Guía N° 7: *Métodos avanzados de procesamiento de señal*

Objetivo:

Incorporar nuevas técnicas para el procesamiento de señales

Fecha de entrega:

1/11/2019

Formato de entrega:

Cada ejercicio se corresponde a un archivo .m, con nombre *eN_M.m*, donde N es el nombre de la guía y M el número de ejercicio. El contenido de cada archivo .m debe contener todos los puntos correspondientes a cada ejercicio.
Todos los archivos se deberán comprimir en un solo archivo .zip o .rar y enviarse por correo con título *Entrega Guía 7.*

Bibliografía recomendada:

The Scientist and Engineer's Guide to Digital Signal Processing

1. Remuestreo

En ocasiones es necesario generar más o menos muestras que las disponibles. A esta operación se la denomina remuestreo y puede realizarse con 3 operaciones elementales

- Upsampling
- Filtrado
- Downsampling o diezmado

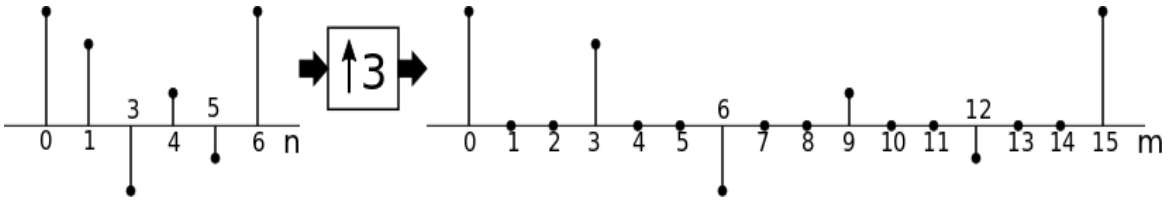
Upsampling

El upsampling por un factor *L* (o sobremuestreo en L veces) consiste en intercalar *L-1* muestras de valor 0 (cero) entre las muestra de la señal.

Matemáticamente

$$y[m] = \begin{cases} x[n] & \text{si} \quad m = Ln \\ 0 & \text{si} \quad m \neq Ln \end{cases}$$

Ejemplo para *L*=3



Vemos que en cada muestra *m* múltiplo de 3 tenemos una muestra de *x[n]* y las restantes son cero.
El efecto que tiene esta transformación en el espectro de la secuencia se puede evaluar a partir de la ecuación de transformada de Fourier discreta (DFT)

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

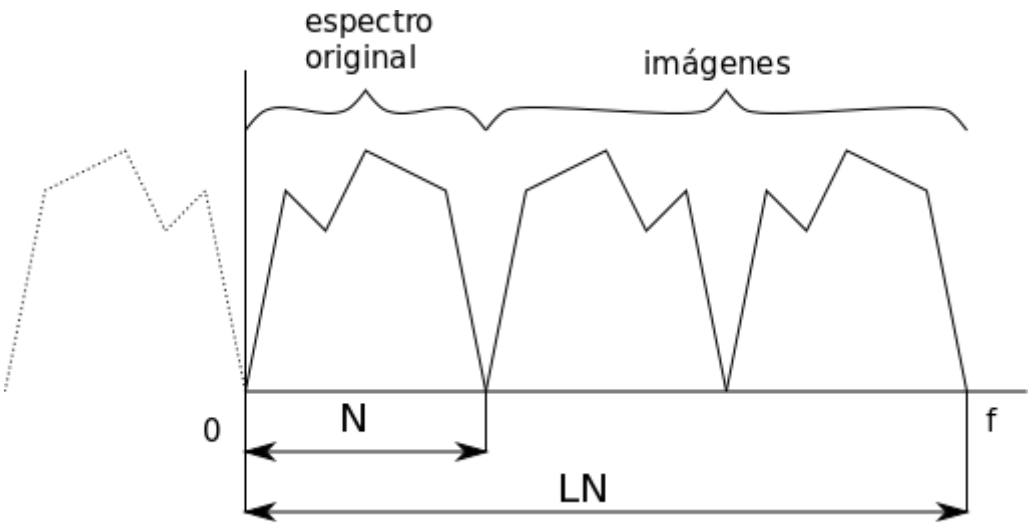
Para y[m] nos quedará

$$Y[k] = \sum_{m=0}^{LN-1} y[m] e^{-\frac{2\pi i}{LN} km} \qquad k = 0, \ldots, LN-1$$

Teniendo en cuenta que para *m=Ln* tenemos que *y[m]=x[Ln]* y los demás términos son cero, nos queda luego de simplificar

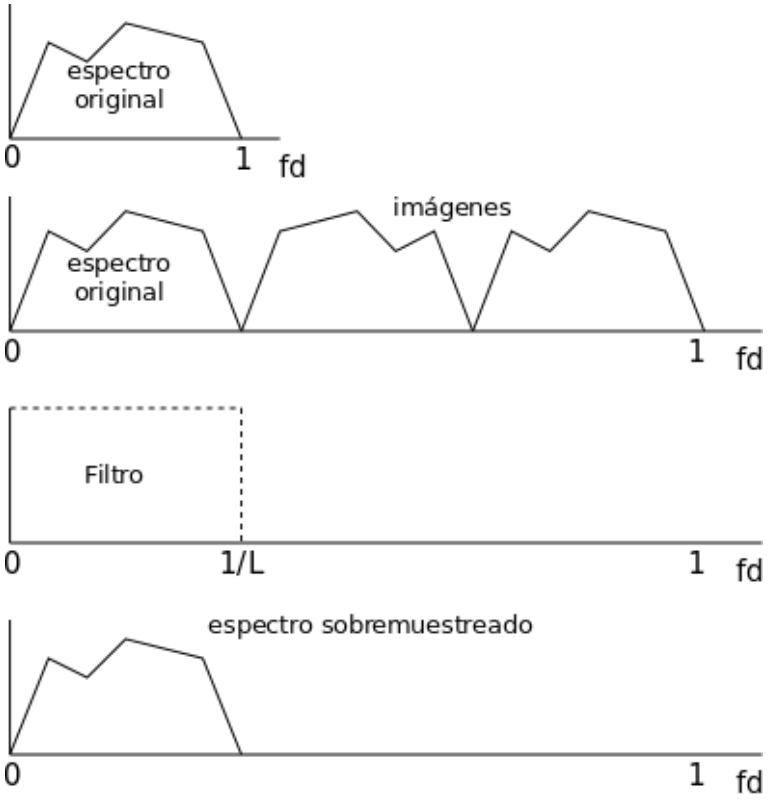
$$Y[k] = \sum_{n=0}^{N-1} x[n] e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, LN-1$$

Podemos ver que los primeros N elementos del espectro son iguales al espectro anterior, y para valores superiores volvemos a generar el espectro original, por lo que tendremos el espectro repetido L veces, como se ve en el siguiente gráfico (sólo para frecuencias positivas).

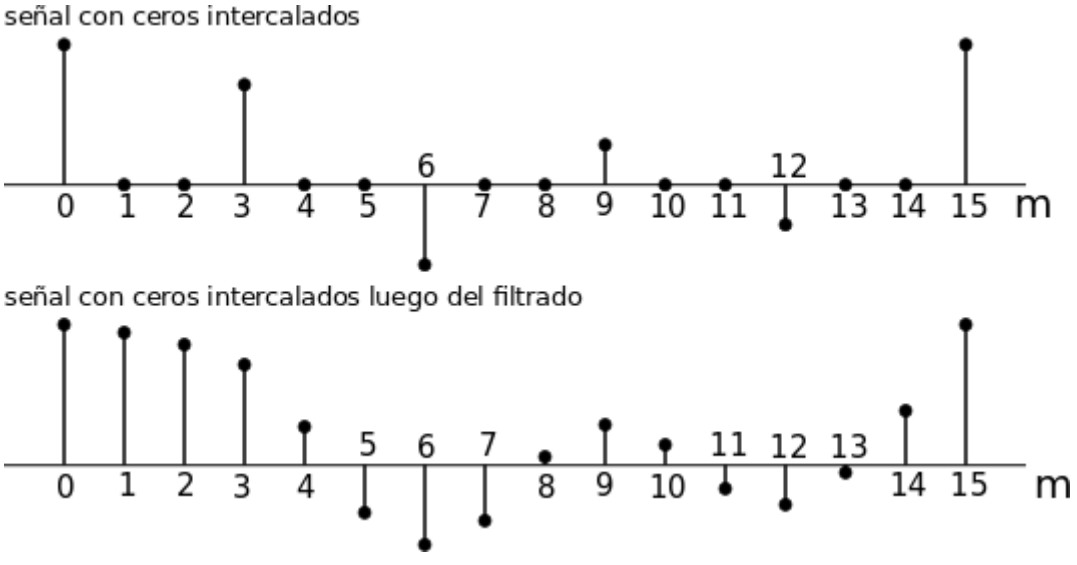


Estas componentes de frecuencia nuevas deberán eliminarse para completar el proceso de upsampling, aplicando un filtro después de introducir los ceros. Como la nueva frecuencia de muestreo es Lf_s, la frecuencia digital de corte del filtro será *1/L*.

El proceso completo, visto en frecuencia se ve a continuación:



En el tiempo, el filtrado suavizará las discontinuidades producidas por el intercalado de ceros, de manera similar a una interpolación.



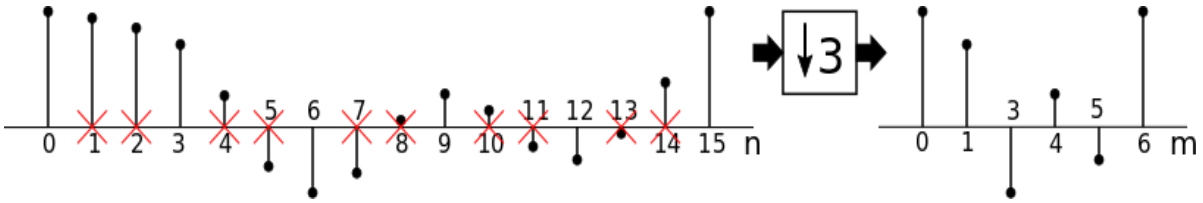
Downsampling

El downsampling por un factor M consiste en quedarse con una de cada M muestras, simplemente descartando el resto.

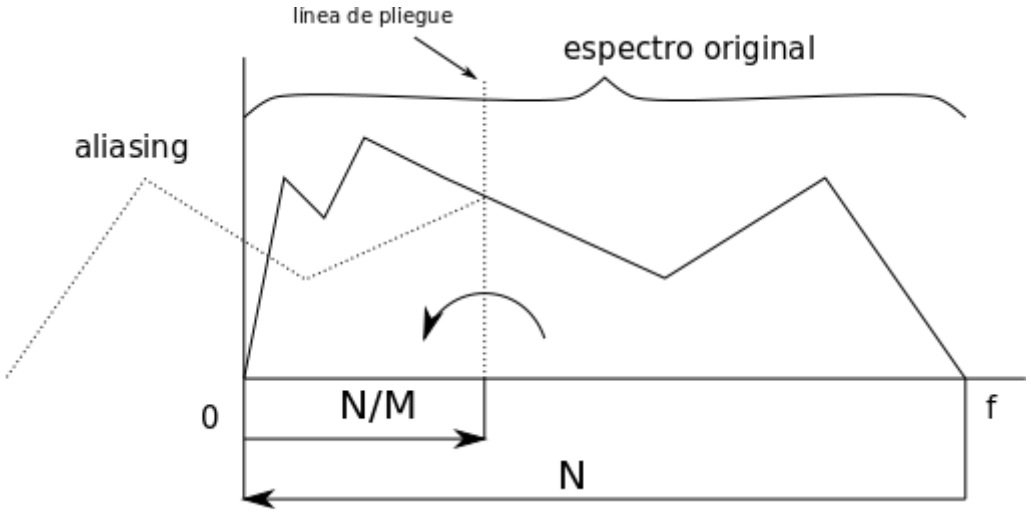
Matemáticamente

$$y[m] = x[Mm]$$

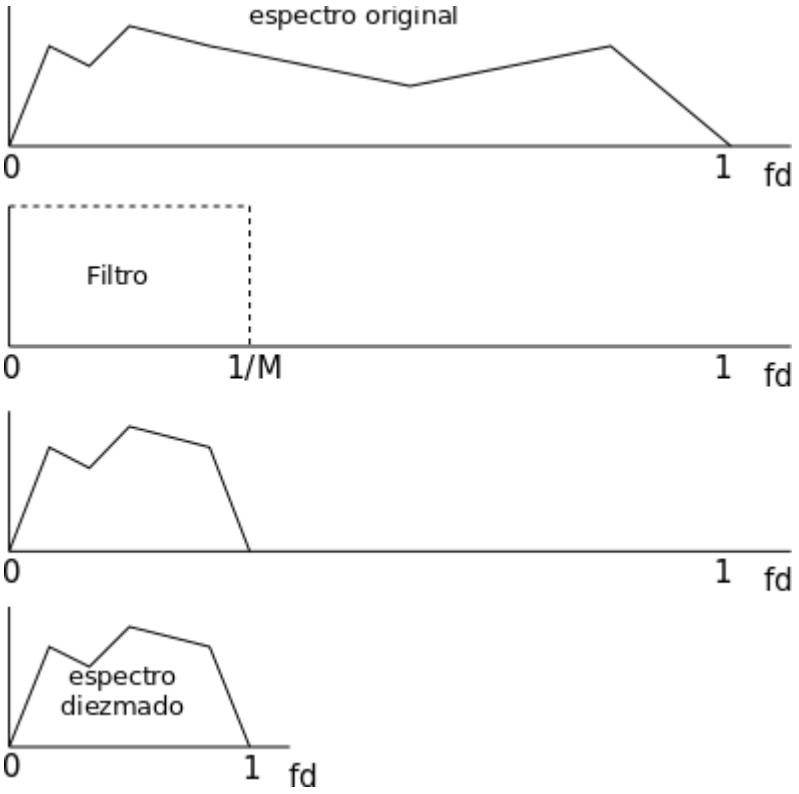
Ejemplo para M=3



Espectralmente, lo que sucede es una reducción de la frecuencia de muestreo por un factor M. Esto producirá que las componentes de alta frecuencia introduzcan aliasing sobre el espectro resultante, como se ve en la siguiente gráfica:



Por lo tanto, y al igual que el caso anterior, será necesario aplicar un filtro antes de realizar el diezmado. Podemos ver como queda entonces el proceso completo en el dominio de la frecuencia:



Por último, las 3 operaciones, upsampling, filtrado y downsampling se pueden combinar para obtener cualquier relación fraccional de conversión.

Es importante respetar el orden de las operaciones, siempre se realizará primero el intercalado de ceros, luego el filtrado, y por último el diezmado. Teniendo en cuenta que el upsampling exige una frecuencia de corte para el filtro de 1/L y que el diezmado exige otra frecuencia de corte de 1/M, se deberá elegir aquella que sea menor.

1.1. Aplicar upsampling, filtrado y downsampling para convertir el audio obtenido a partir del archivo Jahzzar_29400.wav a una frecuencia de muestreo de 44100Hz.
Los pasos a seguir son:
1) Encontrar la relación de conversión L/M
2) Realizar upsampling con la función *upsample*
3) Diseñar el filtro que corresponda y aplicarlo mediante *filter*, *conv* u *fftconv* (este último recomendando para filtros FIR sólo en Octave)
4) Aplicar el downsampling con la función *downsample*
El vector resultante deberá escucharse con el tempo correcto usando una frecuencia de muestreo de 44100Hz.

2. Modulación

Modulación es el proceso de variar uno o más parámetros de una señal periódica, llamada portadora, con una señal modulante con información**.** Para una portadora senoidal se pueden modificar la fase, la frecuencia y amplitud, denominándose modulación en fase (PM), frecuencia (FM) y amplitud (AM), respectivamente.

En este ejercicio vamos a considerar la modulación en amplitud únicamente.

Supongamos como portadora una exponencial compleja, planteada por comodidad sobre una secuencia de *N* elementos:

$$e^{\frac{2\pi i}{N}nl} \qquad n = 0, \ldots, N-1$$

donde *l* es la frecuencia de la portadora. Si multiplicamos esta portadora por una secuencia x[n], obtenemos una señal *y[n]*

$$y[n] = x[n]e^{\frac{2\pi i}{N}l}$$

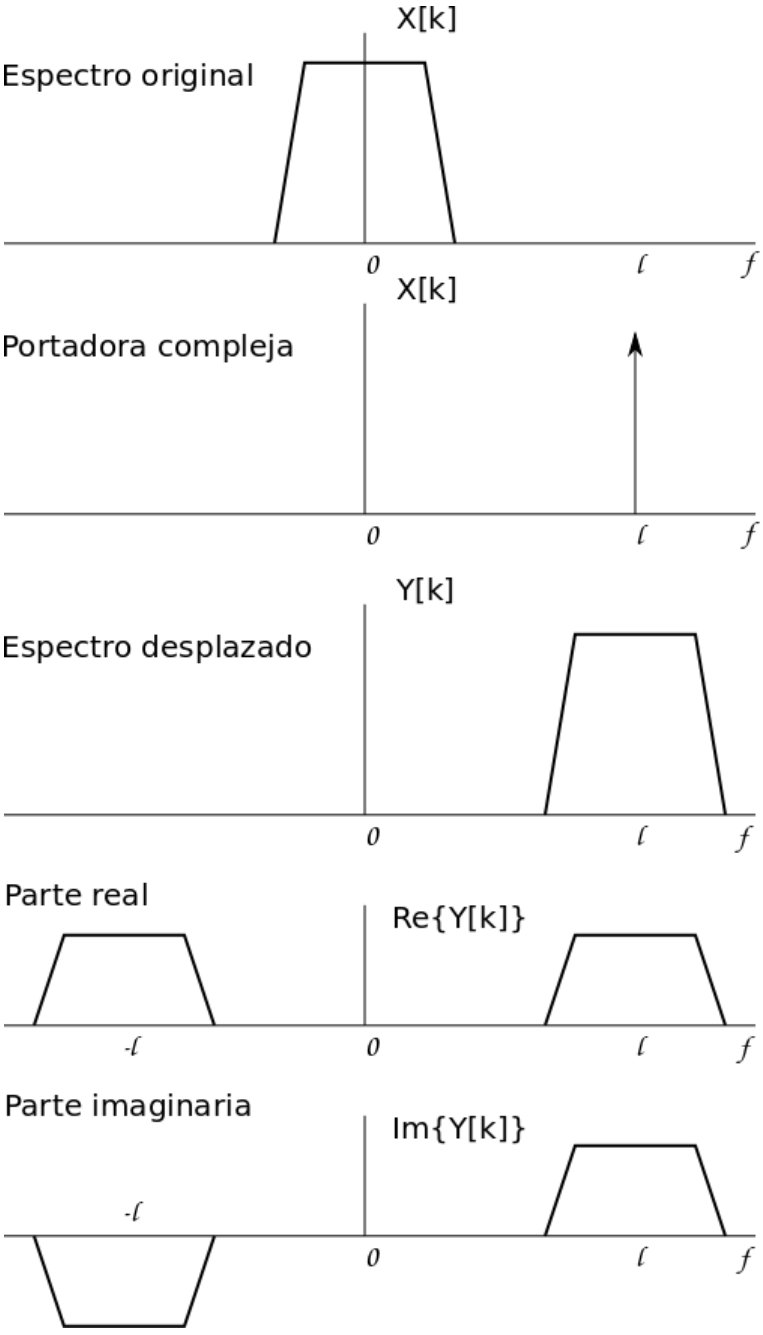
estaremos modulando la amplitud de la portadora con la señal *x[n]*. Para comprender que sucede desde el punto de vista de frecuencia podemos realizar la DFT de la portadora modulada:

$$Y[k] = \sum_{n=0}^{N-1} x[n]e^{\frac{2\pi i}{N}nl}e^{-\frac{2\pi i}{N}kn} \qquad n, k = 0, \ldots, N-1$$

$$Y[k] = \sum_{n=0}^{N-1} x[n]e^{-\frac{2\pi i}{N}(k-l)n}$$

Donde la última ecuación es igual a la DFT de la secuencia *x[n]*, pero desplazada en un valor *l*. Es decir, el centro del espectro de la secuencia *x[n]* se ha trasladado a la frecuencia *l*.

El problema de utilizar una señal una exponencial compleja es que nuestra señal modulada también será compleja, por lo que tendremos que tomar la parte real como salida. Al hacer esto nuestro espectro se volverá simétrico y de la mitad de amplitud que el espectro complejo. Podemos visualizar todo el proceso en la siguiente imagen.



La relación entre l exponencial compleja y la señal real puede entenderse a partir de la relación de euler [1]:

$$e^{ix} = \cos x + i \sin x$$

Por lo tanto, volviendo a nuestra ecuación original y dado que *x[n]* es real, se cumple que

$$\mathfrak{Re}\left(x[n]e^{\frac{2\pi i}{N}ln}\right) = x[n]\cos\left(\frac{2\pi}{N}ln\right)$$

[1] https://es.wikipedia.org/wiki/F%C3%B3rmula_de_Euler

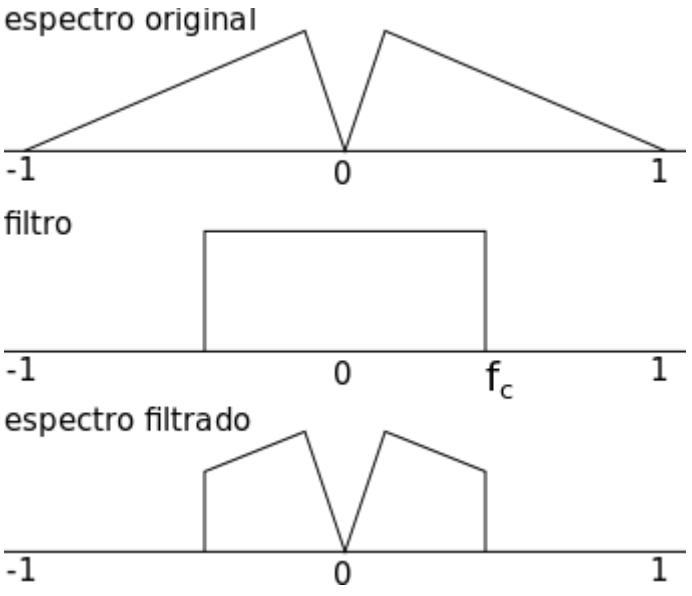
Para una señal de tiempo real (infinita cantidad de muestras), y cambiando *l* por la frecuencia en Hz, podemos expresar la ecuación anterior en términos de la frecuencia de muestreo, de la siguiente manera:

$$y[n] = x[n] \cos\left(\frac{2\pi}{f_s} f n\right)$$

Siendo la anterior la ecuación a aplicar en la modulación de amplitud.

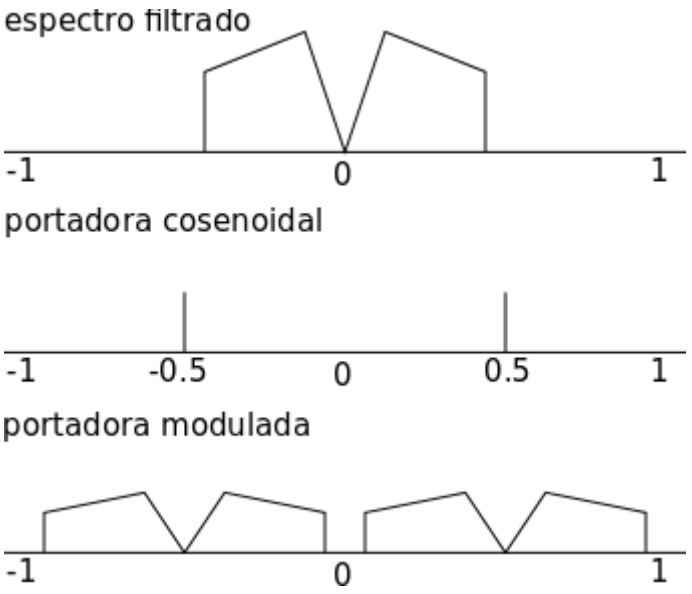
2.1. Aplicar modulación de amplitud para desplazar el espectro del audio contenido en el archivo numeros.wav , de manera que la voz se pueda hacer grave o aguda de acuerdo a un parámetro de desplazamiento en frecuencia, expresado en Hz.
Para lograr esto, se deberán seguir los siguientes pasos

1 - Aplicar un filtro pasabajos para evitar aliasing al modular. Como la señal en cuestión corresponde a la voz, podemos limitar el espectro de la misma y evitar que cualquier ruido de alta frecuencia se introduzca luego como aliasing.
Se deberá elegir la frecuencia de corte del filtro (deberá ser por lo menos menor que la mitad de la frecuencia de muestreo), diseñar el filtro pasabajos y aplicarlo.

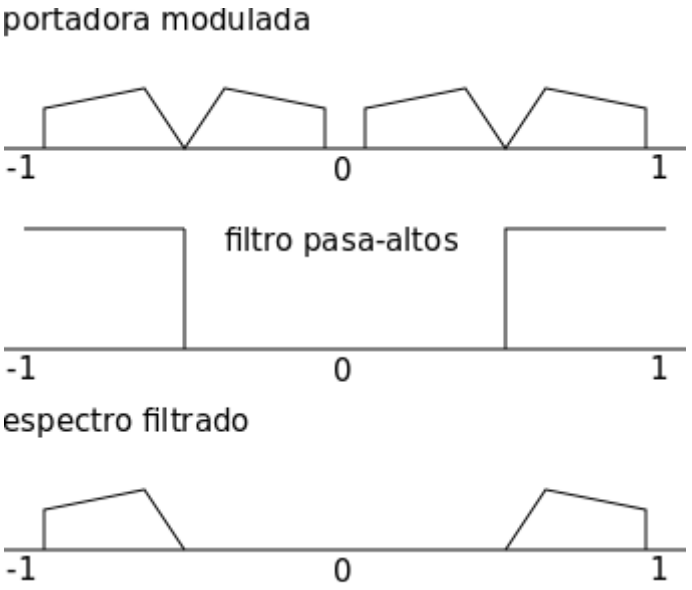


Esta será la señal base con la que iremos a trabajar.

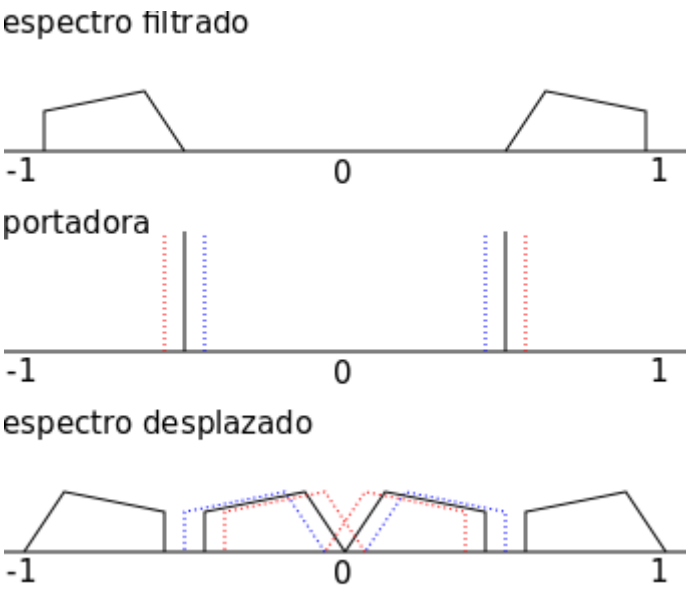
2 - Crear un coseno de frecuencia igual a la mitad de la frecuencia de nyquist, y modularlo con la señal base, como se ve en la siguiente figura



3 - Ahora debemos eliminar la porción correspondiente a las frecuencias negativas, con un filtro pasa-altos, de la siguiente manera:

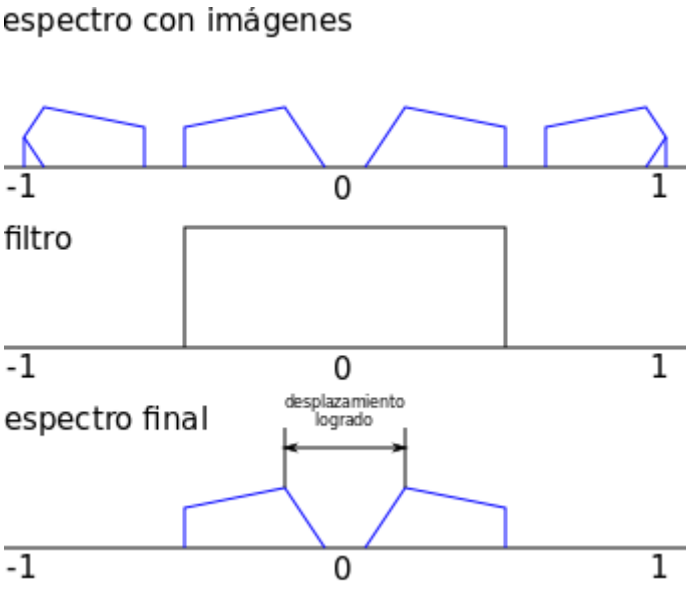


4 - En este punto, si volvemos a modular a la portadora original, el espectro se ubicará en su posición original. Y aquí está el punto central del proceso: Si variamos la frecuencia ligeramente, podremos ubicar el espectro por arriba o por debajo, modificando entonces el tono de voz.



En rojo y en verde se como la variación en la posición de la portadora da una mayor o menor separación entre las frecuencias positivas y negativas, que es en definitiva lo que estamos buscando. También hemos aumentado la amplitud de la portadora para mantener la amplitud del espectro desplazado. Como se ve en la imagen, también quedan componentes de alta frecuencia producto de la modulación que es necesario remover.

5 - El último paso es remover las frecuencias imágenes con otro filtro pasa-bajo, como se ve a continuación:



Por último se deberá reproducir el audio resultante y probar el algoritmo con desplazamientos positivos y negativos

3. Transformada Rápida de Fourier

Ver presentación sobre FFT

3.1. Siguiendo la presentación anterior, crear un programa que implemente el algoritmo de la transformada rápida de Fourier. El programa debe crearse como una función de tipo `y = mi_fft(x)`, el tamaño se obtendrá desde la longitud del vector de entrada y se devolverá un vector de igual longitud con el valor de la DFT.
Además se deberá implementar un programa que llame a la función y compare las salidas con la salida provista por la función `fft` de Matlab/Octave. Se deberán comparar parte real e imaginaria de la salidas con datos aleatorios para secuencias de tamaño 4, 8, 16 y 64.

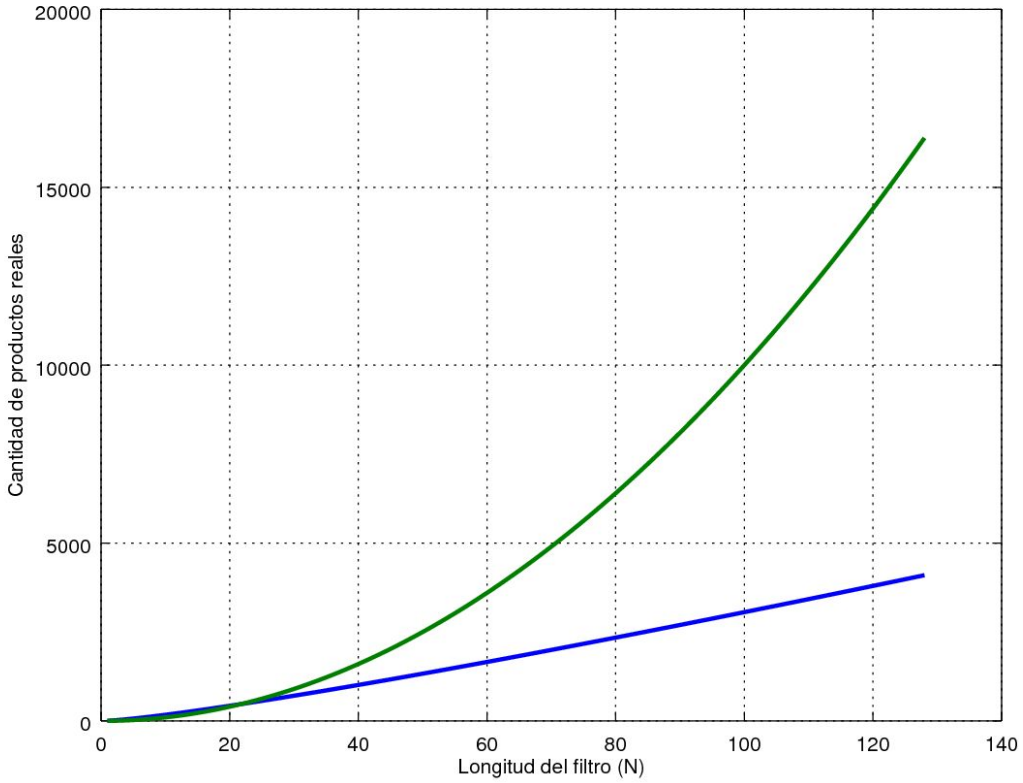
3.2. (Opcional) Repetir el ejercicio anterior pero utilizando únicamente dos bucles FOR. Comparar con utilizando tic y toc diferencia en velocidad de ejecución para valores de N grandes (mayores a 2^12).

4. Filtrado en Frecuencia

Sabemos que una manera de aplicar un filtro FIR es utilizando la convolución entre la secuencia de muestras en el tiempo y la secuencia de coeficientes.
Para un filtro de longitud *M*, se necesitan *M* productos por cada muestra de salida.
Otra manera de aplicar la convolución es hacer uso de la propiedad de la representación en frecuencia que dice que la convolución entre dos secuencias en el dominio del tiempo es igual a la anti transformada del producto de sus respectivas transformadas de Fourier. Matemáticamente

$$x[n] * h[n] = \mathcal{F}^{-1}(X[k]H[k])$$

Suponiendo que ambas secuencias poseen *N* muestras, el costo computacional de la convolución en el tiempo es N^2 multiplicaciones reales, mientras que el costo de hacer la FFT, multiplicar las respuestas (suponiendo que ya disponemos de la respuesta del filtro en frecuencia) y hacer la IFFT es de aproximadamente $4N \log_2 N + 4N$ productos reales.



Podemos ver que con respecto a las multiplicaciones, a partir de filtros mayores a 20 muestras es conveniente realizar las operaciones en el dominio de la frecuencia. La cantidad total de operaciones, como sumas y movimientos de datos va a modificar este número, pero no variará sustancialmente.

Hasta el momento consideramos dos secuencias finitas, de tamaño *M*. Pero que sucede si la secuencia x[n] es una secuencia infinita, como en el caso del procesamiento de señales en tiempo real? Para este caso tenemos a disposición dos métodos:

- Overlap and Save [2]
- Overlap and Add[3]

En ambos casos lo que realiza es tomar bloques de la señal de entrada, convertirlos a frecuencia, multiplicar por la transformada del filtro y aplicar la transformada inversa para volver al tiempo.
Estos bloques deben conectarse entre sí para proveer un filtrado continuo, el *overlap save* lo realiza combinando bloques de entrada y el *overlap add* combinando bloques de salida

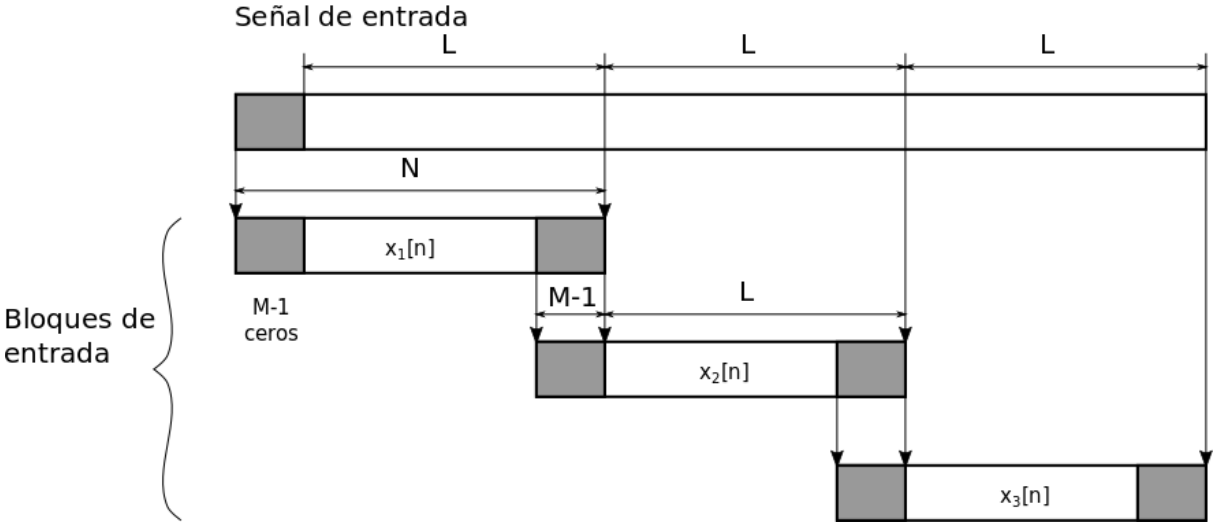
Overlap and Save

[2] https://en.wikipedia.org/wiki/Overlap%E2%80%93save_method
[3] https://en.wikipedia.org/wiki/Overlap%E2%80%93add_method

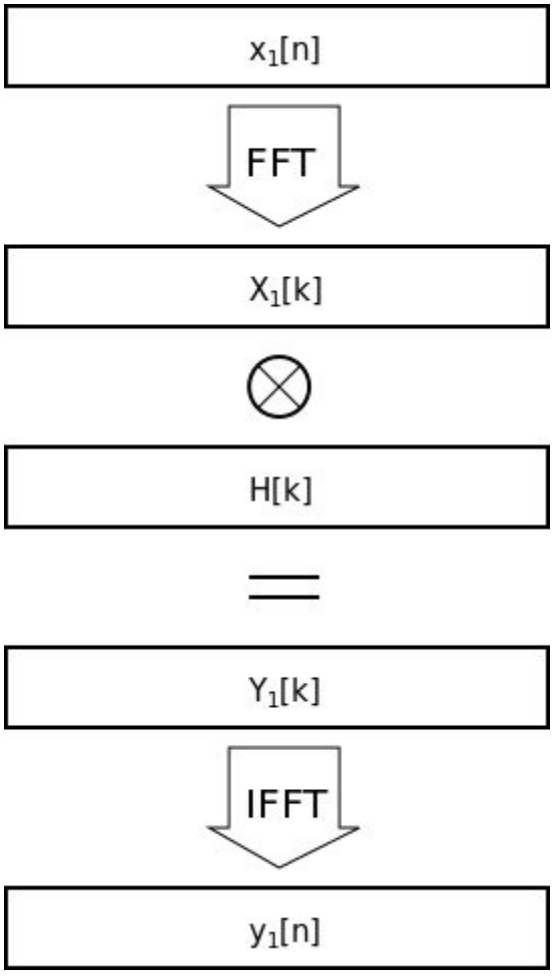
Definiciones
N: Cantidad de muestras de la FFT
M: Longitud de la respuesta al impulso del filtro (coeficientes en el dominio del tiempo)
L: Cantidad de muestras procesadas por bloque

En general se prefiere una relación N/M cercana a 4, lo que maximiza la eficiencia del algoritmo.

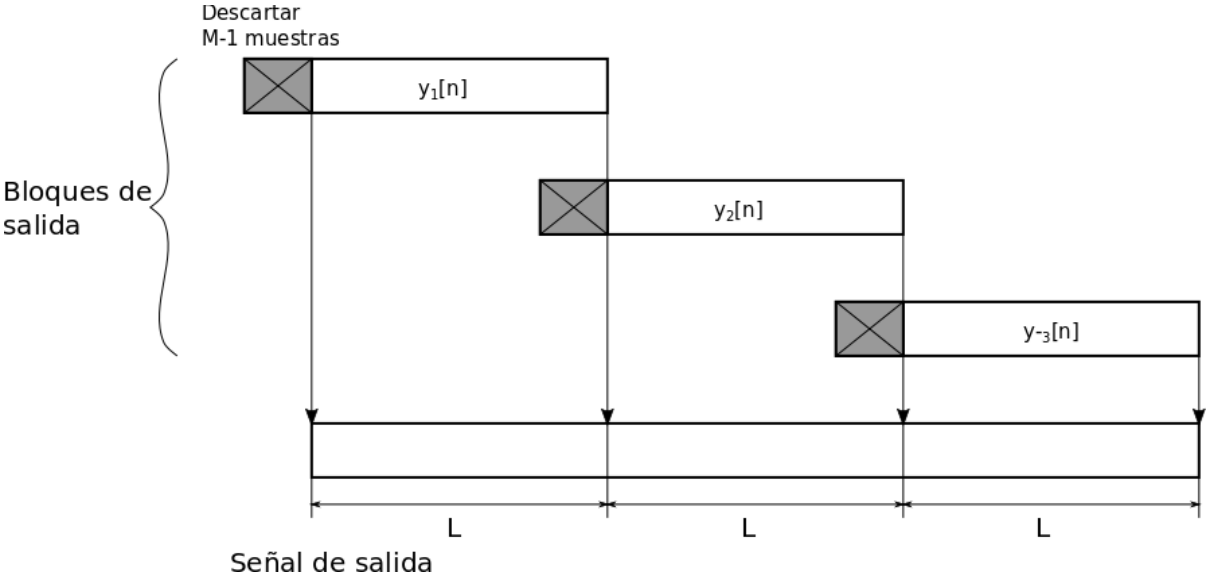
Obtención de bloques de entrada:



Filtrado en frecuencia:



Obtención de salida:



4.1. Crear un programa que permita obtener el resultado de la convolución entre un filtro h[n] y una señal de audio x[n], utilizando el método Overlap & Save.
El filtro a utilizar se obtendrá del archivo s1_r1_b_cd.wav. Este archivo posee dos canales, y se corresponde a la respuesta al impulso de una sala de

conciertos en Finlandia. Pueden verse detalles de la obtención de la respuesta en http://legacy.spa.aalto.fi/projects/poririrs/ . La secuencia de audio queda a elección del alumno, se deberá tener la precaución de operar con la misma frecuencia de muestreo en el filtro y el audio.
Al filtrar, usar el canal izquierdo del filtro para filtrar la señal izquierda y el canal derecho para filtrar la señal de la derecha.
Comparar el tiempo de ejecución del algoritmo con la convolución implementada en la guía 3.

4.2. (Opcional) Implementar y documentar de manera completa el método Overlap-Add y aplicarlo en el ejercicio anterior.

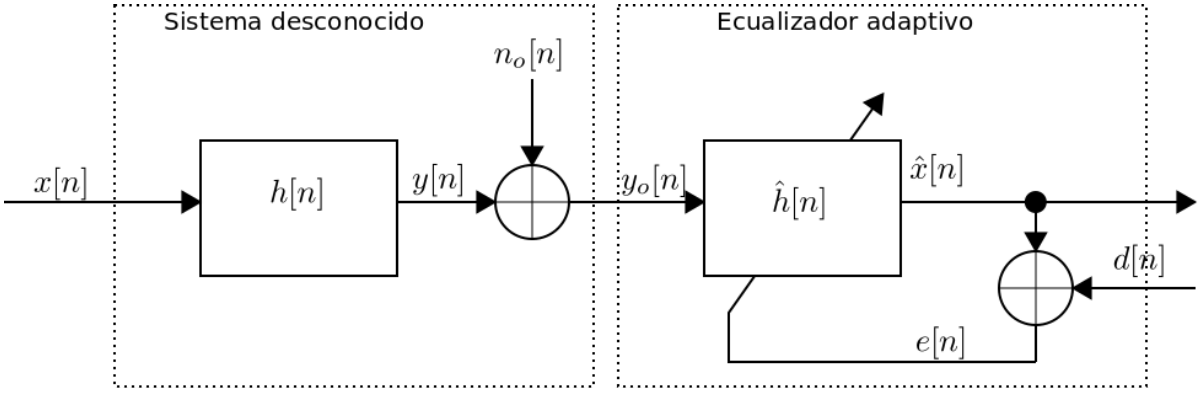
5. Filtrado adaptivo[4]

El filtrado adaptivo se utiliza para:

- La identificación de sistemas
- El filtrado inverso o ecualización
- La remoción de interferencias
- La predicción de sistemas

Nos enfocaremos al uso como ecualizador. En este escenario el problema es el siguiente:

Existe una señal de interés *x[n],* la



cual es afectada por un sistema lineal desconocido *h[n]* y el agregado de ruido $n_o[n]$, generando la señal $y_o[n]$, que será la señal de entrada de nuestro sistema.

Ejemplos que podemos imaginar son la transmisión de una señal de datos que es afectada por el canal de comunicación, o una señal de audio afectada por el eco y la presencia de fuentes de audio adicionales.

[4] https://en.wikipedia.org/wiki/Adaptive_filter

Lo que queremos realizar es obtener un filtro ^*h[n]* que revierta el efecto del filtro *h[n]*. Esta operación de cancelar el efecto de un filtro lineal se denomina *deconvolución*.

Suponiendo que logramos nuestro objetivo, a la salida tendremos una salida ^*x[n]* que consistirá en la señal original afectada solamente por el ruido.

Para poder encontrar el filtro ^*h[n]*, necesitaremos de 2 señales adicionales, una señal de referencia d[n] y una señal de error. La señal de referencia se obtiene a partir de alguna característica de la señal recibida que deseamos mejorar. Por ejemplo, en un sistema de comunicaciones donde los símbolos transmitidos son -1 y 1, la señal de referencia se obtiene de los símbolos generados directamente por la señal ^*x[n]*.
La diferencia entre esta referencia y la señal será nuestra señal de error *e[n]*, que en definitiva es la señal cuya energía buscaremos minimizar.

Para minimizar este error podemos aplicar la técnica LMS (o Least Mean Squared) que minimizará el error cuadrático medio.

El primer paso en nuestro algoritmo consiste en obtener las muestras de ^*x[n]*, lo que logramos convolucionando $y_o[n]$ con ^*h[n]*. ^*h[n]* comienza siendo un filtro pasa todos de tamaño *N*, compuesto por un impulso ubicado al centro del vector de coeficientes *(N-1)/2*.

$$\hat{x}[n] = y_o[n] * \hat{h}[n]$$

El algoritmo opera muestra a muestra, por lo que deberemos implementar la convolución manualmente, pudiendo usar el producto punto entre el vector de coeficientes y el registro de desplazamiento de las muestras de entrada.

$$\hat{x}[n] = \vec{y_o}[n]\vec{\hat{h}}[n]^T$$

donde

$$\vec{y_o}[n] = [y_o[n]\ y_o[n-1]\ \ldots\ y_o[n-N+1]]$$

y

$$\vec{\hat{h}}[n] = [\hat{h}[0]\ \hat{h}[1]\ \ldots\ \hat{h}[N-1]]$$

la señal de error se obtendrá muestra a muestra haciendo

$$e[n] = \hat{x}[n] - d[n]$$

Y el nuevo vector de coeficientes se obtiene mediante la ecuación de actualización

$$\vec{\hat{h}}[n+1] = \vec{\hat{h}}[n] - \mu e[n]\vec{y_o}[n]$$

donde mu es la ganancia de adaptación, y es un escalar.

5.1. Se tiene un sistema de comunicaciones, compuesto por un transmisor, un medio y un receptor. El transmisor recibe bits (unos y ceros) y los convierte a una señal de voltaje entre -1 y 1. Esta señal es enviada al canal de comunicación y del otro lado recibida por el receptor, como se ve en la siguiente figura:

En este sistema el canal de comunicaciones introduce distorsión a través de un efecto denominado interferencia intersímbolo o ISI [5], y es resultado del cambio en la forma de la señal por efecto del filtrado del canal. Este efecto, sumado al ruido que también se introduce genera dificultades al momento de recuperar los datos transmitidos.

Esta es una aplicación ideal para la ecualización adaptativa, puesto que encontrar el filtro que cancele el filtrado del canal permitirá recuperar los símbolos originales.

Para completar el ejercicio, se debe levantar el script lms.m y agregar las ecuaciones de filtrado y de adaptación de coeficientes. Al correr el programa se deberá visualizar la ecualización adaptiva y como se mejora la recepción de las señal, pudiéndose recuperar al final los símbolos correctos.

[5] https://en.wikipedia.org/wiki/Intersymbol_interference