TRABAJO PRÁCTICO FINAL:

Filtro Digital FIR en Scilab

12/07/2019

- **CARRERA:** **Ingeniería en informática.**
- **ASIGNATURA:** **Teoría y elaboración de señales.**
- **DOCENTES:**
  - **VERNENGO, Martín;**
  - **PARADISO, Juan Carlos.**
- **INTEGRANTES:**
  - **CALONGE, Federico;**
  - **CANDELIERI, Santiago.**

Contenido

1. Objetivo

El **objetivo** de este Trabajo de Laboratorio es aplicar todos los conceptos matemáticos vistos en la materia a un ejemplo práctico. De esta manera, utilizaremos **Scilab** para crear nuestros propios ***filtros FIR*** con el cual filtraremos tonos de frecuencias indeseadas en audios, para dejar sólo la voz de la persona.

2. Introducción: Filtro Digital vs Filtro Analógico.

Un **filtro analógico** posee tanto en la entrada como en la salida una **señal analógica**. Ambas, la entrada (x(t)) y la salida (y (t)), son funciones de una **variable continua** t, y puede tener un **rango infinito de valores**. Un filtro analógico emplea circuitos electrónicos con componentes discretos tales como resistencias, capacitores, amplificadores operacionales, etc.; los cuales se requieren para el filtrado deseado. El diseño de filtros analógicos es 50 años más antiguo que el diseño de filtros digitales. Por lo tanto, la mayoría de los filtros analógicos son diseños probados que existen y pueden encontrarse en muchos lados.

En cambio, un **filtro digital** opera con **señales digitales**; empleando un procesador digital que efectúa operaciones matemáticas en valores **muestreados** de la señal. De esta manera, una señal digital de entrada ((secuencia de "n" muestras: x(n)) es **transformada** en una segunda secuencia de muestras o señal digital de salida: y (n). Ambas, la entrada, y la salida, son funciones de **variable discreta n** y tienen un **rango finito de valores**.

Igualmente, la entrada y salida de estos filtros puede ser analógica; para esto la entrada analógica debe ser muestreada y digitalizada previamente utilizando un **ADC** (conversor analógico-digital). El resultado son números binarios que representan los valores sucesivos muestreados. Estos son transferidos al procesador, el cual efectúa las **operaciones matemáticas** en ellos. Un ejemplo de estas operaciones matemáticas puede ser realizar convoluciones en el dominio temporal con otras señales prefijadas; usándose para el diseño de estos filtros un impulso y un desplazamiento sucesivo de veces realizando una multiplicación por alguna constante, es decir, utilizando la Transformada Z. Finalmente, si es necesario, los resultados de estos cálculos (que representan valores muestreados de la señal filtrada) son enviados a través de un **DAC** (conversor digital-analógico) para devolver la señal a una forma analógica.

El muestreo moderno y los instrumentos actuales para procesar señales digitales hicieron posible el reemplazo de los **filtros analógicos** por los **filtros digitales** en aplicaciones que requieren flexibilidad y programación. Estas aplicaciones incluyen tanto audio, telecomunicaciones como geofísica y monitorización médica. Las ventajas de los filtros digitales frente a los analógicos son las siguientes:

- Son software programable (su funcionamiento está terminado por un programa almacenado en la memoria contigua al procesador); y por ello son muy fáciles de montar y testear. Por esto los filtros digitales pueden ser fácilmente diseñados, probados e implementados en un ordenador. En cambio, los analógicos pueden ser simulados, pero siempre hay que implementarlos a través de componentes discretos para ver su funcionamiento real.
- Los filtros digitales son mucho más versátiles a la hora de manipular la señal.
- A diferencia de los filtros analógicos, los digitales pueden manejar con mucha precisión las bajas frecuencias.
- Solo requieren de operaciones aritméticas como la multiplicación y la suma/resta y así son más fáciles de implementar.
- No sufren por variaciones de fabricación o por antigüedad.
- Son estables (no cambian con el tiempo o la temperatura) y previsibles; no requieren componentes de precisión.

En nuestro caso, utilizaremos **Filtros Digitales FIR** (Ver ***Punto 4***), los cuales realizaremos en Scilab para filtrar frecuencias indeseadas en audios; y así solo dejar la voz de la persona.

3. Tipos de filtro según su frecuencia de corte

- **1-Filtro Pasa Bajo:** Aquel que permite el paso de frecuencias bajas, desde frecuencia 0 (o frecuencia de continua) hasta una fc (frecuencia de corte) determinada.
- **2-Filtro Pasa Alto:** Aquel que permite el paso de frecuencias desde una fc determinada hacia arriba, sin que exista un límite superior especificado.
- **3-Filtro Pasa Bandas:** Aquel que **permite** el paso de componentes frecuenciales contenidos en un determinado rango de frecuencias, comprendido entre una frecuencia de corte superior f1 y otra inferior f2.
- **4-Filtro Elimina Bandas – "NOTCH":** Aquel que **dificulta** el paso de componentes frecuenciales contenidos en un determinado rango de frecuencias, comprendido entre una frecuencia de corte superior W1 y otra inferior W2.

*Estos 4 tipos de Filtros están especificados en la ***Figura 1***.

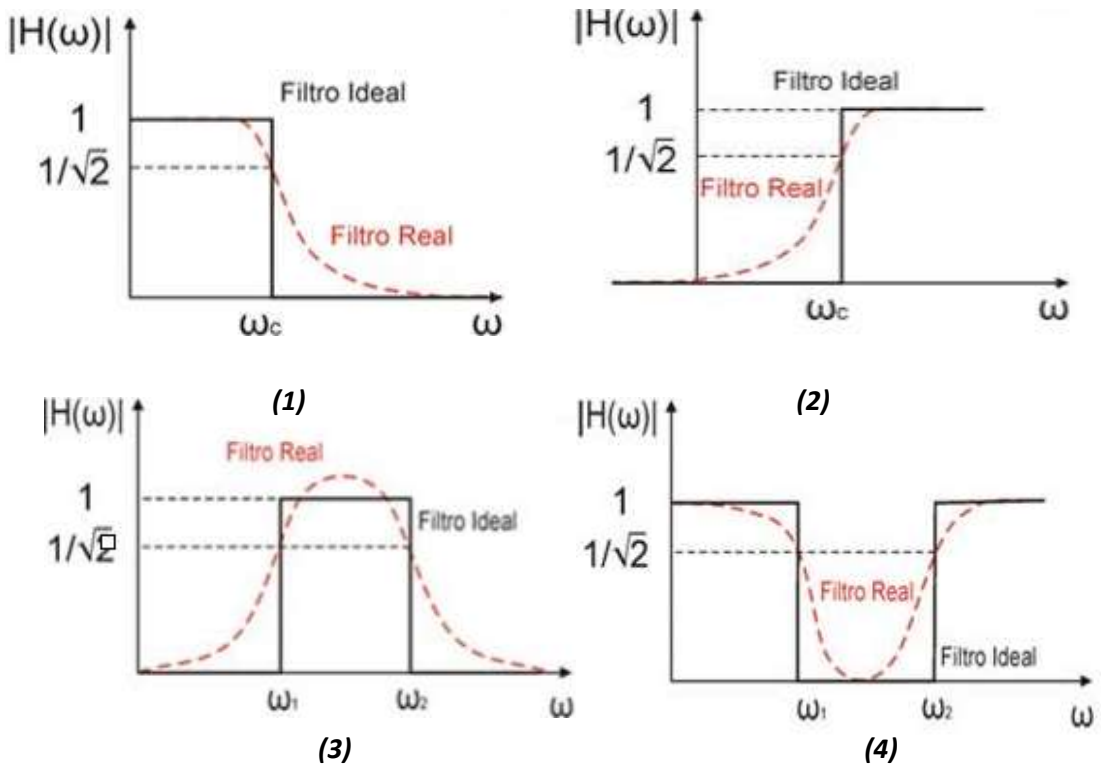


Figura 1. Tipos de filtro según su Wc.
1) Pasa Bajos. 2) Pasa Altos. 3) Pasa Bandas 4) Rechaza Bandas – NOTCH

4. Tipos de filtro según su respuesta ante entrada unitaria.

4.1-Filtro digital IIR - Infinite Impulse Response (Respuesta al Impulso Infinita)

Tipo de filtro digital que si su entrada es un impulso, la salida será un número **ilimitado** de términos no nulos, es decir, que **nunca volverá a un estado de reposo**. Para obtener la salida se emplean valores de la entrada actual y anteriores y, además, valores de salida anteriores que son almacenados en memoria y realimentados a la entrada. Por esto es que también se llaman filtros digitales **recursivos**.

Su expresión en el dominio discreto es:

$$y[n] = \sum_{i=0}^{P} b_i x[n-i] - \sum_{j=1}^{Q} a_j y[n-j]$$

Y su función de transferencia (relación entre la entrada y la salida) para este filtro es:

$$H(z) = \frac{\sum_{i=0}^{P} b_i z^{-i}}{\sum_{j=0}^{Q} a_j z^{-j}}$$

Las ventajas de los filtros IIR respecto a los filtros FIR es que pueden conseguir una misma respuesta empleando un número de coeficientes en el filtro mucho menor, requiriendo un menor tiempo de cálculo. La desventaja, es que su **estabilidad** y **causalidad** pueden ser afectados (esto es debido a la presencia de ceros además de polos en el denominador de H(z)). Otra desventaja es la **introducción de desfasajes** en la señal, que pueden ser compensados pero a costa de añadir más coeficientes al filtro.

4.2-Filtro digital FIR - Finite Impulse Response (Respuesta al Impulso finita)

Es un tipo de filtro digital que si su entrada es un impulso, la salida será un número **limitado** de términos no nulos. Para obtener la salida sólo se emplean valores de la entrada actual y anteriores. Son filtros digitales **NO recursivos**.

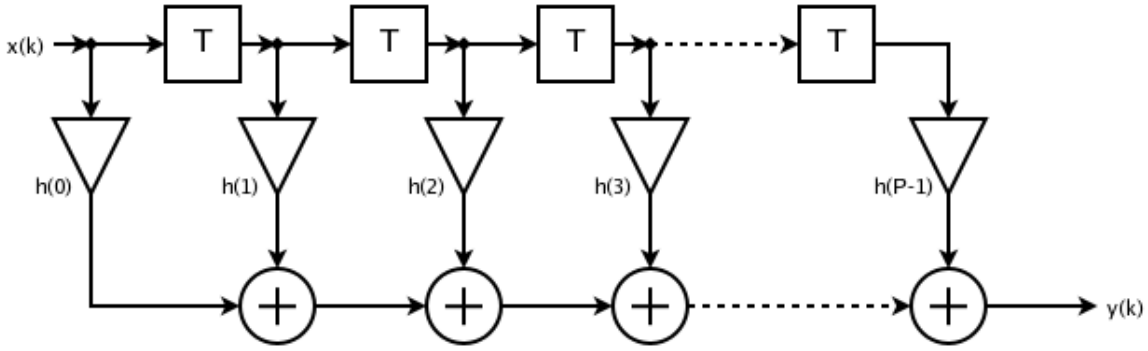
Su expresión en el dominio discreto es:

$$y_n = \sum_{k=0}^{N-1} b_k x(n-k)$$

El orden del filtro está dado por **N** (el número de coeficientes). Además, la salida puede ser expresada como la convolución discreta de una señal de entrada x[n] con un filtro h[n]:

$$y_n = \sum_{k=0}^{N-1} h_k x_{n-k}$$

La estructura de un filtro FIR por tanto es la siguiente:



La cual puede verse reflejada en la aplicación de la **Transformada Z** para obtener su función de transferencia:

$$H(z) = \sum_{k=0}^{N-1} h_k z^{-k} = h_0 + h_1 z^{-1} + \cdots + h_{N-1} z^{-(N-1)}$$

Se puede ver que es la misma entrada retardada cada vez más en el tiempo, multiplicada por diversos coeficientes y finalmente sumada al final. Dado que los filtros FIR sólo tienen polos (elementos en el numerador en su función de transferencia), son **estables**. Otra ventaja de los filtros FIR es que pueden diseñarse para ser de fase lineal, es decir, **no introducen desfases en la señal**, a diferencia de los IIR o los filtros analógicos. Por ese motivo tienen interés en **audio**. Sin embargo, tienen el inconveniente de ser más largos al tener más coeficientes que los filtros IIR capaces de cumplir similares características. Esto requiere un **mayor tiempo de cálculo** que puede dar problemas en aplicaciones en tiempo real.

5. Nuestro filtro en Scilab

Como nombramos anteriormente, el objetivo de este Trabajo de Laboratorio Final es simular un filtro digital FIR para eliminar frecuencias indeseadas en un archivo de audio. En dicho archivo de audio se encuentra una persona hablando y un tono/frecuencia que "molesta"; las cuales debemos atenuar.

Para filtrar los distintos audios necesitaremos un filtro adaptado para cada uno; ya que como los audios tienen tonos en distintas frecuencias (los cuales debemos atenuar), se necesitarán distintas características del filtro. Para simular estos filtros utilizaremos las funciones descritas en la ***Sección 5.1***. La forma en que armamos el filtro está descrita en la ***Sección 5.2***.

5.1-Funciones que utilizaremos en Scilab.

5.1.1-Funciones que realizamos nosotros:

- **sincPi(x):** mediante esta función obtenermos la función $\frac{SINC\ (\pi.x)}{\pi.x}$ (Ver ***Figura 2***).

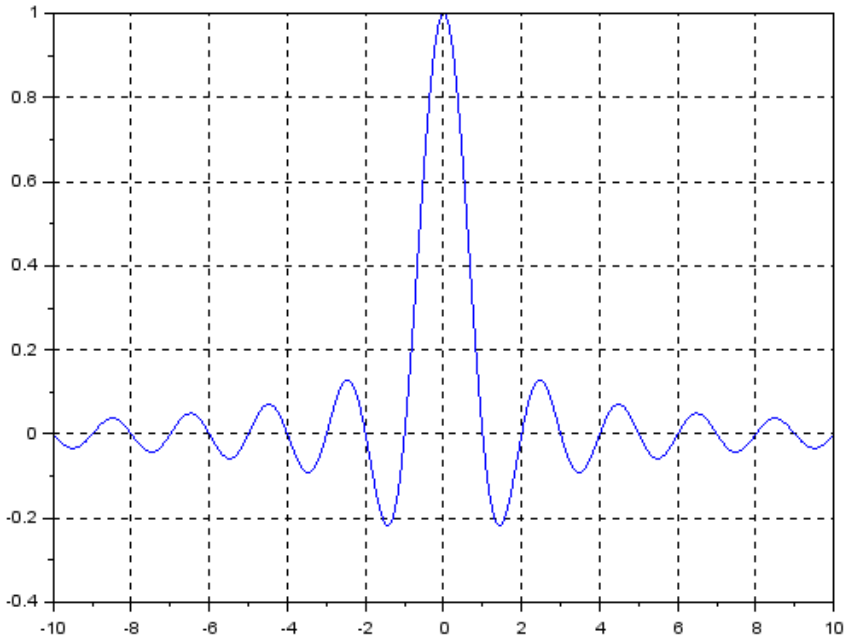


Figura 2

- **Polyval(p,x):** Devuelve el valor de un polinomio de grado n evaluado en **x**. **p** es un vector de entrada cuyos elementos son los coeficientes en las potencias descendentes del polinomio a evaluar.

- **PolyvalNegativo(p,x):** misma procedimiento que Polyval(p,x), solo que la potencia a la cual elevamos a **x** es **negativa**.

- **filterFIR():** misma aplicación que "filter()" (ver Sección 5.1.2). Permite filtrar una secuencia de datos utilizando un filtro digital de tipo **FIR** (no **IIR**, ya que para este tipo de filtro el A(z) debe ser distinto de 1; en el caso del FIR el denominador A(z)=1).

  De esta manera, la función de transferencia de nuestro filtro será:

$$H(Z) = \frac{B(z)}{A(z)} = \frac{\mathrm{VN}(1).\ z^{0} + \mathrm{VN}(2).z^{-1} + \ldots + \mathrm{VN}(\mathrm{LongNum}).Z^{-(LongNum-1)}}{VD(1)}$$

***Código en Scilab de filterFIR():**

```
function y=filterFIR(VN, x)
//Recibo un vector REAL (VN:VectorNumerador) y la señal de entrada (x). La salida de myFilter() es la señal x filtrada.

    //Sacamos longitudes:
    LongNum=length(VN);      //Longitud del vector del NUMERADOR.
    LongX=length(x);         //Longitud de nuestra función de entrada x.

    Maximo=LongNum;
    x2=[zeros(Maximo-1,1);x'];      //Usamos x2 como auxiliar. zeros(2,3) me devuelve una matriz de ceros de 2 filas x 3 columnas. x' es la traspuesta de x. El ";" es para concatenar ambos vectores. Esto lo hacemos porque en el for se abajo (al sacar y[n]) se necesita que los dos vectores tengan el mismo tamaño; entonces agrandamos el tamaño de x rellenándolo con 0s.

   //Ahora hacemos un bucle for para obtener nuestra salida y[n]. Este for está basado en nuestra ecuacion H(z)=B(Z)/A(Z) --> ver help filter(). Va de n=Maximo HASTA Maximo+Nx-1.
    for n=Maximo:LongX+Maximo-1
     y(n)=( (VN(LongNum:-1:1))*(x2(n-LongNum+1:n)) );
  //Al hacer: 8:-1:4 (ejemplo) nos devuelve 8,7,6,5,4 (disminuimos el 8 en 1 hasta llegar al 4).
    end

    //Salida...
    y2=y(Maximo:LongX+Maximo-1);  //Nos sirve para eliminar los 0s.
    y=y2';  //La trasponemos y obtenemos nuestra señal x ya filtrada = señal y.

endfunction
```

5.1.2-Funciones propias de Scilab:

- **figure(), title(), plot(),xgrid():** funciones que utilizamos para graficar las señales y agregarles títulos y cuadriculados a las gráficos.

- **Wavread():** función que nos sirve para **leer** archivos con extensión .wav y retornar los datos muestreados en un vector "y". Utilizaremos esta función para leer el archivo .wav con frecuencias indeseadas para luego realizar el filtrado y atenuarlas. En nuestro caso utilizaremos Wavread()de la siguiente forma:

```
[x, fs, bits] = wavread('.\Audio200Hz\nombreArchivoWAV.wav');
```

En **x** obtenemos la señal de audio; **fs** es la frecuencia de muestreo en Hz que nos devuelve y **bits** es la cantidad de bits por muestra utilizada para codificar los datos en el archivo wav.

- **Wavwrite():** función que nos sirve para grabar archivos con extensión .wav. En nuestro caso utilizamos Wavwrite() de la siguiente forma:

```
wavwrite(y, fs, bits, '.\Audio200Hz\TPFiltroFIRAaudio200Hz_out3N300.wav')
```

  En **fs** le especificamos la frecuencia de muestreo en Hz; y (nuestra señal de salida) será la señal que queremos que grabe y el 3er parámetro es el archivo wav que queremos que cree y su ubicación.

- **Filter():** función que filtra una secuencia de datos utilizando un filtro digital. De esta manera, la función de transferencia del filtro será:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_n z^{-n}}{a_0 + a_1 z^{-1} + \ldots + a_n z^{-n}}$$

  *En vez de utilizar esta función creamos una propia llamada "**filterFIR()**" (Ver **Sección 5.1.2**).

- **length(parametro):** función que recibe como **parámetro** una matriz o un vector y cuenta cuántos valores tiene.

5.2-Armando el filtro.

Se desea armar un filtro notch que se encargue de anular la banda específica en la que se encontrará el tono indeseado. Para ello, como primer paso, se comienza con un filtro pasa bajos compuesto por un trapezoide, como resultado de una convolución entre dos funciones rect de ancho D y E:

$$H(f) = \frac{1}{2E} \cdot rect(\frac{f}{2D}) * rect(\frac{f}{2E})$$

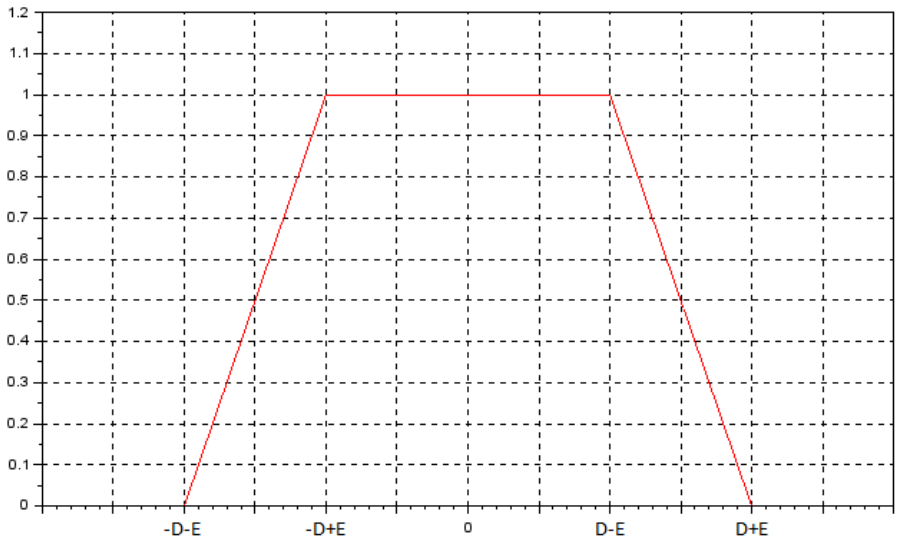


Figura 3 – Gráfico en frecuencia del filtro pasa bajos H(f).

Ahora, para convertir dicho filtro en uno de tipo notch, se debe sumar al mismo otro filtro pasa banda. Partiendo de un filtro pasa bajos, y valiéndose de la propiedad de modulación de la transformada de Fourier, se agregan entonces dos trapezoides desplazados en f0 y –f0 para formar el filtro pasa banda:

$$H(f) = \frac{1}{2E} \cdot rect\left(\frac{f}{2D}\right) * rect\left(\frac{f}{2E}\right) + \frac{1}{2B} \cdot rect\left(\frac{f - f_0}{2A}\right) * rect\left(\frac{f - f_0}{2B}\right) + \frac{1}{2B} \cdot rect\left(\frac{f + f_0}{2A}\right) * rect\left(\frac{f + f_0}{2B}\right)$$

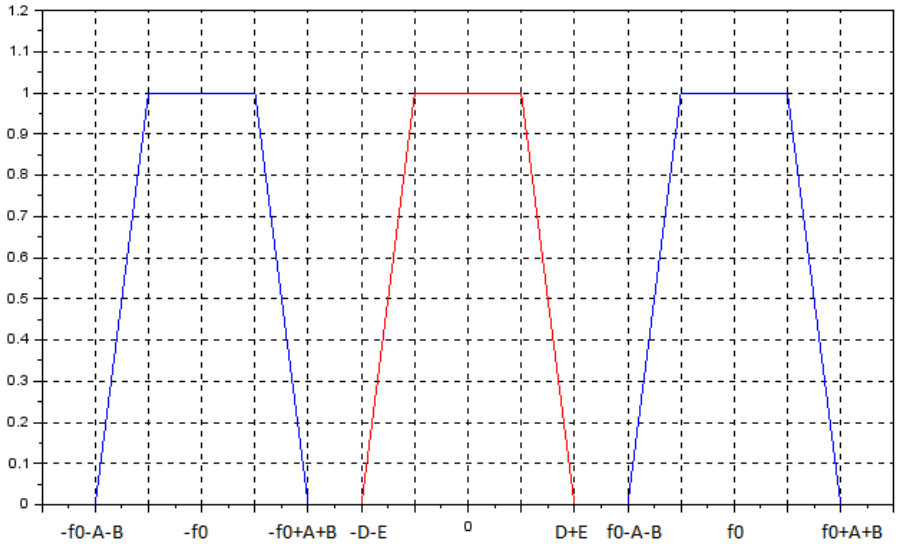


Figura 4 – Gráfico en frecuencia del filtro notch H(f) compuesto de un pasa bajos y un pasa banda.

Entonces, ajustando los valores de A, B, f0, D y E obtenemos el filtro deseado. Aplicando la anti transformada obtenemos nuestro filtro en tiempo:

$$h(t) = \ 2D \cdot sinc(2Dt) \cdot sinc(2Et) + \ 4A \cdot sinc(2At) \cdot sinc(2Bt) \cdot \cos(2\pi f_0 t)$$

A simple vista, se puede observar que dicho filtro no es causal, al encontrarse gran parte del mismo del lado del tiempo negativo. Para solucionarlo, se debe realizar un desplazamiento en tiempo, para que gran parte del filtro se encuentre del lado positivo, y así considerar el lado negativo como despreciable. Esto generará una fase lineal del lado de la frecuencia. El nuevo filtro será entonces:

$$h_c(t) = \ h(t - \ t_d)$$

Al muestrearlo con una frecuencia fs, y reemplazando al desplazamiento td por un múltiplo N del período de muestreo, se obtiene entonces:

$$H(z)]_{z= e^{j2\pi f T_s}} = \ rep_{f_s}[H_c(f)]$$

$$H(z)]_{z= e^{j2\pi f T_s}} = \ T_s \cdot \sum_{k=0}^{2N} h((k-N) \cdot T_s) \cdot e^{-j2\pi f k T_s}$$

<u>Entonces el filtro será:</u>

$$H(Z) = \ T_s \cdot \sum_{k=0}^{2N} [2D \cdot sinc(2D(k-N)T_s) \cdot sinc(2E(k-N)T_s) + \ 4A \cdot sinc(2A(k-N)T_s) \cdot sinc(2B(k-N)T_s) \cdot \cos(2\pi f_0 (k-N)T_s)] \cdot Z^{-k}$$

5.3-Aplicación del filtro para los distintos audios

Se eligió un N de 700 como el más apropiado para recrear la respuesta del filtro.

5.3.1 - Para el audio de 200 Hz:

Luego de varios ajustes, se estableció un A = 7,39 KHz; un f0 = 7627,3 Hz; un D = 177,3 Hz y un B y E de 20 Hz como los valores más apropiados para lograr una banda lo más estrecha posible que anule el tono de 200 Hz.

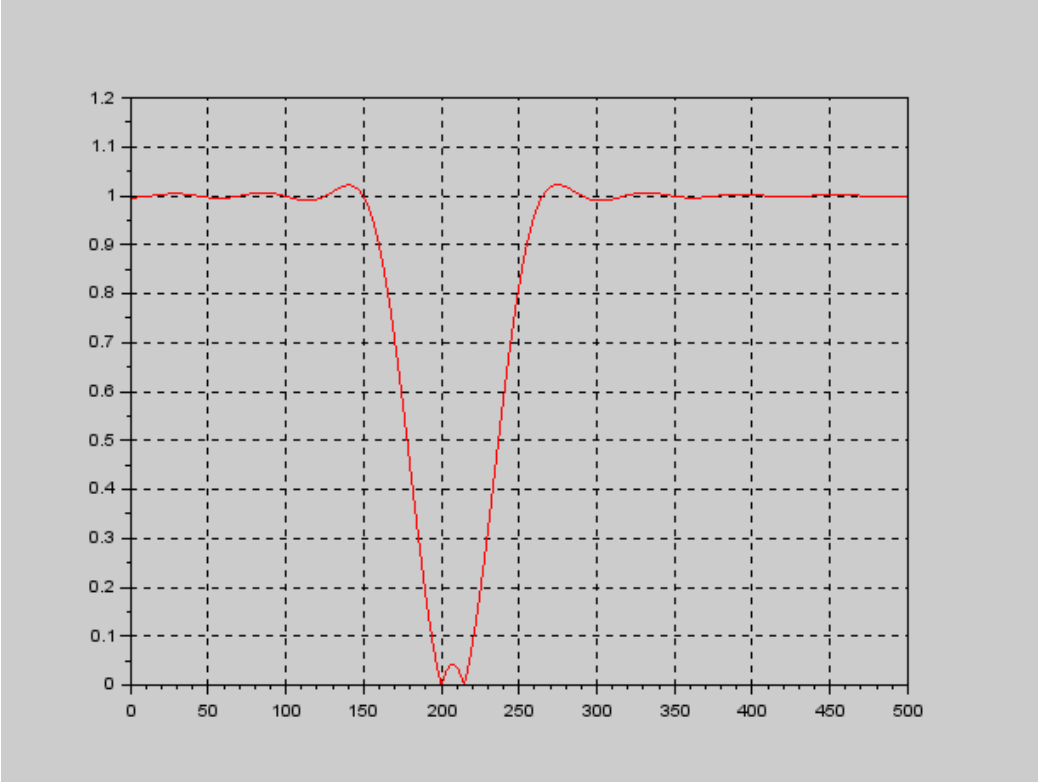


Figura 5 – Respuesta en frecuencia del filtro notch de 200 Hz.

De esta manera, el audio de entrada:

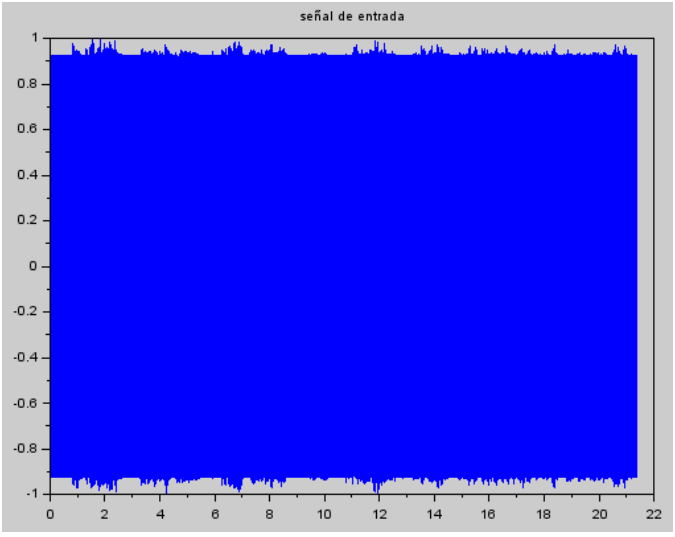


Figura 6 – Gráfico en el tiempo del audio de entrada.

Y el de salida:

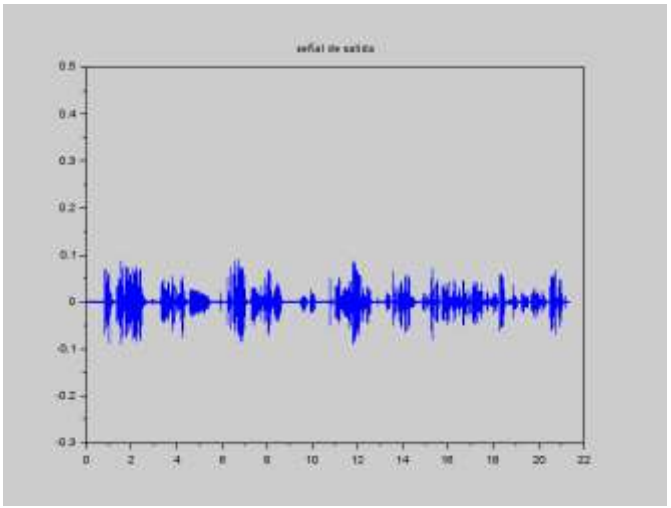


Figura 7 – Gráfico en el tiempo del audio de salida. Se puede apreciar la ausencia del tono de 200 Hz.

5.3.2 - Para el audio de 500 Hz:

Luego de varios ajustes, se estableció un A = 7,39 KHz; un f0 = 7927,3 Hz; un D = 477,3 Hz y un B y E de 20 Hz como los valores más apropiados para lograr una banda lo más estrecha posible (dentro de lo razonable, ya que aumentar el valor de N aumenta considerablemente el tiempo de procesamiento) que anule el tono de 500 Hz.

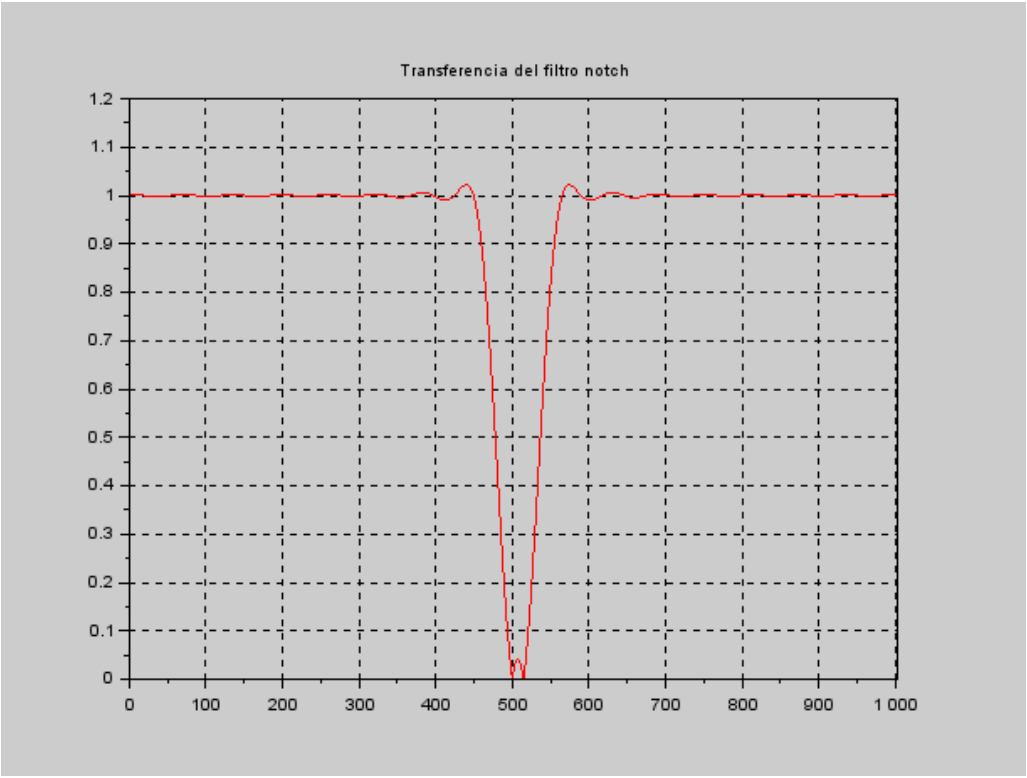


Figura 8 – Respuesta en frecuencia del filtro notch de 500 Hz.

De esta manera, la entrada de audio:

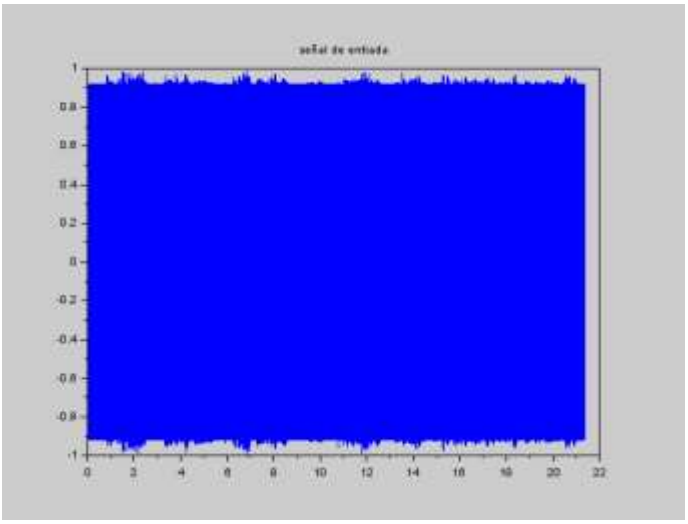


Figura 9 – Gráfico en el tiempo del audio de entrada.

Y la salida:

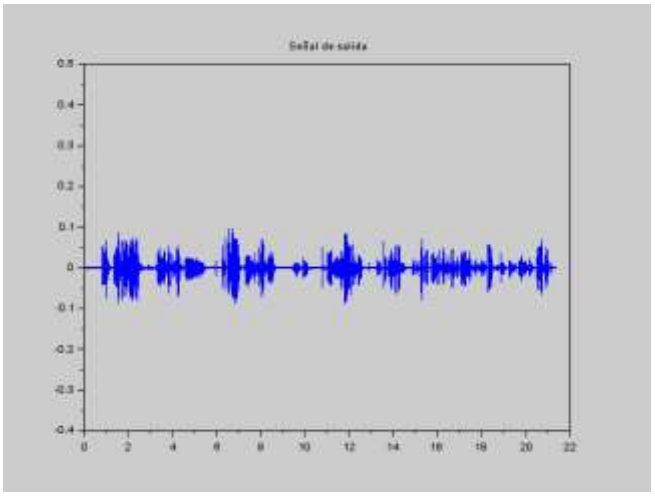


Figura 10 – Gráfico en el tiempo del audio de salida.

6. Anexo: Código.

6.1 – Filtro Notch 200 Hz

```
function y=negativePolyval(p, x)
    y = 0
    p = p(length(p):-1:1)
    for k = 1:length(p)
        y = y + p(k) * x.^(1 - k)
    end
endfunction

function g=getGain(x)
    g = 1 / max(abs(x))
endfunction

function y=sincPi(x)
    y = sinc(x*%pi)
endfunction

function h=lowPassFilter(A, B, t)
    h = 2*A*sincPi(2*A*t).*sincPi(2*B*t)
endfunction

function h=bandPassFilter(A, B, f0, t)
    h = 2*lowPassFilter(A, B, t).*cos(2*%pi*f0*t)
endfunction

function y=filterFIR(VN, x) //Recibo 1 vector REAL (VN:VectorNumerador) y la señal de entrada (x).
    //Sacamos longitudes:
    LongNum = length(VN);    //Longitud del vector del NUMERADOR.
    LongX = length(x);       //Longitud de nuestra función de entrada x.
    x2 = [zeros(LongNum-1,1); x'];    //Usamos x2 como auxiliar. zeros(2,3) me devuelve una matriz de ceros de 2 filas x 3 columnas. x' es la traspuesta de x.
    //Y la ";" es para concatenar ambos vectores. Esto lo hacemos porque en el for se abajo (al sacar y[n]) se necesita que los dos vectores tengan el mismo tamaño; entonces agrandamos el tamaño de x rellenandolo con 0s.
    //Ahora hacemos un bucle for para obtener nuestra salida y[n]:
    for n = LongNum:LongX+LongNum-1 //for basado en nuestra ecuacion H(z)=B(Z)/A(Z) --> ver help filter(), es un for que va de n=Maximo HASTA Maximo+Nx-1.
        y(n) = ( (VN(LongNum:-1:1))*(x2(n-LongNum+1:n)) ); //Al hacer: 8:-1:4 (ejemplo) nos devuelve 8,7,6,5,4 (disminuimos el 8 en 1 hasta llegar al 4).
    end

    //Salida...
    y2 = y(LongNum:LongX+LongNum-1); //Nos sirve para eliminar los 0s.
    y = y2'; //La trasponemos y obtenemos nuestra señal x ya filtrada = señal y.
endfunction

[x, fs, bits] = wavread('.\Audio200Hz\ParlaProfeTESundavCONtono200Hz_1erCuat2019.wav');
A = 7390 // Ancho del filtro pasabanda
B = 20
D = 177.3 // Ancho del filtro pasabajos
E = 20
N = 700 // Maxima cantidad de terminos positivos de T.Fourier en tiempo discreto
Ts = 1 / fs;
k = 0:2*N

// Se define el filtro notch de 200Hz compuesto por un filtro pasa bajos y un filtro pasa banda.
hc = lowPassFilter(D, E, (k - N)*Ts) + bandPassFilter(A, B, 7627.3, (k - N)*Ts)

// Se grafica la respuesta en frecuencia del filtro hc
f = 0:0.1:fs
Z = exp(%i*2*%pi*f*Ts)
Hz = negativePolyval(Ts*hc, Z)
figure(0)
title('Transferencia del filtro notch')
plot(f, abs(Hz), 'r')
xgrid()

// Se usa hc para filtrar una señal de audio
t = (1:length(x))*Ts;
y = filterFIR(Ts*hc, x)  // denominador = 1 para que sea filtro FIR.
```

```
// Se grafica la señal de entrada x
figure(1)
title('Señal de entrada')
plot(t, x)

// Se grafica la señal filtrada y
figure(2)
title('Señal de salida')
plot(t, y)

wavwrite(y, fs, bits, '.\Audio200Hz\TPFiltroFIRAaudio200Hz_out.wav')
playsnd(y*getGain(y), fs, bits, '')
```

6.2 – Filtro Notch 500 Hz

```
function y=negativePolyval(p, x)
    y = 0
    p = p(length(p):-1:1)
    for k = 1:length(p)
        y = y + p(k) * x.^(1 - k)
    end
endfunction

function g=getGain(x)
    g = 1 / max(abs(x))
endfunction

function y=sincPi(x)
    y = sinc(x*%pi)
endfunction

function h=lowPassFilter(A, B, t)
    h = 2*A*sincPi(2*A*t).*sincPi(2*B*t)
endfunction

function h=bandPassFilter(A, B, f0, t)
    h = 2*lowPassFilter(A, B, t).*cos(2*%pi*f0*t)
endfunction

function y=filterFIR(VN, x) //Recibo 1 vector REAL (VN:VectorNumerador) y la señal de entrada (x).
    //Sacamos longitudes:
    LongNum = length(VN);    //Longitud del vector del NUMERADOR.
    LongX = length(x);       //Longitud de nuestra función de entrada x.
    x2 = [zeros(LongNum-1,1); x'];    //Usamos x2 como auxiliar. zeros(2,3) me devuelve una matriz de ceros de 2 filas x 3 columnas. x' es la traspuesta de x.
    //Y la ";" es para concatenar ambos vectores. Esto lo hacemos porque en el for se abajo (al sacar y[n]) se necesita que los dos vectores tengan el mismo tamaño; entonces agrandamos el tamaño de x rellenandolo con 0s.
    //Ahora hacemos un bucle for para obtener nuestra salida y[n]:
    for n = LongNum:LongX+LongNum-1 //for basado en nuestra ecuacion H(z)=B(Z)/A(Z) --> ver help filter(), es un for que va de n=Maximo HASTA Maximo+Nx-1.
        y(n) = ( (VN(LongNum:-1:1))*(x2(n-LongNum+1:n)) ); //Al hacer: 8:-1:4 (ejemplo) nos devuelve 8,7,6,5,4 (disminuimos el 8 en 1 hasta llegar al 4).
    end

    //Salida...
    y2 = y(LongNum:LongX+LongNum-1); //Nos sirve para eliminar los 0s.
    y = y2'; //La trasponemos y obtenemos nuestra señal x ya filtrada = señal y.
endfunction

[x, fs, bits] = wavread(' .\Audio500Hz\ParlaProfeTESundavCONtono500Hz_1erCuat2019.wav');
A = 7390 // Ancho del filtro pasabanda
B = 20
D = 477.3 // Ancho del filtro pasabajos
E = 20
N = 700 // Maxima cantidad de terminos positivos de T.Fourier en tiempo discreto
Ts = 1 / fs;
k = 0:2*N

// Se define el filtro notch de 500Hz compuesto por un filtro pasa bajos y un filtro pasa banda.
hc = lowPassFilter(D, E, (k - N)*Ts) + bandPassFilter(A, B, 7927.3, (k - N)*Ts)
```

```
// Se grafica la respuesta en frecuencia del filtro hc
f = 0:0.1:fs
Z = exp(%i*2*%pi*f*Ts)
Hz = negativePolyval(Ts*hc, Z)
figure(0)
title('Transferencia del filtro notch')
plot(f, abs(Hz), 'r')
xgrid()

// Se usa hc para filtrar una señal de audio
t = (1:length(x))*Ts;
y = filterFIR(Ts*hc, x)  // denominador = 1 para que sea filtro FIR.

// Se grafica la señal de entrada x
figure(1)
title('Señal de entrada')
plot(t, x)

// Se grafica la señal filtrada y
figure(2)
title('Señal de salida')
plot(t, y)

wavwrite(y, fs, bits, '.\Audio500Hz\TPFiltroFIRAaudio500Hz_out.wav')
playsnd(y*getGain(y), fs, bits, '')
```