DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Evaluación 2

PROCESAMIENTO DE SEÑALES E IMÁGENES

Profesores:

- Violeta Chang C.
- Leonel E. Medina

Ayudante: Luis Corral

Alumno: John Serrano C.

Problema 1

La auralización es un proceso en el cual una señal de audio, para este ejercicio una voz, es reproducida con las características de la respuesta al impulso de un sistema, en este caso un recinto. Una simplificación de la auralización nos permite obtener la señal discreta de salida $y[n]$ a partir de la convolución de la señal de voz de entrada $x[n]$ y la respuesta al impulso $h[n]$ medida en el recinto. Para esto, ambas señales discretas $x[n]$ y $h[n]$ deben tener la misma frecuencia de muestreo.

$$y[n] = x[n] * h[n]$$

Obtenga las señales $x[n]$ y $h[n]$ a partir de la lectura de los archivos "HellOfAGuy_Voc_EDIT.wav" y "x06y00_DS.wav" respectivamente. Verifique que las frecuencias de muestreo sean iguales. Luego obtenga la señal de salida $y[n]$ utilizando la función conv de Matlab.

Precaución al reproducir la señal $h[n]$ (o sus resultados) en audífonos o altavoces ya que posee alta energía en un corto periodo de tiempo, lo que puede ocasionar daño a su sistema de reproducción o más importante a sus oídos.

Problema 2

Una señal de voz ruidosa puede dificultar su entendimiento. Para este problema, la señal "Harvard_list_01_NOISE.wav" está contaminada con ruido eléctrico de baja frecuencia (*hum noise* o *ground noise*) y de alta frecuencia (*hiss noise*). Diseñe e implemente un filtro que elimine el ruido de la señal, grafique la respuesta de frecuencia de la señal e indique la o las frecuencias de corte del o los filtros utilizados en la misma figura. Adicionalmente, explique el diseño del o los filtros utilizados y justifique la elección de la o las frecuencias de corte.

Problema 3

Diseñe y procese la señal $x[n]$ de ruido blanco "white_noise_263s_Matlab10_EDIT.wav" con un filtro Butterworth pasa bajo de orden 6 y frecuencia de corte 1000Hz con la función butter de Matlab. Luego de obtener los coeficientes a y b, utilice la función impz para obtener la respuesta al impulso $h[n]$ del filtro. Filtre la señal mediante la convolución $y[n] = x[n] * h[n]$ y mediante $y[n] \leftarrow \mathcal{F}^{-1} \leftarrow Y(e^{jw}) = X(e^{jw})H(e^{jw})$ utilizando las funciones conv y fft (e ifft) respectivamente. Utilizando la función timeit, verifique que filtrar las señales mediante convolución toma más tiempo que utilizando las transformadas discretas de Fourier.

Referencias

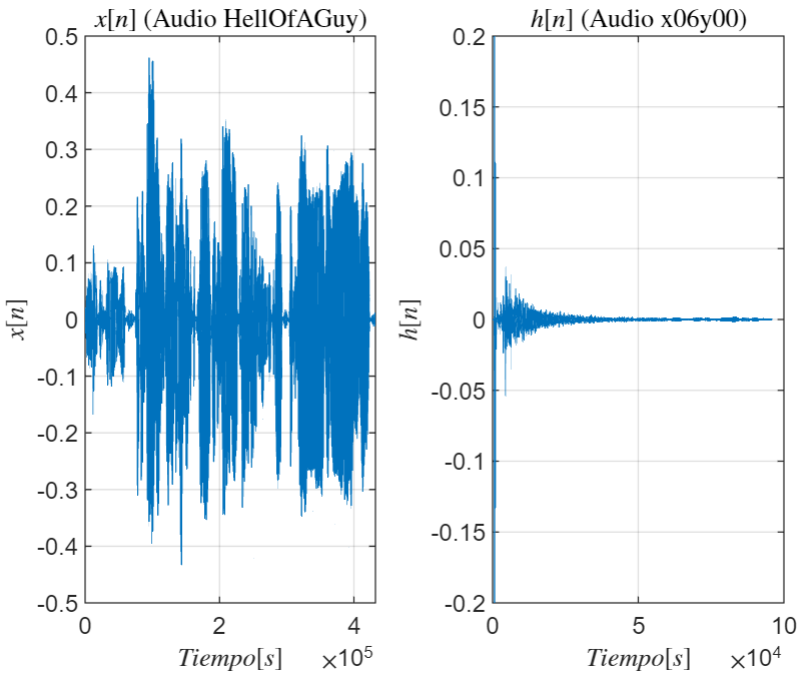
[1] Oppenheim, A.V. & Willsky, A.S. & Nawab, S.H. (1997). Señales y sistemas (2nd ed.). Prentice Hall.

Desarrollo Problema N°1

Lo primero que se debe hacer es leer los archivos **"HellOfAGuy_Voc_EDIT.wav"** y **"x06y00_DS.wav"** utilizando la función audioread(), para así poder obtener $y[n] = x[n] * h[n]$.

```
clearvars

% Lectura de archivos de audio
[x, Fsx] = audioread("HellOfAGuy_Voc_EDIT.wav"); % Se lee el archivo para x[n]
```

```
[h,Fsh] = audioread("x06y00_DS.wav"); % Se lee el archivo para h[n]


figure                          % Se inicializa una figura para graficar
subplot(1,2,1)                  % Se inicializa un subgráfico
plot(x)                         % Se grafica x[n]

% Configuraciones adicionales del subgráfico
title('$x[n]$ (Audio HellOfAGuy)', 'interpreter', 'latex')
xlabel('$$Tiempo [s]$$', 'interpreter', 'latex')
ylabel('$$x[n]$$','interpreter','latex')
grid on

subplot(1,2,2)                  % Se inicializa un subgráfico
plot(h)                         % Se grafica h[n]
ylim([-0.2 0.2])                % Se definen los límites para el eje y entre -0.2 y 0.2

% Configuraciones adicionales del subgráfico
title('$h[n]$ (Audio x06y00)', 'interpreter', 'latex')
xlabel('$$Tiempo [s]$$', 'interpreter', 'latex')
ylabel('$$h[n]$$','interpreter','latex')
grid on
```



En la variable Fsx se guarda la frecuencia de muestreo de la señal $x[n]$ y en la variable Fsh se guarda la frecuencia de muestreo de la señal $h[n]$. Al acceder y comparar los valores, podemos observar que se cumple el hecho de que ambas frecuencias de muestreo son iguales.

```
Fsx    % Frecuencia de muestreo de x[n]
```

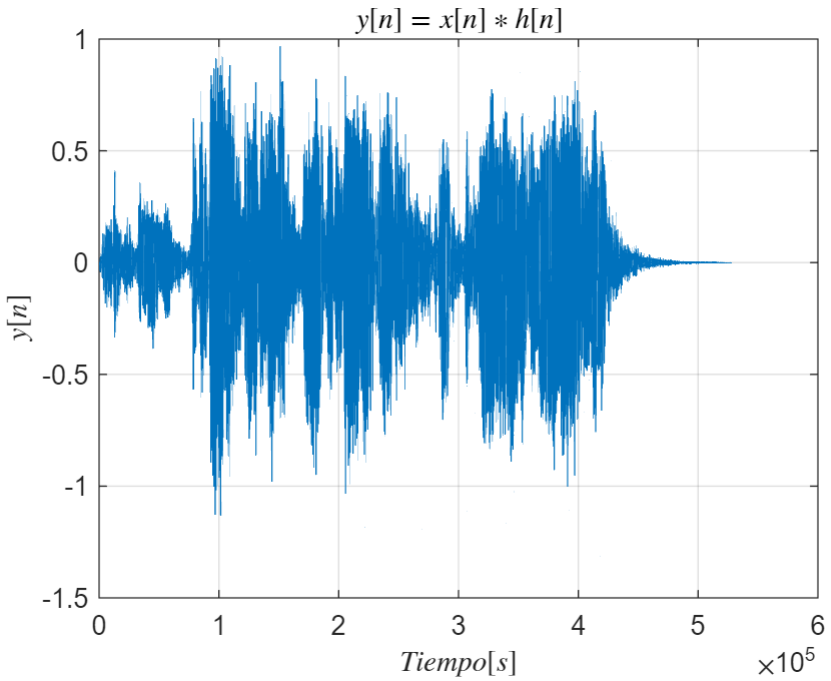
```
Fsx = 48000
```

```
Fsh    % Frecuencia de muestreo de h[n]
```

```
Fsh = 48000
```

Ahora, para realizar la convolución entre $x[n]$ y $h[n]$, se puede utilizar la función conv() que ya viene por defecto con Matlab. Se sabe (de la teoría y especialmente de la sesión de laboratorio 4.5) que el resultado de la función debe tener un largo $N_1 + N_2 - 1$ muestras, siendo N_1 y N_2 los largos de, para este caso, $x[n]$ y $h[n]$. Lo anterior se cumple si ocupamos el atributo **"full"** para la función conv().

```
% Convolución de x e h con el atributo full
y = conv(x,h, 'full');

% Gráfico de la convolución
figure;
plot(y)

% Configuraciones adicionales del gráfico
title('$y[n] = x[n] * h[n]$', 'interpreter', 'latex')
xlabel('$$Tiempo [s]$$', 'interpreter', 'latex')
ylabel('$$y[n]$$','interpreter','latex')
grid on
```



Al observar el Workspace de Matlab, se puede notar que se proporcionan los largos de $x[n]$, $h[n]$ e $y[n]$. El largo de la última es 528000, lo cual es equivalente:

$$432001 + 96000 = 528001$$

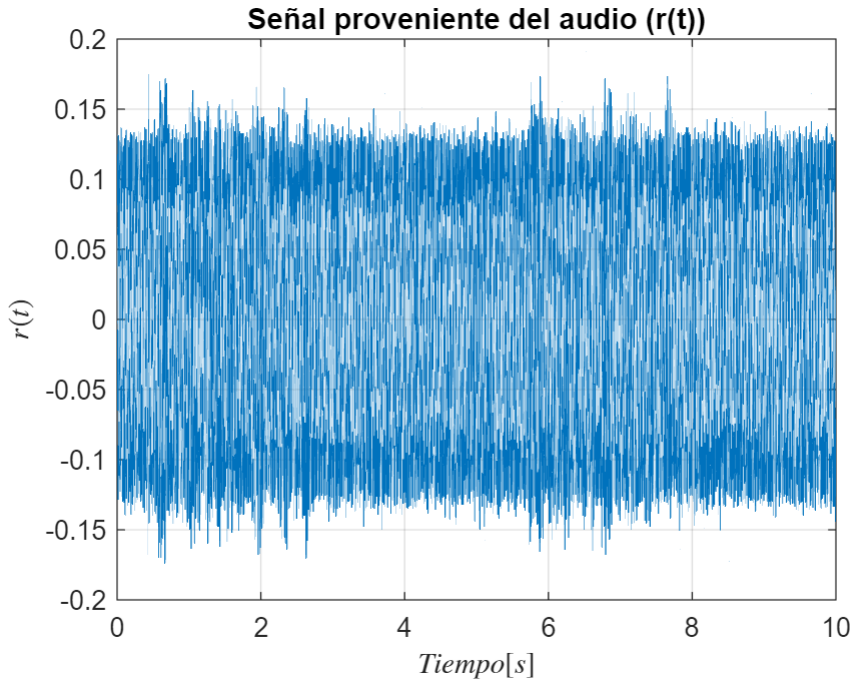
$$528001 - 1 = \mathbf{528000}$$

Con esto, se comprueba que la convolución anterior cumple con la característica del largo de la señal resultante de la convolución.

Desarrollo Problema N°2

Lo primero es leer el archivo **"Harvard_list_01_NOISE.wav"** que contiene la señal con ruido. Vamos a llamar a esta señal $r(t)$.

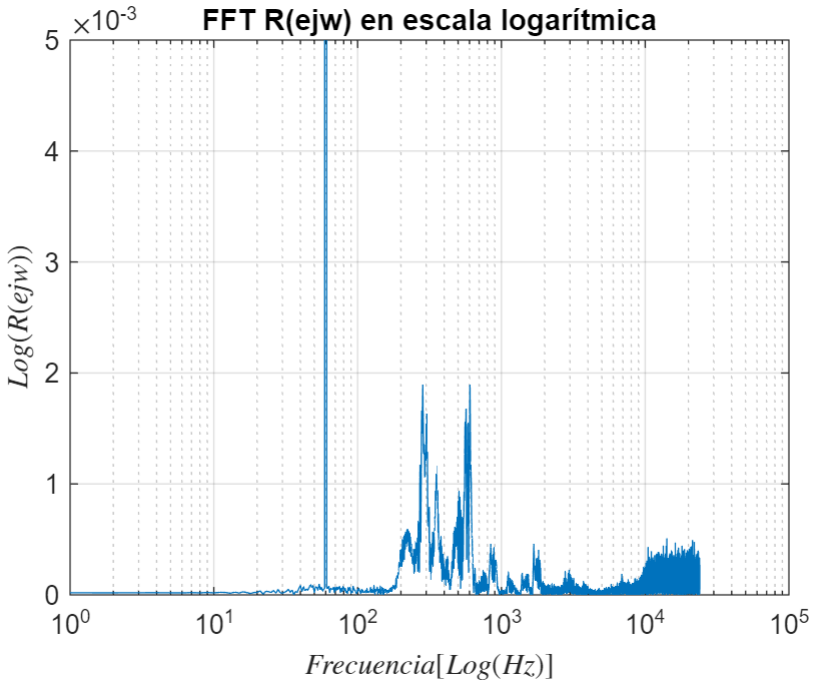
```
% Lectura del archivo de audio
[r, Fsr] = audioread("Harvard_list_01_NOISE.wav");
% sound(r,Fsr)     % DESCOMENTAR ESTA LINEA PARA ESCUCHAR EL AUDIO
r = r.';           % Se traspone el arreglo de muestras
d = length(r)/Fsr; % Se calcula la duración del audio
t = linspace(0,d,length(r)); % Se crea un arreglo de tiempo para la señal

% Gráfico de la señal r(t)
figure;
plot(t,r)

% Configuraciones adicionales del gráfico
title('Señal proveniente del audio (r(t))')
xlabel('$$Tiempo [s]$$', 'interpreter', 'latex')
ylabel('$$r(t)$$','interpreter','latex')
grid on
```



Si se reproduce la señal, se puede escuchar claramente que hay un Hiss y un Hum Noise que impiden que la voz se pueda escuchar claramente. Se busca construir un filtro que elimine el ruido de esta señal. Para ello, vamos a utilizar la FFT e IFFT, junto con filtros Pasa Bajos y Pasa Altos. Tomando como base la sesión de

laboratorio 4.5, se debe obtener la FFT de $r(t)$ y luego graficarla para ver cuáles son las frecuencias de corte para los filtros.

```
% FFT de r(t) y grafico

r_n = r(1:Fsr); % Se guarda 1 segundo de la señal r(t), definiendo asi r_n(t)
R = fft(r_n); % Se obtiene la FFT de r_n(t)
N = length(r_n); % Se calcula el largo de r_n(t)
R_log = 2*abs(R(1:round(N/2)+1))/N; % Se pasa R a una escala logaritmica
f_hz = Fsr *(0:round(N/2))/N; % Se obtiene un arreglo de la frecuencia en Hz para graficar

% Gráfico de la FFT en escala logarítmica
figure % Se define una figura para graficar
semilogx(f_hz,R_log) % Se grafica en escala logarítmica
ylim([0 0.005])  % Se definen los límites del eje y entre 0 y 0.005

% Configuraciones adicionales del gráfico
title('FFT R(ejw) en escala logarítmica')
xlabel('$$Frecuencia [Log(Hz)]$$', 'interpreter', 'latex')
ylabel('$$Log(R(ejw))$$','interpreter','latex')
grid on
```



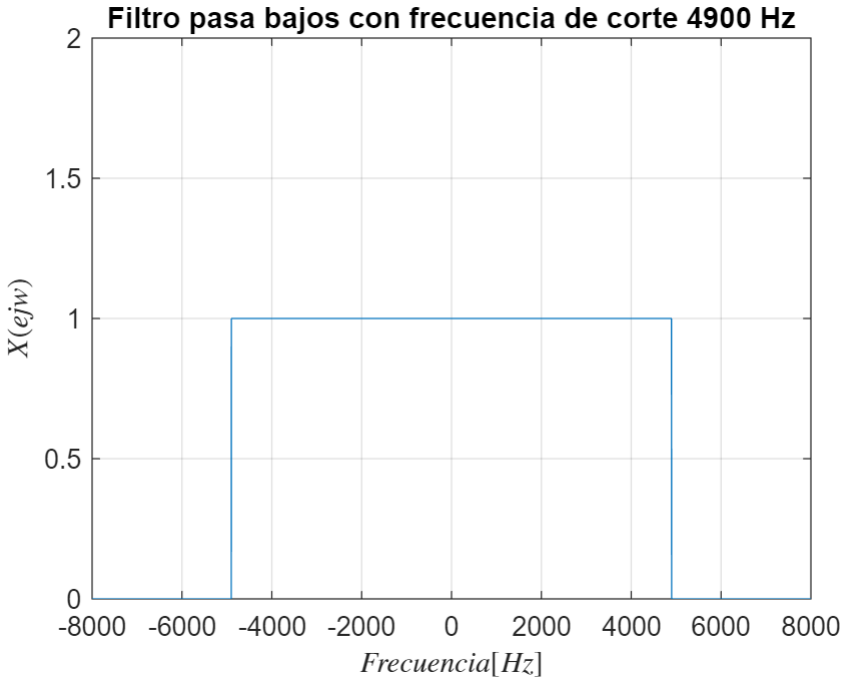
Si se observa el gráfico con atención, se puede notar que las frecuencias más bajas están cercanos a los 4500 - 5500 Hz. Considerando este dato, se puede idear un filtro pasa bajos que logre filtrar todas las frecuencias menores o iguales a cierta frecuencia. Ahora se va a obtener y trabajar con la FFT de toda la señal r(t).

```
R= fftshift(fft(r));  % Se calcula la FFT de r(t)
% Nótese que se aplica un fftshift a la FFT de r(t) para así poder
% visualizar el gráfico de la FFT de r(t) centrada en el origen
```

```
% Se crea un arreglo de la frecuencia para la FFT
f = linspace(-Fsr/2,Fsr/2,length(R));
```

Ahora podemos crear el filtro pasa bajos para así poder filtrar el Hiss Noise. Vamos a utilizar **una frecuencia de corte de 4900 Hz**, ya que, si se observa con atención el gráfico realizado anteriormente, se puede ver que corresponde a una de las frecuencias más bajas de la señal.

```
% Se crea el filtro pasa bajos, con frecuencia de corte 4900 Hz.
filtro_bajo = 1.*(abs(f)<=4900);

% Gráfico del filtro pasa bajos

figure; % Se inicializa una figura para graficar
plot(f,filtro_bajo) % Se grafica el filtro pasa bajos
xlim([-8000 8000]) % Se definen los límites del eje x entre -8000 y 8000
ylim([0 2]) % Se definen los límites del eje y entre 0 y 2

% Configuraciones adicionales del gráfico
title('Filtro pasa bajos con frecuencia de corte 4900 Hz')
xlabel('$$Frecuencia [Hz]$$', 'interpreter', 'latex')
ylabel('$$X(ejw)$$','interpreter','latex')
grid on
```



Aplicando el filtro a la FFT de $r(t)$ y luego aplicando la **IFFT** para volver al dominio del tiempo, se obtiene la señal sin el ruido de frecuencia alta:

```
% Se aplica el filtro pasa bajos a la FFT de R
R_filtro_bajo = R .*filtro_bajo;
```

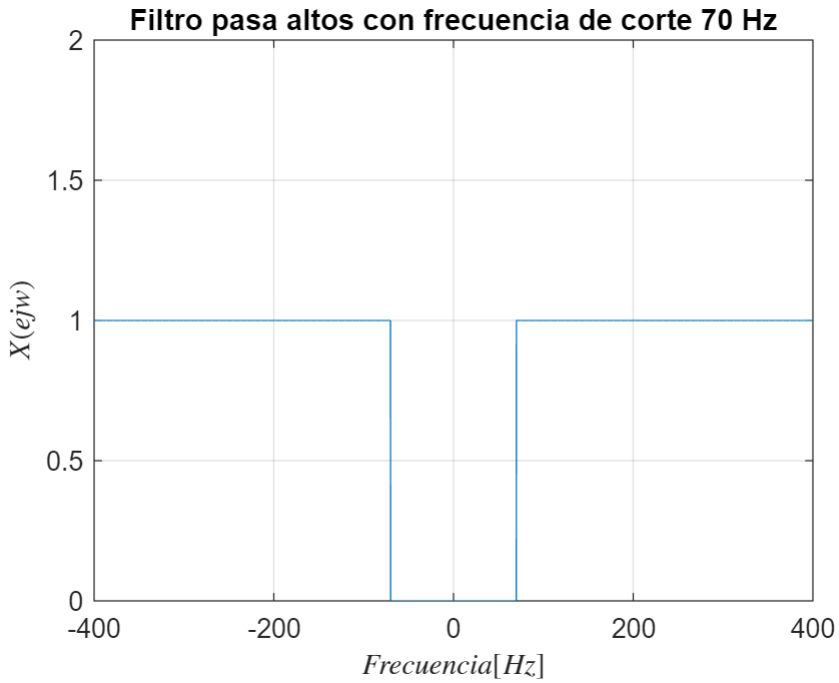
```
% Se utiliza la IFFT para volver al dominio del tiempo
r_sin_alto = ifft(ifftshift(R_filtro_bajo));

% Como lo anterior es complejo, solo nos interesa la parte real
r_sin_alto= real(r_sin_alto);

% Se puede escuchar el audio para comprobar el estado de la señal
% sound(r_sin_alto, Fsr)
```

Ahora se debe repetir el proceso para eliminar el ruido de frecuencia baja:

```
% Se obtiene la FFT de la señal sin el ruido de frecuencia alta
R_sin_alto = fftshift(fft(r_sin_alto));
% Se obtiene la frecuencia para la FFT anterior
f_sin_alto = linspace(-Fsr/2,Fsr/2,length(r_sin_alto));
```

Vamos a aplicar un filtro pasa altos con una **frecuencia de corte de 70 Hz**, ya que es una de las frecuencias más altas si se observa la gráfica de la FFT obtenida anteriormente.

```
% Se crea el filtro pasa altos, con frecuencia de corte 70 Hz
filtro_alto = 1.*(abs(f_sin_alto)> 70);

% Gráfico del filtro pasa altos
figure; % Se inicializa una figura para graficar
plot(f_sin_alto,filtro_alto) % Se grafica el filtro pasa altos
xlim([-400 400]); % Se definen los límites del eje X entre -400 y 400
ylim([0 2]); % Se definen los límites del eje y entre 0 y 2

% Configuraciones adicionales del gráfico
title('Filtro pasa altos con frecuencia de corte 70 Hz')
xlabel('$$Frecuencia [Hz]$$', 'interpreter', 'latex')
ylabel('$$X(ejw)$$','interpreter','latex')
grid on
```



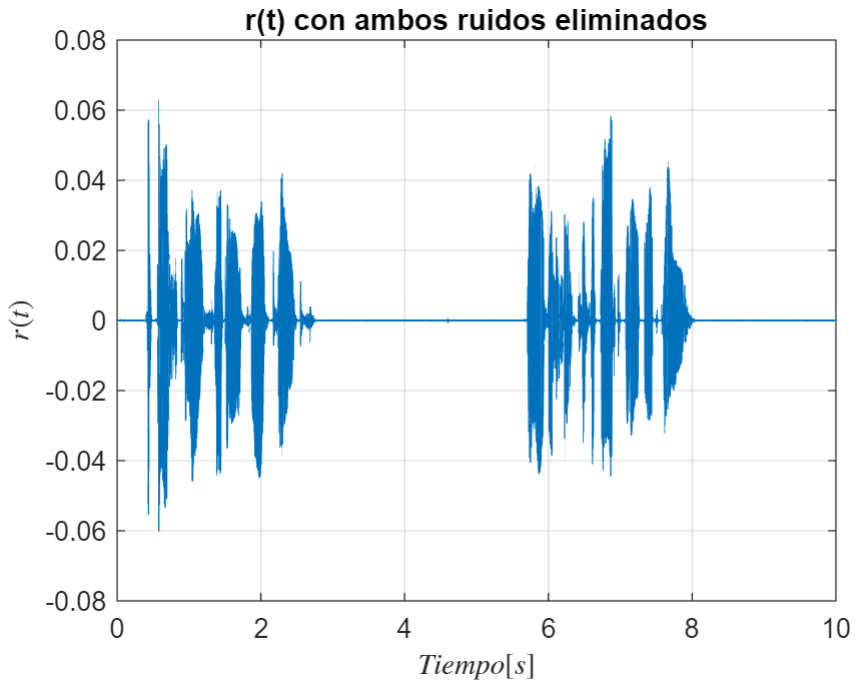
Finalmente, para poder eliminar el ruido de frecuencia baja, aplicamos el filtro pasa altos a la FFT.

```
% Se aplica el filtro pasa altos a la FFT de R
R_filtro_alto = R_sin_alto .*filtro_alto;

% Se aplica la IFFT para volver al dominio del tiempo
r_sin_ruido = ifft(ifftshift(R_filtro_alto));

% Como lo anterior es un complejo, solo interesa la parte real
r_sin_ruido= real(r_sin_ruido);

% Se puede comprobar que se han eliminado completamente ambos ruidos
% escuchando la señal. DESCOMENTAR SI SE QUIERE COMPROBAR.
% sound(r_sin_ruido, Fsr)
```

Por lo tanto, aplicando un filtro pasa bajos con frecuencia de corte 4900 Hz y un filtro pasa altos con frecuencia de corte 70 Hz, se obtienen la señal original con ambos ruidos completamente eliminados. Al graficar la señal sin ruido, se obtiene:

```
d = length(r_sin_ruido)/Fsr; % Se calcula la duración del audio
t = linspace(0,d,length(r_sin_ruido)); % Se crea un arreglo de tiempo para la señal

% Gráfico de la señal r(t)
figure;
plot(t,r_sin_ruido)

% Configuraciones adicionales del gráfico
title('r(t) con ambos ruidos eliminados')
xlabel('$$Tiempo [s]$$', 'interpreter', 'latex')
ylabel('$$r(t)$$','interpreter','latex')
```

```
grid on
```

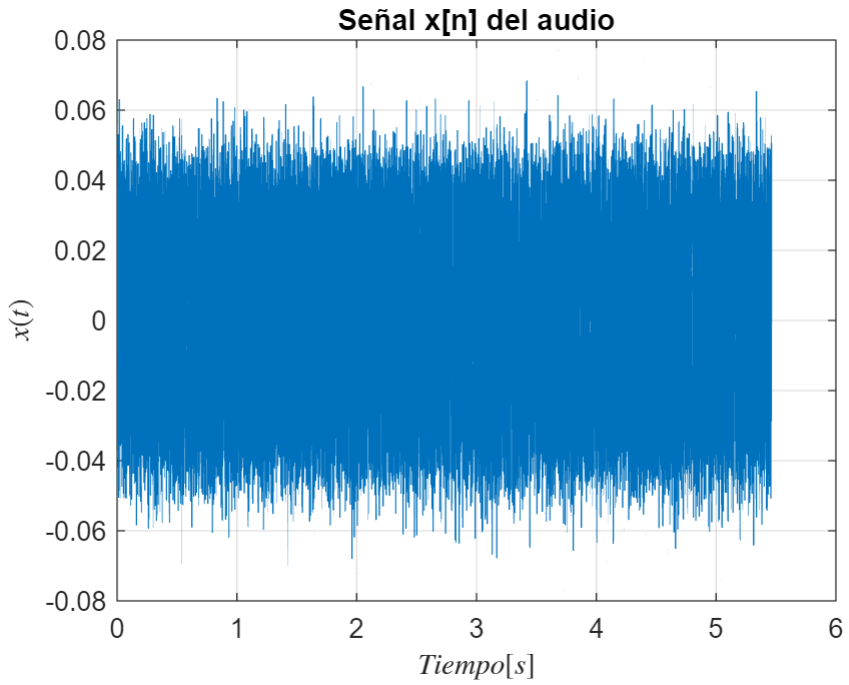


Nótese que la gráfica **muestra exactamente el comportamiento del audio cuando se considera solo la voz, sin los ruidos.** Por lo tanto, esto también comprueba que se ha logrado eliminar ambos ruidos por completo.

Desarrollo Problema N°3

Al igual que los problemas anteriores, lo primero que se debe hacer es leer el archivo de audio o **"white_noise_263s_Matlab10_EDIT.wav"** que contiene la señal $x[n]$.

```
[xn, Fsxn] = audioread("white_noise_263s_Matlab10_EDIT.wav");
d_xn = length(xn)/Fsxn; % Se calcula la duración del audio
t_xn = linspace(0,d_xn,length(xn)); % Se crea un arreglo de tiempo para la señal

% Gráfico de la señal r(t)
figure;  % Se inicializa una figura
plot(t_xn,xn) % Se grafica x[n]

% Configuraciones adicionales del gráfico
title('Señal x[n] del audio')
xlabel('$$Tiempo [s]$$', 'interpreter', 'latex')
ylabel('$$x(t)$$','interpreter','latex')
grid on
```



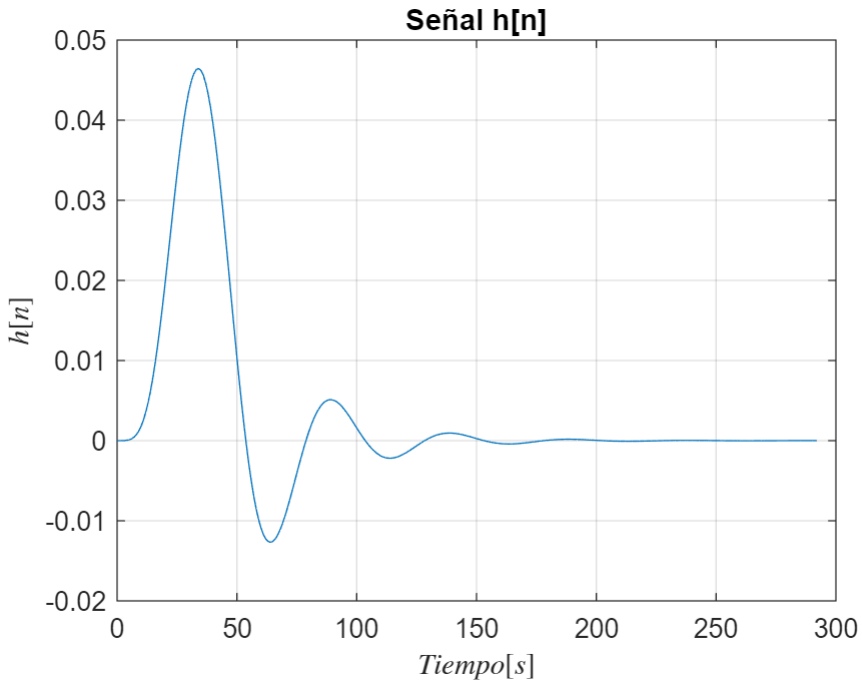
Con la señal ya definida, podemos continuar aplicando en primer lugar, el filtro Butterorth pasa bajo de orden 6 y frecuencia de corte 1000 Hz, utilizando la función **butter().** Con esto se pueden obtener los coeficientes a y b.

```
% Se definen los datos del enunciado
orden = 6;  % Cantidad de filtros en cascada
Fc = 1000;  %

% Se utiliza butter() junto al atributo "low" para obtener a y b
[b,a] = butter(orden, Fc/(Fsxn / 2), "low")
```

```
b = 1×7
10^-5 ×
    0.0062    0.0369    0.0923    0.1231    0.0923    0.0369    0.0062
a = 1×7
    1.0000   -5.4943   12.5978  -15.4285   10.6437   -3.9214    0.6028
```

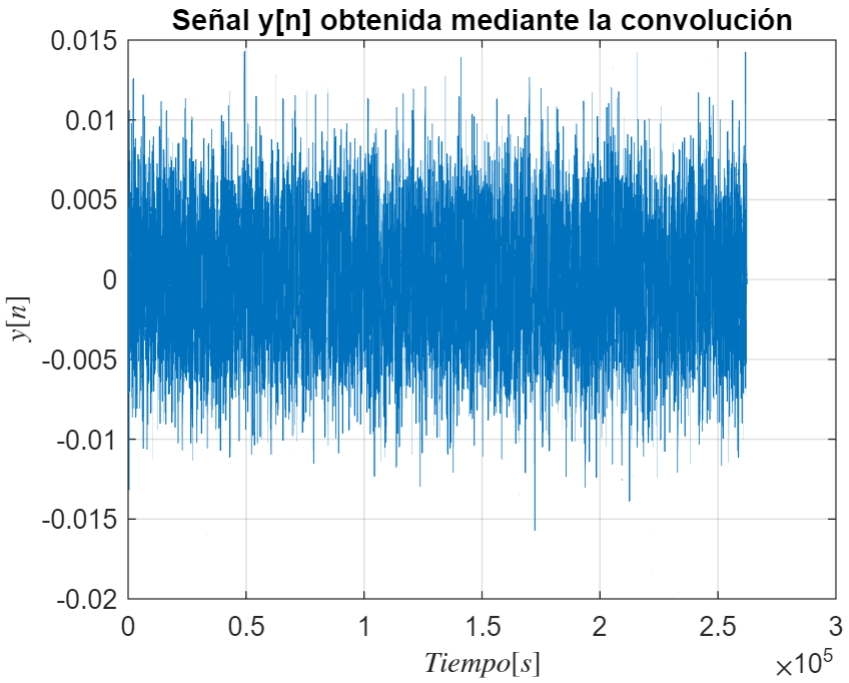
Utilizando la función **impz()** junto con los coeficientes obtenidos, se puede obtener la respuesta al impulso $h[n]$ del filtro.

```
% Se utiliza impz para obtener h[n] y su tiempo
[hn,tn] = impz(b,a);

% Gráfico de h[n]
figure; % Se inicializa una figura
plot(tn,hn) % Se realiza el gráfico de h[n]

% Configuraciones adicionales del gráfico
title('Señal h[n]')
xlabel('$$Tiempo [s]$$', 'interpreter', 'latex')
ylabel('$$h[n]$$','interpreter','latex')
grid on
```



Ahora, se busca filtrar la señal $x[n]$ mediante la convolución $y[n] = x[n] * h[n]$. Similar al problema 1, es necesario utilizar la función conv() para poder llevar a cabo la convolución.

```
% y[n] = x[n] * h[n]

yconv = conv(xn, hn, "full"); % Se aplica conv con "full" para realizar la convolución

% Gráfico de y[n] obtenida mediante la convolución de x[n] y h[n]
figure; % Se inicializa una figura para graficar
plot(yconv) % Se grafica y[n]

% Configuraciones adicionales del gráfico
title('Señal y[n] obtenida mediante la convolución')
xlabel('$$Tiempo [s]$$', 'interpreter', 'latex')
ylabel('$$y[n]$$','interpreter','latex')
grid on
```



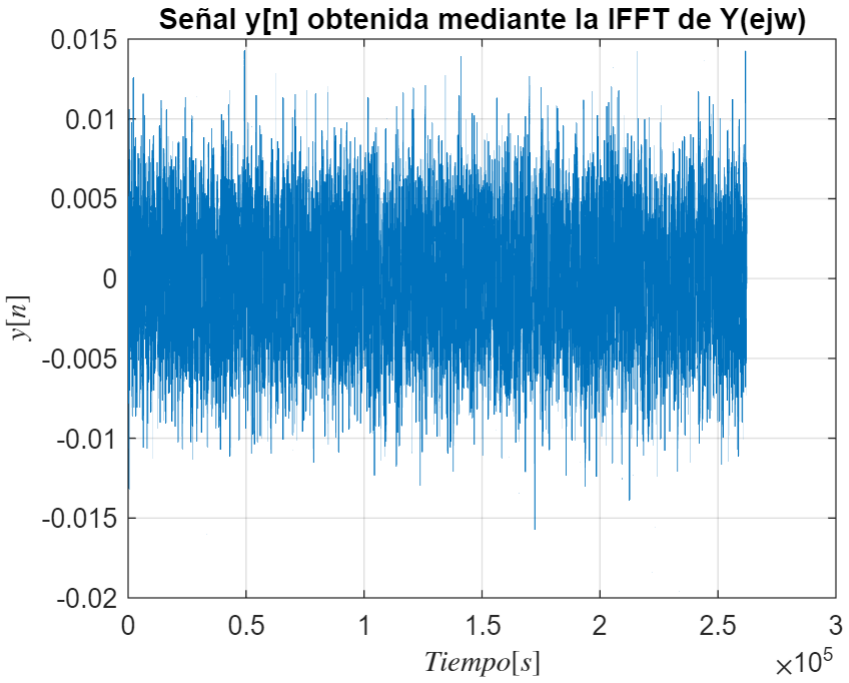
Ahora es el turno de obtener la señal $y[n]$ mediante las FFT de $x[n]$ y $h[n]$, aplicando:

$$y[n] \leftarrow \mathscr{F}^{-1} \leftarrow Y(e^{jw}) = X(e^{jw})H(e^{jw})$$

Esto se puede lograr en Matlab utilizando las funciones **fft()** e **ifft()** de Matlab.

```
% Si se intenta realizar la multiplicación de las FFTs, Matlab dará el
% siguiente error: "Arrays have imcompatible sizes for this operation". Si
% observamos el Workspace, podemos ver que h[n] tiene un largo bastante
% menor comparado con el de x[n]. Por lo tanto, se puede rellenar h[n] con
% ceros para que así tenga el mismo largo de x[n].

% Utilizando los largos de x[n] y h[n], se rellena h[n] con ceros
hn = [hn(:); zeros(length(xn) - length(hn),1)];

% Ahora si, se puede realizar la multiplicación de las FFTs
Xn = fft(xn); % Se obtiene la FFT de x[n]
Hn = fft(hn); % Se obtiene la FFT de h[n]
Yfft = Xn .* Hn; % Y(ejw) = X(ejw)H(ejw)

% Como lo anterior corresponde a la FFT de y[n], se debe aplicar la IFFT
% para poder obtener y[n]
yfft = ifft(Yfft); % Se aplica la IFFT a Y(ejw)

% Gráfico de y[n] obtenida mediante la IFFT de Y(ejw) = X(ejw)H(ejw)
figure; % Se inicializa una figura para graficar
plot(yfft) % Se grafica y[n]

% Configuraciones adicionales del gráfico
title('Señal y[n] obtenida mediante la IFFT de Y(ejw)')
```

```
xlabel('$$Tiempo [s]$$', 'interpreter', 'latex')
ylabel('$$y[n]$$','interpreter','latex')
grid on
```



De los gráficos anteriores, se puede llegar a la conclusión de que al obtener $y[n]$ con la convolución o mediante la multiplicación de las FFT de $x[n]$ y $h[n]$ y luego aplicando la IFFT, se llega a exactamente el mismo resultado. Esto comprueba el teorema de la convolución vista en la teoría, el cual dice:

$$\mathcal{F}(x[n] * h[n]) = X(e^{jw})H(e^{jw})$$

Sin embargo, a pesar de que independiente de cual método se use se llega al mismo resultado, la diferencia puede estar en el tiempo que toma en realizar cada operación. Para comprobar esto, se puede utilizar la función **timeit()** de Matlab. El supuesto inicial es que **filtrar las señales mediante convolución toma más tiempo que utilizando las transformadas discretas de Fourier.**

La función timeit() recibe una función sin atributos, por lo que se deben crear funciones que llamen a las funciones conv, FFT e IFFT para que así timeit() reciba funciones sin atributos. Estas funciones fueron creadas en los archivos **convolucion.m** y **multiplicacion.m.** Ahora, vamos a llamar a estas funciones y luego utilizar timeit().

```
% Llamado a las funciones de los archivos .m
convolucion_sin_args = @()convolucion(xn,hn); % Se llama a la función convolución
multiplicacion_sin_args = @()multiplicacion(xn,hn); % Se llama a la función multiplicación

% Ahora es posible aplicar timeit() y obtener la diferencia de tiempo
tiempo_convolucion = timeit(convolucion_sin_args) % Se aplica timeit a convolución
```

```
tiempo_convolucion = 0.0155
```

```
tiempo_multiplicacion = timeit(multiplicacion_sin_args) % Se aplica timeit a multiplicación
```

```
tiempo_multiplicacion = 0.0066
```

```
% Se obtiene la diferencia de tiempo
diferencia_tiempo = tiempo_convolucion / tiempo_multiplicacion
```

```
diferencia_tiempo = 2.3411
```

Como se puede apreciar en los resultados, efectivamente **filtrar las señales mediante convolución toma más tiempo que utilizando las transformadas discretas de Fourier.** El tiempo obtenido mediante la convolución es de 0,0217 segundos, mientras que el tiempo utilizando las transformadas discretas de Fourier es de 0,0094 segundos, claramente menor que el de la convolución. La diferencia de tiempo entre ambas operaciones es de 2,3 segundos.

En conclusión, filtrar las señales utilizando las transformadas discreta de Fourier es 2,3 segundos más rápido que utilizando la convolución, por lo que a pesar de que se llega al mismo resultado aplicando ambos métodos, si hay una diferencia de tiempo que en ciertos contextos puede ser significativa.

Referencias utilizadas para el desarrollo

Si bien, la gran mayoría del desarrollo tuvo como principal referencia la sesión 4.5 de Laboratorio realizada el día viernes 28 de octubre de 2022, también se utilizó el siguiente material como referencia:

- Tutorías con Ingenio Universidad Nacional. (2020). *Filtrado de audio en MATLAB utilizando filtros ideales.* Video Online. Recuperado de https://youtu.be/eDdQPr8cTgA el día 09 de noviembre de 2022.