint32_T wordL;
    uint32_T wordH;
  } words;
} LittleEndianIEEEDouble;
```

```
} LittleEndianIEEEDouble;

typedef struct {
  union {
    real32_T wordLreal;
    uint32_T wordLuint;
  } wordL;
} IEEESingle;

#endif
```

rtGetInf.c

```c
/*
 * rtGetInf.c
 */

/*
 * Abstract:
 *       MATLAB for code generation function to initialize non-finite, Inf and MinusInf
 */
#include "rtGetInf.h"
#define NumBitsPerChar  8U

/* Function: rtGetInf ================================================
 * Abstract:
 * Initialize rtInf needed by the generated code.
 * Inf is initialized as non-signaling. Assumes IEEE.
 */
real_T rtGetInf(void)
{
  size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
  real_T inf = 0.0;
  if (bitsPerReal == 32U) {
    inf = rtGetInfF();
  } else {
    uint16_T one = 1U;
    enum {
      LittleEndian,
      BigEndian
    } machByteOrder = (*((uint8_T *) &one) == 1U) ? LittleEndian : BigEndian;
    switch (machByteOrder) {
     case LittleEndian:
      {
        union {
          LittleEndianIEEEDouble bitVal;
          real_T fltVal;
        } tmpVal;

        tmpVal.bitVal.words.wordH = 0x7FF00000U;
        tmpVal.bitVal.words.wordL = 0x00000000U;
        inf = tmpVal.fltVal;
        inf = tmpVal.fltVal;
        break;
      }

     case BigEndian:
      {
        union {
          BigEndianIEEEDouble bitVal;
          real_T fltVal;
        } tmpVal;

        tmpVal.bitVal.words.wordH = 0x7FF00000U;
        tmpVal.bitVal.words.wordL = 0x00000000U;
        inf = tmpVal.fltVal;
        break;
      }
    }
  }

  return inf;
}

/* Function: rtGetInfF ===============================================
 * Abstract:
 * Initialize rtInfF needed by the generated code.
 * Inf is initialized as non-signaling. Assumes IEEE.
 */
real32_T rtGetInfF(void)
{
  IEEESingle infF;
  infF.wordL.wordLuint = 0x7F800000U;
  return infF.wordL.wordLreal;
}

/* Function: rtGetMinusInf ===========================================
 * Abstract:
 * Initialize rtMinusInf needed by the generated code.
 * Inf is initialized as non-signaling. Assumes IEEE.
 */
```

```
 */
real_T rtGetMinusInf(void)
{
  size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
  real_T minf = 0.0;
  if (bitsPerReal == 32U) {
    minf = rtGetMinusInfF();
  } else {
    uint16_T one = 1U;
    enum {
      LittleEndian,
      BigEndian
    } machByteOrder = (*((uint8_T *) &one) == 1U) ? LittleEndian : BigEndian;
    switch (machByteOrder) {
     case LittleEndian:
      {
        union {
          LittleEndianIEEEDouble bitVal;
          real_T fltVal;
        } tmpVal;

        tmpVal.bitVal.words.wordH = 0xFFF00000U;
        tmpVal.bitVal.words.wordL = 0x00000000U;
        minf = tmpVal.fltVal;
        break;
      }

     case BigEndian:
      {
        union {
          BigEndianIEEEDouble bitVal;
          real_T fltVal;
        } tmpVal;

        tmpVal.bitVal.words.wordH = 0xFFF00000U;
        tmpVal.bitVal.words.wordL = 0x00000000U;
        minf = tmpVal.fltVal;
        break;
      }
```

```
      }
    }
  }

  return minf;
}

/* Function: rtGetMinusInfF ================================================
 * Abstract:
 * Initialize rtMinusInfF needed by the generated code.
 * Inf is initialized as non-signaling. Assumes IEEE.
 */
real32_T rtGetMinusInfF(void)
{
  IEEESingle minfF;
  minfF.wordL.wordLuint = 0xFF800000U;
  return minfF.wordL.wordLreal;
}
```

rtGetInf.h

```
/*
 * rtGetInf.h
 */

#ifndef __RTGETINF_H__
#define __RTGETINF_H__

#include <stddef.h>
#include "rtwtypes.h"
#include "rt_nonfinite.h"

extern real_T rtGetInf(void);
extern real32_T rtGetInfF(void);
extern real_T rtGetMinusInf(void);
extern real32_T rtGetMinusInfF(void);

#endif
```

rtGetNaN.C

```c
/*
 * rtGetNaN.c
 */

/*
 * Abstract:
 *       MATLAB for code generation function to initialize non-finite, NaN
 */
#include "rtGetNaN.h"
#define NumBitsPerChar  8U

/* Function: rtGetNaN ================================================
 * Abstract:
 * Initialize rtNaN needed by the generated code.
 * NaN is initialized as non-signaling. Assumes IEEE.
 */
real_T rtGetNaN(void)
{
  size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
  real_T nan = 0.0;
  if (bitsPerReal == 32U) {
    nan = rtGetNaNF();
  } else {
  uint16_T one = 1U;
  enum {
    LittleEndian,
    BigEndian
  } machByteOrder = (*((uint8_T *) &one) == 1U) ? LittleEndian : BigEndian;
  switch (machByteOrder) {
   case LittleEndian:
    {
      union {
        LittleEndianIEEEDouble bitVal;
        real_T fltVal;
      } tmpVal;

      tmpVal.bitVal.words.wordH = 0xFFF80000U;
      tmpVal.bitVal.words.wordL = 0x00000000U;
      nan = tmpVal.fltVal;
      nan = tmpVal.fltVal;
      break;
    }

   case BigEndian:
    {
        union {
          BigEndianIEEEDouble bitVal;
          real_T fltVal;
        } tmpVal;

        tmpVal.bitVal.words.wordH = 0x7FFFFFFFU;
        tmpVal.bitVal.words.wordL = 0xFFFFFFFFU;
        nan = tmpVal.fltVal;
        break;
      }
    }
  }

  return nan;
}

/* Function: rtGetNaNF ================================================
 * Abstract:
 * Initialize rtNaNF needed by the generated code.
 * NaN is initialized as non-signaling. Assumes IEEE.
 */
real32_T rtGetNaNF(void)
{
  IEEESingle nanF = { { 0 } };
  uint16_T one = 1U;
  enum {
    LittleEndian,
    BigEndian
  } machByteOrder = (*((uint8_T *) &one) == 1U) ? LittleEndian : BigEndian;
  switch (machByteOrder) {
   case LittleEndian:
    {
      nanF.wordL.wordLuint = 0xFFC00000U;
```

```
77      nanF.wordL.wordLuint = 0xFFC00000U;
78      break;
79    }
80
81   case BigEndian:
82    {
83      nanF.wordL.wordLuint = 0x7FFFFFFFU;
84      break;
85    }
86  }
87
88  return nanF.wordL.wordLreal;
89}
```

rtGetNan.h

```
1 /*
2  * rtGetNaN.h
3  */
4
5 #ifndef __RTGETNAN_H__
6 #define __RTGETNAN_H__
7
8 #include <stddef.h>
9 #include "rtwtypes.h"
10 #include "rt_nonfinite.h"
11
12 extern real_T rtGetNaN(void);
13 extern real32_T rtGetNaNF(void);
14
15 #endif
```

D.10 Definición de estructuras

Pruebac_types.h

```
1 /*
2  * Pruebac_types.h
3  */
4
5 #ifndef __PRUEBAC_TYPES_H__
6 #define __PRUEBAC_TYPES_H__
7
8 /* Include files */
9 #include "rtwtypes.h"
10
11 /* Type Definitions */
12 #ifndef typedef_b_struct_T
13 #define typedef_b_struct_T
14 typedef struct
15 {
16     unsigned int alpha;
17     unsigned int rows;
18 } b_struct_T;
19 #endif /*typedef_b_struct_T*/
20 #ifndef typedef_struct_T
21 #define typedef_struct_T
22 typedef struct
23 {
24     boolean_T CaseSensitivity;
25     boolean_T PartialMatching;
26     boolean_T StructExpand;
27 } struct_T;
28 #endif /*typedef_struct_T*/
29
30 #endif
```

D.11 Definición de tipos de datos

rtwTypes.h

```
/*
 * rtwtypes.h
 */

#ifndef __RTWTYPES_H__
#define __RTWTYPES_H__
#ifndef TRUE
# define TRUE (1U)
#endif
#ifndef FALSE
# define FALSE (0U)
#endif
#ifndef __TMWTYPES__
#define __TMWTYPES__

#include <limits.h>

/*=======================================================================*
 * Target hardware information
 *   Device type: Generic->MATLAB Host Computer
 *   Number of bits:     char:   8    short:   16    int:  32
 *                       long:  32    long long:  64
 *                       native word size:  32
 *   Byte ordering: LittleEndian
 *   Signed integer division rounds to: Zero
 *   Shift right on a signed integer as arithmetic shift: on
 *=======================================================================*/

/*=======================================================================*
 * Fixed width word size data types:                                     *
 *   int8_T, int16_T, int32_T     - signed 8, 16, or 32 bit integers     *
 *   uint8_T, uint16_T, uint32_T  - unsigned 8, 16, or 32 bit integers   *
 *   real32_T, real64_T           - 32 and 64 bit floating point numbers *
 *=======================================================================*/

typedef signed char int8_T;
typedef unsigned char uint8_T;
typedef short int16_T;
typedef unsigned short uint16_T;
typedef unsigned short uint16_T;
typedef int int32_T;
typedef unsigned int uint32_T;
typedef long long int64_T;
typedef unsigned long long uint64_T;
typedef float real32_T;
typedef double real64_T;

/*=======================================================================*
 * Generic type definitions: real_T, time_T, boolean_T, int_T, uint_T,   *
 *                           ulong_T, ulonglong_T, char_T and byte_T.    *
 *=======================================================================*/

typedef double real_T;
typedef double time_T;
typedef unsigned char boolean_T;
typedef int int_T;
typedef unsigned int uint_T;
typedef unsigned long ulong_T;
typedef unsigned long long ulonglong_T;
typedef char char_T;
typedef char_T byte_T;

/*=======================================================================*
 * Complex number type definitions                                       *
 *=======================================================================*/
#define CREAL_T
   typedef struct {
      real32_T re;
      real32_T im;
   } creal32_T;

   typedef struct {
      real64_T re;
      real64_T im;
   } creal64_T;

   typedef struct {
      real_T re;
```

```
    real_T re;
    real_T im;
} creal_T;

typedef struct {
    int8_T re;
    int8_T im;
} cint8_T;

typedef struct {
    uint8_T re;
    uint8_T im;
} cuint8_T;

typedef struct {
    int16_T re;
    int16_T im;
} cint16_T;

typedef struct {
    uint16_T re;
    uint16_T im;
} cuint16_T;

typedef struct {
    int32_T re;
    int32_T im;
} cint32_T;

typedef struct {
    uint32_T re;
    uint32_T im;
} cuint32_T;

typedef struct {
    int64_T re;
    int64_T im;
} cint64_T;

typedef struct {
    uint64_T re;
    uint64_T im;
} cuint64_T;


/*=======================================================================*
 * Min and Max:                                                          *
 *   int8_T, int16_T, int32_T     - signed 8, 16, or 32 bit integers     *
 *   uint8_T, uint16_T, uint32_T  - unsigned 8, 16, or 32 bit integers   *
 *=======================================================================*/

#define MAX_int8_T      ((int8_T)(127))
#define MIN_int8_T      ((int8_T)(-128))
#define MAX_uint8_T     ((uint8_T)(255))
#define MIN_uint8_T     ((uint8_T)(0))
#define MAX_int16_T     ((int16_T)(32767))
#define MIN_int16_T     ((int16_T)(-32768))
#define MAX_uint16_T    ((uint16_T)(65535))
#define MIN_uint16_T    ((uint16_T)(0))
#define MAX_int32_T     ((int32_T)(2147483647))
#define MIN_int32_T     ((int32_T)(-2147483647-1))
#define MAX_uint32_T    ((uint32_T)(0xFFFFFFFFU))
#define MIN_uint32_T    ((uint32_T)(0))
#define MAX_int64_T ((int64_T)(9223372036854775807LL))
#define MIN_int64_T ((int64_T)(-9223372036854775807LL-1LL))
#define MAX_uint64_T    ((uint64_T)(0xFFFFFFFFFFFFFFFFULL))
#define MIN_uint64_T    ((uint64_T)(0ULL))

/* Logical type definitions */
#if !defined(__cplusplus) && !defined(__true_false_are_keywords)
#  ifndef false
#   define false (0U)
#  endif
#  ifndef true
#   define true (1U)
#  endif
#endif
```

```
#endif

/*
 * MATLAB for code generation assumes the code is compiled on a target using a 2's compliment representation
 * for signed integer values.
 */
#if ((SCHAR_MIN + 1) != -SCHAR_MAX)
#error "This code must be compiled using a 2's complement representation for signed integer values"
#endif

/*
 * Maximum length of a MATLAB identifier (function/variable)
 * including the null-termination character. Referenced by
 * rt_logging.c and rt_matrx.c.
 */
#define TMW_NAME_LENGTH_MAX 64

#endif
#endif
```

D.12 Función principal Pruebac

Puebac.c

```
/*
 * Pruebac.c
 *
 *
 */

/* Include files */
#include <xdc/std.h>
#include <xdc/runtime/System.h>
#include "rt_nonfinite.h"
#include "Pruebac.h"
#include "corrcoef.h"
#include "sin.h"
#include "linspace.h"
#include "randn.h"

/* Function Definitions */
void Pruebac(double R[4])
{
  static double x[100000];
  static double t[100000];
  int j;
  double w[512];
  double xin[512];
  int i;
  static double y[100000];
  double mu;
  double a;
  randn(x);
  linspace(0.0, 2.2675736961451248E-5, t);
  for (j = 0; j < 100000; j++) {
    t[j] *= 277088.47204661975;
  }

  b_sin(t);
  for (i = 0; i < 512; i++) {
    w[i] = 0.0;
    xin[i] = 0.0;
  }
```

```
  }

  memset(&y[0], 0, 100000U * sizeof(double));
  for (i = 0; i < 100000; i++) {
    for (j = 0; j < 511; j++) {
      xin[511 - j] = xin[510 - j];
    }

    xin[0] = x[i];
    mu = 0.0;
    for (j = 0; j < 512; j++) {
      mu += w[j] * xin[j];
    }

    y[i] = mu;
    mu = 0.0;
    for (j = 0; j < 512; j++) {
      mu += xin[j] * xin[j];
    }

    mu = 0.99 / (8.0E-28 + mu);
    a = 1.0 - mu * 0.02;
    mu = 2.0 * mu * (t[i] - y[i]);
    for (j = 0; j < 512; j++) {
      w[j] = a * w[j] + mu * xin[j];
    }
  }
```

Pruebac.h

```
/*
 * Pruebac.h
 */

#ifndef __PRUEBAC_H__
#define __PRUEBAC_H__
/* Include files */
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>

#include "rtwtypes.h"
#include "Pruebac_types.h"

/* Function Declarations */
extern void Pruebac(double R[4]);
#endif
```

D.13 main.c

```c
/*
 * main.c
 */

#include <xdc/std.h>
#include <xdc/runtime/System.h>
#include "Pruebac.h"

 double R[4]={0,0,0,0};

int main(void)
{
    R[0] =0.0f;
    Pruebac(R);
    return 0;
}
```

D.14 Configuración de la memoria de la TMS320C6748 LCDK

```
/*****************************************************************************/
/*  C6748.cmd                                                                */
/*  Copyright (c) 2010 Texas Instruments Incorporated                        */
/*  Author: Rafael de Souza                                                  */
/*                                                                           */
/*    Description: This file is a sample linker command file that can be     */
/*                 used for linking programs built with the C compiler and   */
/*                 running the resulting .out file on a C6748                */
/*                 device.  Use it as a guideline.  You will want to         */
/*                 change the memory layout to match your specific C6xxx     */
/*                 target system.  You may want to change the allocation     */
/*                 scheme according to the size of your program.             */
/*                                                                           */
/*****************************************************************************/

MEMORY
{
    DSPL2ROM     o = 0x00700000  l = 0x00100000   /* 1MB L2 Internal ROM */
    DSPL2RAM     o = 0x00800000  l = 0x00040000   /* 256kB L2 Internal RAM */
    DSPL1PRAM    o = 0x00E00000  l = 0x00008000   /* 32kB L1 Internal Program RAM */
    DSPL1DRAM    o = 0x00F00000  l = 0x00008000   /* 32kB L1 Internal Data RAM */
    SHDSPL2ROM   o = 0x11700000  l = 0x00100000   /* 1MB L2 Shared Internal ROM */
    SHDSPL2RAM   o = 0x11800000  l = 0x00040000   /* 256kB L2 Shared Internal RAM */
    SHDSPL1PRAM  o = 0x11E00000  l = 0x00008000   /* 32kB L1 Shared Internal Program RAM */
    SHDSPL1DRAM  o = 0x11F00000  l = 0x00008000   /* 32kB L1 Shared Internal Data RAM */
    EMIFACS0     o = 0x40000000  l = 0x20000000   /* 512MB SDRAM Data (CS0) */
    EMIFACS2     o = 0x60000000  l = 0x02000000   /* 32MB Async Data (CS2) */
    EMIFACS3     o = 0x62000000  l = 0x02000000   /* 32MB Async Data (CS3) */
    EMIFACS4     o = 0x64000000  l = 0x02000000   /* 32MB Async Data (CS4) */
    EMIFACS5     o = 0x66000000  l = 0x02000000   /* 32MB Async Data (CS5) */
    SHRAM        o = 0x00840000  l = 0x00400000   /* 128kB Shared RAM */
    DDR2         o = 0xCFFFFFFF  l = 0xC0000000   /* 512MB DDR2 Data */


}

SECTIONS
{
    .text          >  DDR2
```

```
SECTIONS
{
    .text           >  DDR2
    .stack          >  DDR2
    .bss            >  DDR2
    .cio            >  DDR2
    .const          >  DDR2
    .data           >  DDR2
    .switch         >  DDR2
    .sysmem         >  DDR2
    .far            >  DDR2
    .args           >  DDR2
    .ppinfo         >  SHRAM
    .ppdata         >  DDR2

    /* COFF sections */
    .pinit          >  DDR2
    .cinit          >  DDR2

    /* EABI sections */
    .binit          >  SHRAM
    .init_array     >  SHRAM
    .neardata       >  SHRAM
    .fardata        >  DDR2
    .rodata         >  SHRAM
    .c6xabi.exidx   >  SHRAM
    .c6xabi.extab   >  SHRAM
}

```