# FILTROS ADAPTATIVOS

Cajamarca Villa Julio Eduardo, Idrovo Coronel Diego Rolando, Valverde Jara Juan Andrés

Universidad Politécnica Salesiana

Enero 8, 2010

Cuenca Ecuador

**Abstract**. Aunque las redes adaline presentan ciertos inconvenientes, los filtros adaptativos que son su aplicación más importante son de gran utilidad debido a que permite suprimir el ruido de sistemas que no son conocidos para el diseñador, pero a su vez se conoce el ruido o la fuente que lo provoca.

***Palabras clave:*** *Adaline, Filtro adaptativo, ALC, LMS, Matlab*

## 1 Introducción

En este trabajo presentamos una breve descripción de las redes ADALINE, como funcionan, algoritmos de aprendizaje, y sus funciones principales. En especial se aborda el tema de filtros adaptativos, la aplicación más importante de este tipo de neurona, su utilidad, comportamiento, su estructura y funcionamiento.

Como aplicación se realiza un programa en Matlab que graba una señal de audio, a esta se le suma un ruido, y por medio de un filtro adaptativo, se recupera la señal grabada libre de ruido.

## 2 Desarrollo Teórico

### 2.1 La red ADALINE

Una red ADALINE significa ADAptive LINear Element o ADAptive LINear NEuron, trabaja con patrones reales tanto a su entrada como a su salida.

Su salida se calcula por una única unidad de procesamiento, y es función lineal de las entradas. Si se desea una salida binaria se puede adjuntar una función de activación con un umbral.

La salida de la red es la suma ponderada de las N entradas, respecto a los pesos asignados a cada una de ellas, más un bias :

 Ec. 1



Figura 1. (a) Con salida lineal ALC (Adaptive Linear Combiner) (b) con salida binaria (umbral)

### 2.2 Aprendizaje de la red Adaline

La neurona aprende de forma supervisada gracias a la regla Delta de Widrow-Hoff basada en la regla del mínimo cuadrado medio LMS (LEAST MEAN SQUARED), con la que se encuentra un vector de pesos W que es único. Este vector asocia el conjunto de patrones de entrada con su correspondiente patrón de valores esperados en su salida. Para cada una de estas combinaciones, los pesos se ajustan automáticamente hasta obtener las salidas correctas (sin error), o por lo menos, con un error mínimo.

### 2.3 Filtro Adaptativo

Un filtro adaptativo es un sistema que pretende modelar la relación existente entre señales en tiempo real de forma repetida. La diferencia con los filtros digitales radica, en que éstos últimos poseen coeficientes invariantes en el tiempo, mientras que un filtro adaptativo puede cambiar su comportamiento en base a un algoritmo adaptativo.

Algo que no debemos olvidar es que los coeficientes del filtro w(n) son desconocidos, estos coeficientes son calculados a medida de que el filtro se implementa, reajustándose automáticamente en cada iteración mientras aprende.

**2.4 Estructura de un filtro adaptativo**

Un filtro adaptativo es un sistema al que llegan dos señales x(n) y e(n), donde e(n) se la conoce como la señal de error y está dada de la resta entre la señal deseada, d (n), y otra que es la salida del filtro y(n).

Ec. 2

A los coeficientes del filtro w(n) se los multiplican por los valores presentes en la entrada x(n) para obtener la salida por lo tanto tenemos:

  y(n)=w(n)\, x(n) Ec. 3

El principal objetivo es hacer que el error entre x(n) y y(n) sea cero, en la práctica no se logra pero se aproxima mucho, para ello el sistema debe configurarse para que a partir de la señal de entrada x(n) se genere la salida y(n) de forma que sea igual a la señal deseada d(n).

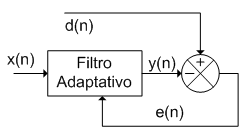
[](http://es.wikipedia.org/wiki/Imagen:Filtro_adapt.PNG)

Figura 2. Estructura de un filtro adaptativo

**2.5 ALGORITMO**

Un algoritmo de aprendizaje del filtro adaptativo tiene la siguiente forma:

1. los pesos w(n) hay que inicializarlos de forma aleatoria.
2. Elegir un valor α para ajustar la velocidad convergencia.
3. Calcular la salida y(n).
4. Calcular el error e(n).
5. Actualizar los pesos con la función J = e2(n).
6. Repetir un determinado número de veces desde el punto 3.

**2.6 APLICACIONES**

Las aplicaciones de los filtros adaptativos son:

* Sistema de identificación.
* Cancelación de eco.
* Eliminación de [ruido blanco](http://es.wikipedia.org/wiki/Ruido_blanco).
* Predictores.

**2.7 Implementación de un filtro adaptativo en Matlab**

El filtro consiste en sumar a una señal de audio cualquiera, un ruido conocido, que puede ser del tipo senoidal, cosenoidal, etc., para luego entrenar una red de tipo adaline, que nos servirá para crear el filtro de tipo adaptativo, este tendrá como parámetros de entrada el ruido y el valor deseado la suma de la señal de audio mas el ruido.

## 2.7.1 Creación

El comando newlin devuelve una red de tipo adaline.

NET = newlin(PR,S,ID,LR)

Donde cada uno de los parámetros son:

* **PR** Es una matriz con los valores máximos y mínimos que puede tomar cada una de las entradas
* **S** es el número de elementos del vector de salida
* **ID** es retardo de la entrada, por defecto su valor es [0]
* **LR** es la velocidad de aprendizaje, por defecto 0.01

**La red anteriormente creada tiene una serie de parámetros a configurar como son:**

* **NET.biasConnect** Define que capas tienen bias.
* **NET.trainParam.epochs** Número máximo de ciclos de entrenamiento.
* **NET.trainParam.goal** Error máximo que el diseñador espera.
* **NET.IW** Matrices de pesos de las capas de la red.
* **NET.b** Define los vectores del bias.
* **NET.layers** Define las propiedades de cada capa de la red. Con NET.layers{i} se tiene acceso a las propiedades de la capa i.

## 2.7.2 Entrenamiento

Una vez creada la red, tenemos que entrenarla esto se logra con el comando adapt, este entrenamiento es del tipo dinámico, es decir se recalculan los pesos por cada patrón de entrenamiento. La sintaxis del comando es la siguiente:

[net,Y,E] = adapt(NET,P,T)

Donde cada uno de los parámetros de entrada son:

* NET Una red inicializada.
* P Los patrones de entrada.
* T Las salidas deseadas.

Y para los parámetros de salida tenemos

* net Red entrenada.
* Y Es la salida de la red.

E Errores de la red

## 2.7.3 Código del M-File

function varargout = practica(varargin)

gui\_Singleton = 1;

gui\_State = struct('gui\_Name', mfilename, ...

'gui\_Singleton', gui\_Singleton, ...

'gui\_OpeningFcn', @practica\_OpeningFcn, ...

'gui\_OutputFcn', @practica\_OutputFcn, ...

'gui\_LayoutFcn', [] , ...

'gui\_Callback', []);

if nargin && ischar(varargin{1})

gui\_State.gui\_Callback = str2func(varargin{1});

end

if nargout

[varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:});

else

gui\_mainfcn(gui\_State, varargin{:});

end

% --- Executes just before practica is made visible.

function practica\_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

% Update handles structure

guidata(hObject, handles);

initialize\_gui(hObject, handles, false);

% --- Outputs from this function are returned to the command line.

function varargout = practica\_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

% --- Executes on button press in filtro.

function filtro\_Callback(hObject, eventdata, handles)

global t Y fs r se;

e=r-Y; %restamos la (señal con ruido) menos (entrenamiento)

axes(handles.graf\_recuperada);

cla;

if(1==get(handles.espectro,'value'));

sfe=fft(e,512);

ef=((0:255)/256\*(fs/2));

sfeesp=abs(sfe);

plot(ef,sfeesp(1:256));

else

plot(e);

end

grid on;

dif=e-se;

axes(handles.diferencia\_senal);

cla;

if(1==get(handles.espectro,'value'));

sfdif=fft(dif,512);

diff=((0:255)/256\*(fs/2));

sfdifesp=abs(sfdif);

plot(diff,sfdifesp(1:256));

else

plot(dif);

end

grid on;

soundsc(e,fs);

% --- Executes on button press in selec\_audio.

function selec\_audio\_Callback(hObject, eventdata, handles)

set(handles.selec\_audio,'value',1)

set(handles.grabar,'value',0)

% --- Executes on button press in grabar.

function grabar\_Callback(hObject, eventdata, handles)

set(handles.selec\_audio,'value',0)

set(handles.grabar,'value',1)

% --- Executes on button press in aceptar\_senal.

function aceptar\_senal\_Callback(hObject, eventdata, handles)

global tam1 se fs;

if( 0 ==get(handles.selec\_audio,'value'));

duracion = 4; %TIEMPO DE ADQUISION DE DATOS

Fs = 44100; %constante segun el uso del comando 8000, 11025, 22050, and 44100 Hz

grab = wavrecord(duracion\*Fs,Fs,1);

[filename,pathname] = uiputfile('grabado.wav','Save file name');

wavwrite(grab,Fs,8,'grabado');

else

[filename, pathname] = uigetfile('\*.wav','select a wave file to load');

end

[se,fs,NBITS1]= wavread([pathname filename]);

se=se';

tam1=size(se)-1;

axes(handles.graf\_senal);

cla;

if(1==get(handles.espectro,'value'));

sfse=fft(se,512);

sef=((0:255)/256\*(fs/2));

sfseesp=abs(sfse);

plot(sef,sfseesp(1:256));

else

plot(se);

end

grid on;

soundsc(se,fs);

% --- Executes on selection change in tipo\_ruido.

function tipo\_ruido\_Callback(hObject, eventdata, handles)

global w tam1 fs t Y;

freq=3000;

t=(0:tam1(1,2))/fs;

if( 1 == get(handles.tipo\_ruido,'value'))

w=sin(freq\*2\*pi\*t);

load '0.001senoidal\_Y.mat' Y;

end

if( 2 == get(handles.tipo\_ruido,'value'))

w=cos(freq\*2\*pi\*t);

end

if( 3 == get(handles.tipo\_ruido,'value'))

w=randn(1,tam1(1,2)+1);

end

function tipo\_ruido\_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on button press in aceptar\_ruido.

function aceptar\_ruido\_Callback(hObject, eventdata, handles)

global w fs t tam1;

t=(0:tam1(1,2))/fs;

axes(handles.graf\_ruido);

cla;

if(1==get(handles.espectro,'value'));

sfw=fft(w,512);

wf=((0:255)/256\*(fs/2));

sfwesp=abs(sfw);

plot(wf,sfwesp(1:256));

else

plot(w);

end

grid on;

soundsc(w,fs);

% --- Executes on button press in sumar\_senal.

function sumar\_senal\_Callback(hObject, eventdata, handles)

global se w r fs;

r=se+w; %sumamos la señal y ruido

r=r/max(r); % normalizamos la señal sumada

axes(handles.graf\_suma);

cla;

if(1==get(handles.espectro,'value'));

sfr=fft(r,512);

rf=((0:255)/256\*(fs/2));

sfresp=abs(sfr);

plot(rf,sfresp(1:256));

else

plot(r);

end

grid on;

soundsc(r,fs);

% --- Executes on button press in entrenar.

function entrenar\_Callback(hObject, eventdata, handles)

% hObject handle to entrenar (see GCBO)

global r w Y;

f=msgbox('Entrenando sistema.','Busy');

P=con2seq(w); % ruido %Hacemos en filas, no en columnas

T=con2seq(r); %senal mas ruido %r= senal mas ruido, w= ruido

net=newlin([-1 1],1,[0 2],0.08); %Definimos valores max, min y 1 neurona, Error mínimo: 0.08 en base a varia pruebas

net.adaptParam.epoch=20;

net.IW{1,1}=rands(1,2);

net.b{1}=[0];

[net,Y,E,Pf]=adapt(net,P,T); %entrenamos a la red como entrada el ruido y como valor esperado se;al mas ruido

Y=cell2mat(Y);

delete(f)

function initialize\_gui(fig\_handle, handles, isreset)

global w;

global fs;

global tam1;

global se;

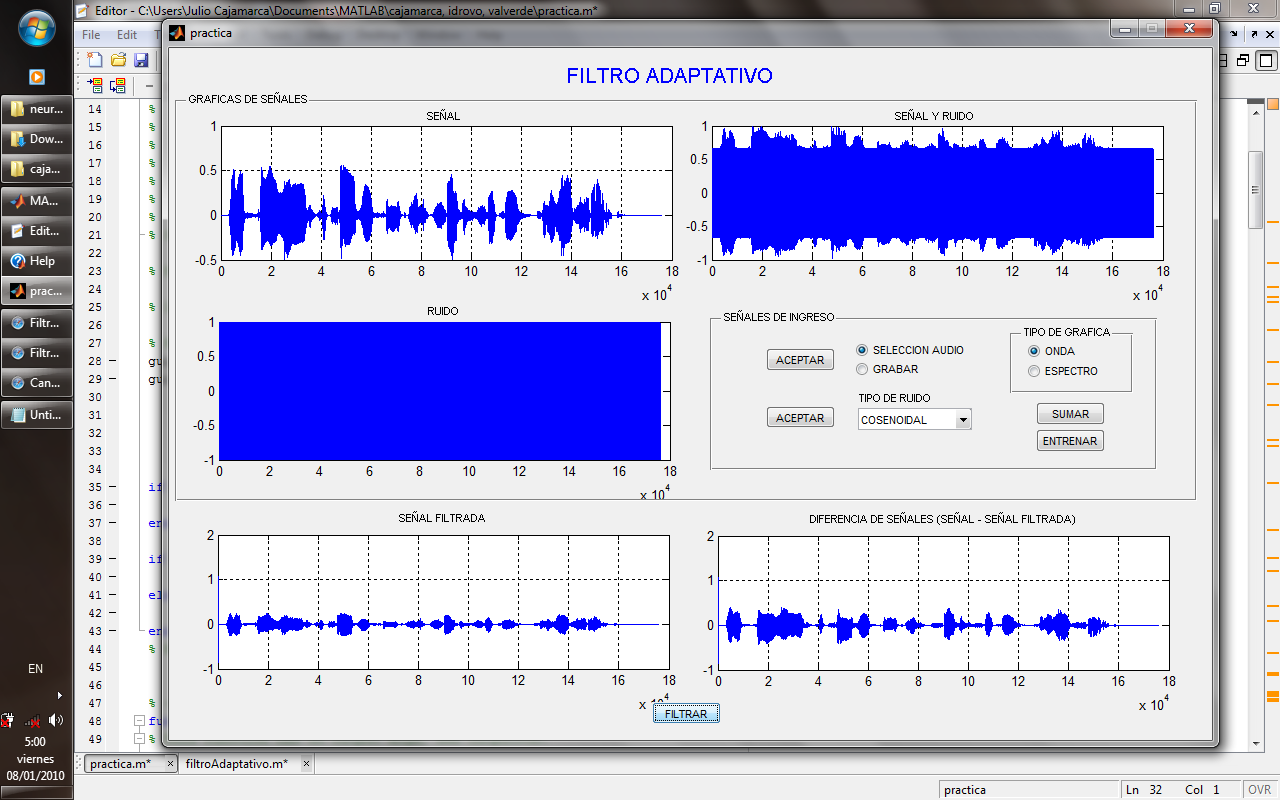
set(handles.selec\_audio,'value',1)

set(handles.grabar,'value',0)

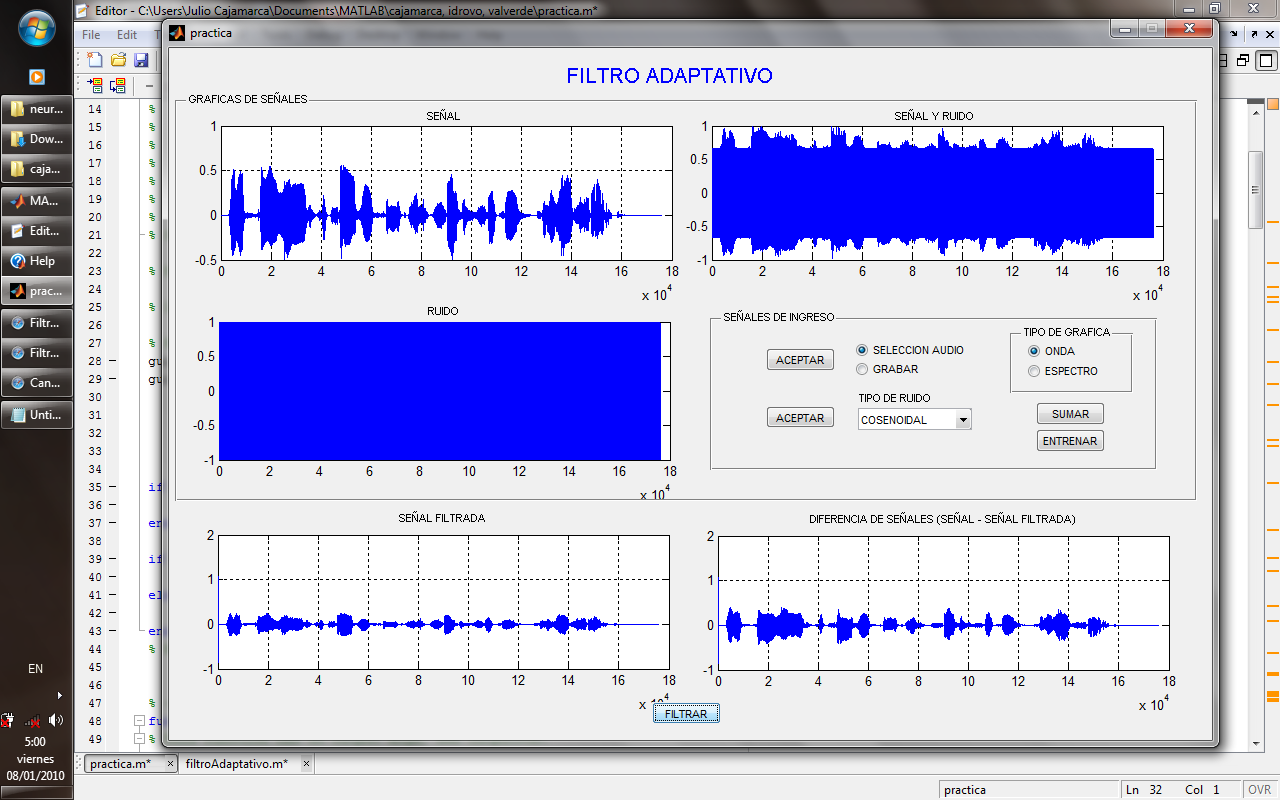
set(handles.onda,'value',1)

set(handles.espectro,'value',0)

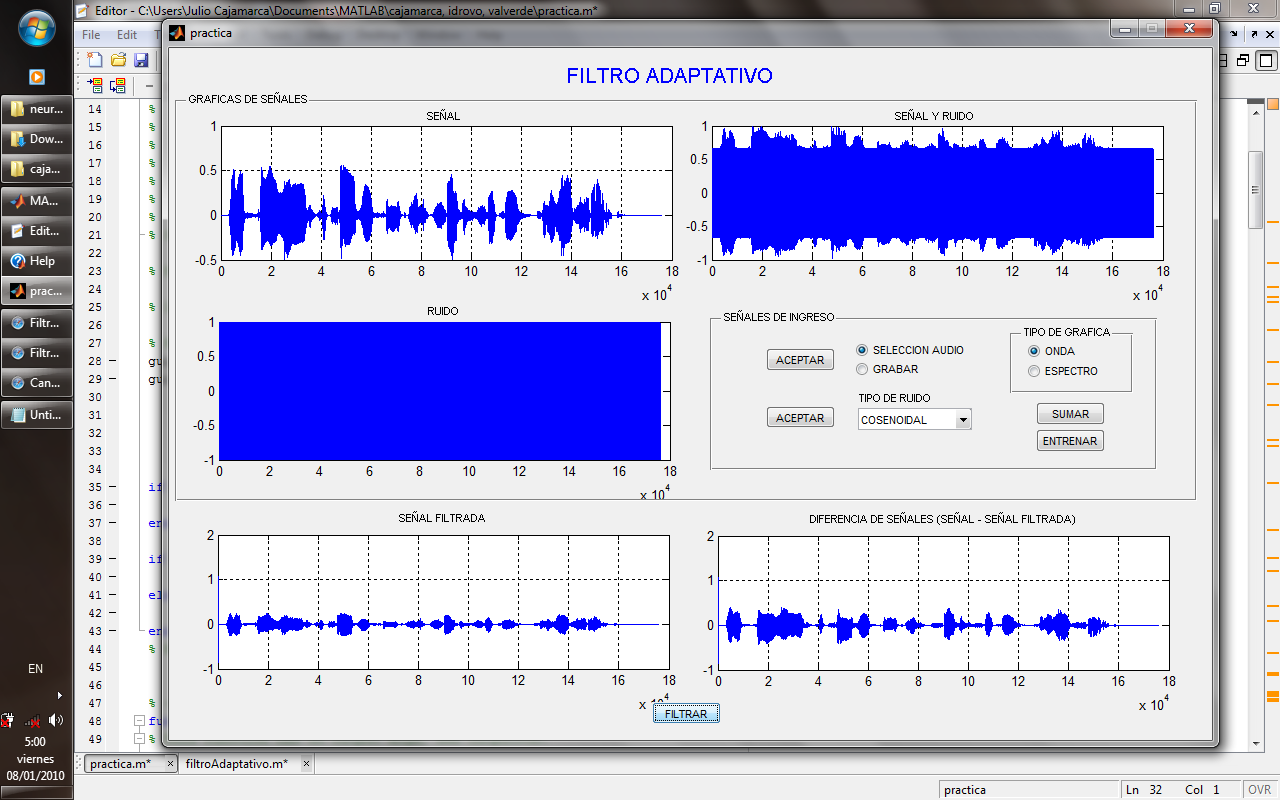
## 4. Resultados

****

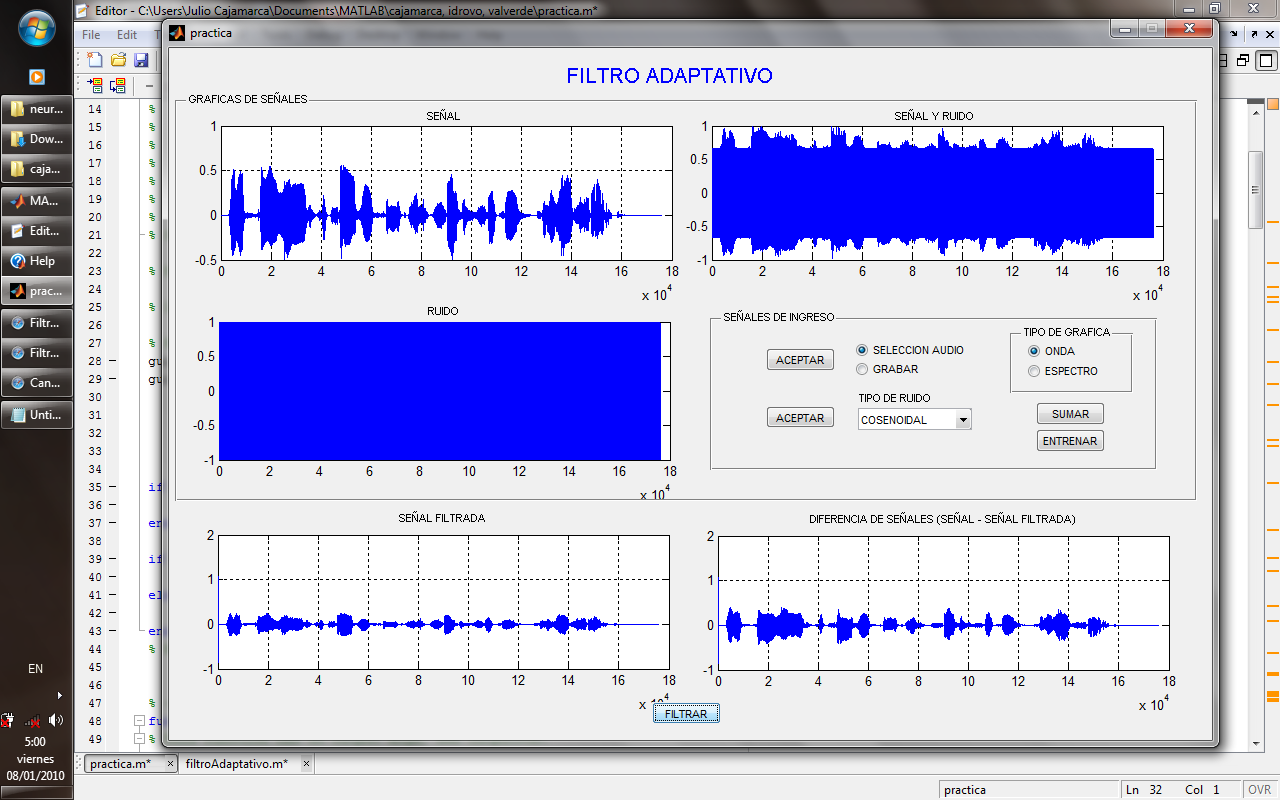
**Figura 3.** Señal de audio

****

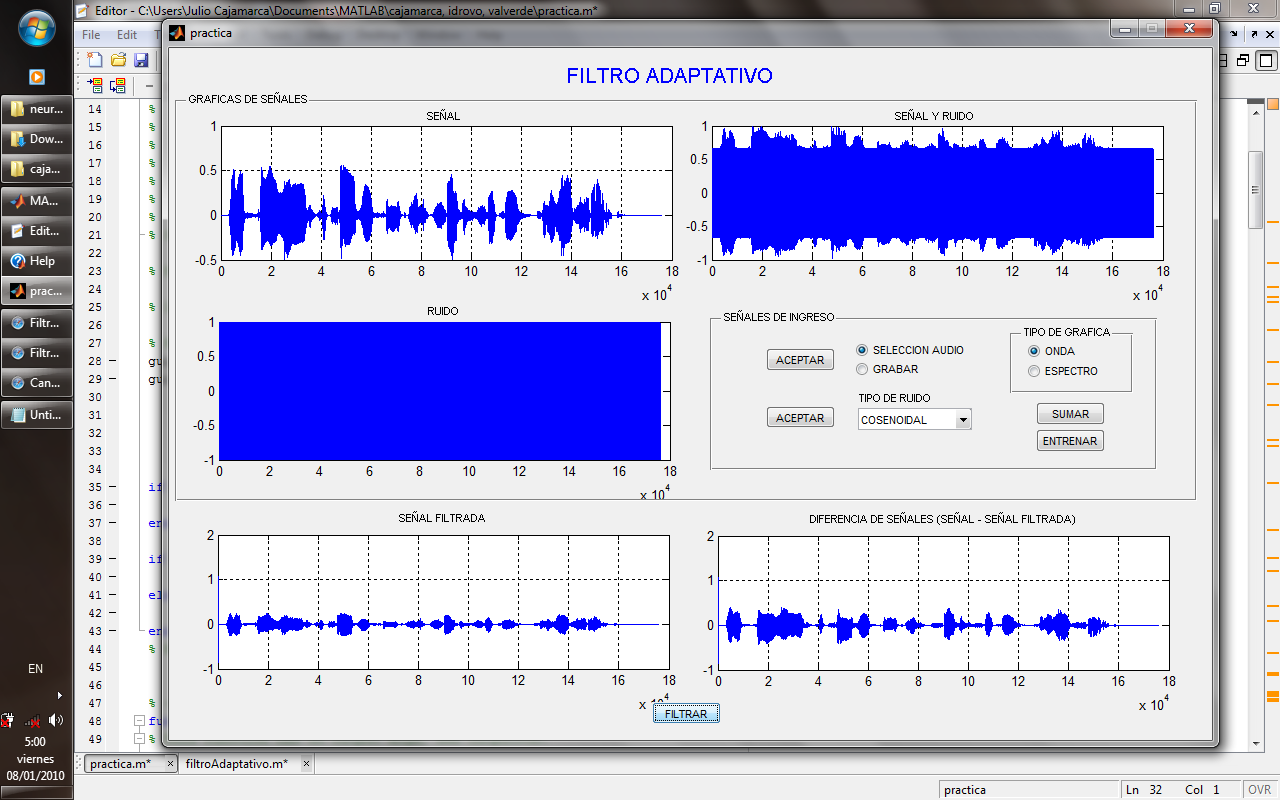
**Figura 4.** Señal de ruido



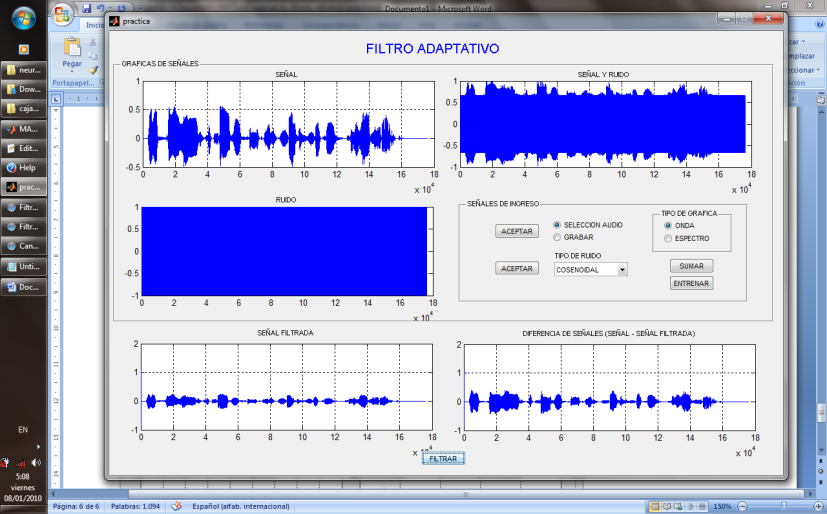
**Figura 5.** Señal de audio mas ruido

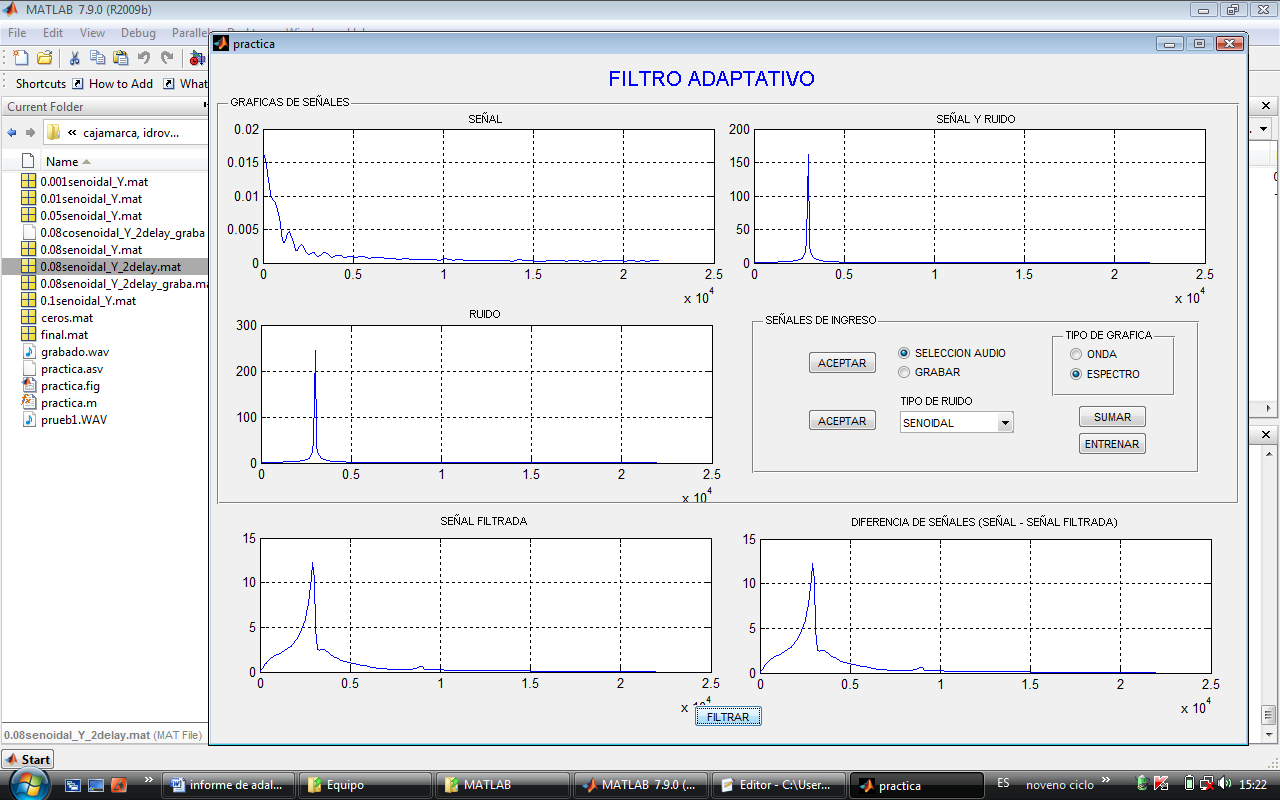


**Figura 6.** Señal filtrada



**Figura 7.** Diferencia de señales (señal-señal filtrada)

**Figura 8.** Analisis en el dominio del tiempo



**Figura 9.** Análisis espectral

## 5. Conclusiones

La red adaline presenta una serie de limitaciones, es por ello que se dejo de estudiarla por un tiempo, sin embargo se ha retomado su estudio debido a la utilidad principal de filtro, que nos permite suprimir el ruido de sistemas que no son conocidos para el diseñador, pero a su vez se conoce el ruido o la fuente que lo provoca.

Cuando se realiza una red adaline se puede configurar varios de sus parámetros, entre estos tenemos los delays (retardos) en su entrada. En base a varias pruebas realizadas se observo que la calidad del filtrado mejora hasta un cierto punto, de aquí en adelante no existe una diferencia significativa y además aumenta el tiempo de aprendizaje de la red por lo que para aplicaciones en tiempo real esto puede ser perjudicial pues necesitaríamos más tiempo de procesador y recursos para el procesamiento de la señal. Otro de los parámetros que se modifica es la taza de aprendizaje, esta indica que tan rápido y que tan preciso es el aprendizaje, al experimentar se observo que al establecer valores desde 0.001 hasta 0.1, se observo que mientras más pequeño sea el valor la respuesta final es más precisa, pero tiene una atenuación más larga del error antes de eliminarlo, mientras que un número más grande genera una atenuación más corta en la eliminación del error pero la señal recuperada es menos precisa y mas atenuada que la original.

La señal de ingreso está entre ±1 V, que es lo que habitualmente tiene una señal de audio, al sumarle ruido a la señal los valores picos aumentan y esto significa un mayor rango en el que varia la señal lo que significa menos velocidad de aprendizaje, es por ello que se procedió a normalizar la señal más el ruido antes de entrenar el sistema.

El sistema realizado no es tiempo real por lo que no se le puede considerar aplicable para cualquier tipo de señal de ingreso, pues los valores que se obtienen al entrenar la red son solo para esa señal y la modificación de la misma requerirá un nuevo entrenamiento.

## Referencias

[1] Lakhmi C. Jain; N.M. Martin (1998). Fusion of Neural Networks, Fuzzy Systems and Genetic Algorithms: Industrial Applications. CRC Press LLC.

[2] Krose Ben, van der Smagt Patrick (1996). An

Introduction to Neural Networks. The University of Amsterdam.

[3] Ramírez Ignacio, Alcarraz Andrés (2001). Clasiliptus. Obtenido de: http://www.fing.edu.uy/iie/~nacho/clasiliptus/

[4] Demuth Howard, Beale Mark (2001). Neural Network Toolbox. The MathWorks, Inc.