



ugr | Universidad
de **Granada**



GRADO EN INGENIERÍA DE
TECNOLOGÍAS DE TELECOMUNICACIÓN

Cuaderno de Prácticas

Señales Digitales

Departamento de Teoría de la Señal, Telemática y Comunicaciones
Escuela Técnica Superior de Ingenierías Informática y Telecomunicación

El principal propósito de este cuaderno es desarrollar los objetivos y competencias prácticas propuestos en la guía docente de la asignatura Señales Digitales del grado en Ingeniería de Tecnologías de Telecomunicación. Para abordar la tarea se proporciona este Cuaderno de Prácticas como un hilo conductor que permita al alumno desarrollar, mediante trabajo autónomo y durante las sesiones de prácticas, el trabajo planteado.

Índice general

0	Introducción a Matlab™	4
0.1	Introducción	4
0.2	Impulso unitario	4
0.2.1	Tren de impulsos	5
0.3	Señales sinusoidales	6
0.3.1	Generación de señales sinusoidales	6
0.4	Exponenciales y exponenciales complejas	7
0.4.1	Exponencial decreciente	8
0.4.2	Exponenciales complejas	8
1	Teorema del muestreo	10
1.1	Introducción	10
1.1.1	¿Por qué usar señales digitales?	11
1.1.2	¿Que es el muestreo?	11
1.2	Realización guiada	12
1.2.1	Frecuencia Normalizada	13
1.2.2	El aliasing	13
1.2.3	El teorema del muestreo	14
1.2.4	Visión Espectral	14
1.2.5	Experiencia auditiva	15
2	Convolución de Señales Discretas	17
2.1	Introducción	17
2.2	Realización guiada	18
2.2.1	Visualización de la convolución	18
2.2.2	Función de convolución	18
2.2.3	Aplicación: Reverberación por Convolución	19
3	Sistemas Lineales e Invariantes en el Tiempo (LIT)	21
3.1	Introducción	21
3.1.1	Respuesta en frecuencias	22
3.2	Realización guiada	22
4	Estudio de filtros digitales mediante FDATool	26
4.1	Introducción	26
4.2	Realización guiada	26
5	Filtrado Digital	31
5.1	Introducción	31
5.2	Realización guiada	31
5.2.1	La señal de audio	32
5.2.2	Diseño del filtro FIR	33
5.2.3	Diseño del filtro IIR	35

Práctica 0

Introducción a Matlab™

Conoce a tu enemigo y concóctete a ti mismo; en cien batallas, nunca saldrás derrotado.

(Sun Tzu)

Objetivos:

- Familiarizarse con el entorno Matlab™.
- Ser capaz de construir la representación digital de diferentes tipos de señal.
- Visualizar e interpretar señales básicas.

0.1. Introducción

La presente práctica (aproximadamente media sesión) supone una breve introducción al uso de Matlab™ como herramienta básica para las prácticas de Señales Digitales.

Hoy por hoy, Matlab™ es la herramienta más utilizada en ingeniería. Pero, ¿qué es Matlab™? Los más puristas dirán que no es un lenguaje de programación en sí, sino un entorno de programación que incluye un gran número de librerías y funciones para el cálculo. En cualquier caso, Matlab™ y su contrapartida libre, OCTAVE- es y seguirá siendo por algunos años una herramienta óptima para el tratamiento de señales, que es lo que nos va a ocupar durante este viaje.

En esta primera práctica vamos a centrarnos en conocer las diferentes órdenes para crear y visualizar las señales digitales que en futuras prácticas vamos a analizar y procesar.

0.2. Impulso unitario

Como hemos visto en teoría la señal digital más simple es la señal impulso unidad:

$$\delta(n - n_0) = \begin{cases} 1 & \text{si } n = n_0 \\ 0 & \text{si } n \neq n_0 \end{cases} \quad (0.1)$$

Que podemos generar con el siguiente código Matlab™:

```
L = 30;
nn = 0:(L-1);
imp = zeros(L,1);
imp(1) = 1;
stem(nn,imp);
```

El código anterior genera una secuencia, `imp`, de 30 muestras en donde la primera muestra es igual a 1. Nótese como, a través de la definición del vector de tiempos `nn`, podemos representar el pulso en el instante de tiempo 0. La función `stem` visualiza una señal de tiempo discreto. También podemos utilizar la función `plot` para visualizar señales, solo que en este caso todos los puntos aparecen conectados.

Utiliza `stem` y `plot` para visualizar la señal anterior, y comenta las diferencias. ¿Por qué preferimos `stem` en señales discretas?

Si quieres más información sobre una orden de Matlab™, basta con utilizar la instrucción `help`. En este caso, escribiendo la instrucción:

```
> help stem
```

obtendremos la ayuda de la función `stem`, incluyendo sus entradas y salidas. Esto nos será de mucha utilidad más tarde. También podemos utilizar la instrucción `doc`, que muestra la documentación más detallada en una ventana diferente, aunque es más lenta de ejecutar. Estas dos funciones te ayudarán muchísimo en el desarrollo de estas prácticas, por tanto, no las olvides: `help` y `doc` serán tus mejores amigas.

Ahora, genera y representa gráficamente las siguientes secuencias:

$$X_1(n) = 0,9 * \delta(n - 5), \quad 1 \leq n \leq 20. \quad (0.2)$$

$$X_2(n) = 4,5 * \delta(n + 5), \quad -10 \leq n \leq 10. \quad (0.3)$$

0.2.1. Tren de impulsos

Vamos a construir ahora un tren de pulsos. La expresión asociada a un tren con M pulsos y periodo P es la siguiente:

$$S(n) = \sum_{l=0}^{M-1} A * \delta(n - lP), \quad 0 \leq n \leq M * P - 1. \quad (0.4)$$

Este tipo de operaciones con sumatorias siempre se pueden realizar dentro de un bucle `for`, de la siguiente forma:

```
s = zeros(M*P)
for i=1:P:M*P
    s(i)=A
end
```

En este código, recorreremos los elementos de $S(n)$ a intervalos P , con la definición del vector que recorre el bucle: `1:P:M*P`, esto es, va desde 1 hasta $M*P$ a intervalos P . En cada uno

de estos elementos, asignamos la amplitud A . Este código se puede “vectorizar”, una de las grandes ventajas de MatlabTM, permitiendo usar vectores en muchas ocasiones para evitar bucles de este tipo. En nuestro caso, podemos vectorizar la expresión anterior utilizando el vector del bucle para indexar o acceder a esos determinados elementos del vector:

```
s(1:P:M*P)=A
```

una expresión mucho más sencilla, en la que utilizamos indexado para asignar el valor A en las posiciones determinadas.

Con estos datos, ahora vamos a generar y representar un tren de impulsos periódicos cuya amplitud sea $A = 2$, periodo $P = 5$ y longitud $M*P = 50$, comenzando en $n = 0$.

0.3. Señales sinusoidales

Como hemos visto en teoría, podemos describir una señal sinusoidal en el dominio digital de la forma:

$$X(n) = A * \cos(\omega n - \theta) \quad (0.5)$$

donde A es la amplitud, θ la fase y ω la frecuencia (expresada en radianes **por muestra**).

0.3.1. Generación de señales sinusoidales

MatlabTM cuenta con la ventaja de ser un lenguaje vectorial, de forma que las funciones admiten como parametro un vector. La salida no es más que otro vector cuyos elementos son los mismos elementos del vector de entrada, pero aplicando la función indicada. Por ejemplo, si escribimos en MatlabTM:

```
valores=sin([0 pi/4 pi/2 pi 1.5])
```

obtendremos como salida:

```
valores =  
0      0.7071      1.0000      0.0000      0.9975
```

es decir, se calcula el seno de cada uno los elementos del vector `[0 pi/4 pi/2 pi 1.5]` generando el nuevo vector `valores`. Con ello podemos generar una serie de vectores, utilizando el vector de tiempo, definido como en el primer ejercicio.

Ahora, trata de generar y visualizar las siguientes sinusoidales:

- $X_1(n) = \sin(\frac{\pi}{17}n)$, $0 \leq n \leq 75$
- $X_2(n) = \sin(3\pi n + \frac{\pi}{2})$, $-10 \leq n \leq 10$
- $X_3(n) = \sin(\pi n + \frac{\pi}{2})$, $-10 \leq n \leq 10$
- $X_4(n) = \cos(\frac{\pi}{\sqrt{23}}n)$, $0 \leq n \leq 50$

Podemos utilizar la orden `subplot` de MatlabTM para visualizar mejor todas estas señales en su conjunto. La instrucción `subplot(n_filas, n_columnas, posicion)` nos permite situar diferentes gráficas en una misma figura. Para ello, podemos ejecutar:

```
figure();  
subplot(4,1,1);  
stem(nn, x1);
```

```
subplot(4,1,2);
stem(nn, x2);
subplot(4,1,3);
stem(nn, x3);
subplot(4,1,4);
stem(nn, x4);
```

De esta forma podemos visualizar las cuatro señales a la vez en la misma figura, lo que nos permitirá comparar mejor entre ellas.

Calcule ahora las frecuencias f , ciclos por muestra, de cada una de ellas:

$$f_1 = \boxed{}, f_2 = \boxed{}, f_3 = \boxed{}, f_4 = \boxed{}$$

¿Cuales son periodicas y cuales no lo son? ¿A que se debe?

¿Por qué $X_2(n)$ y $X_3(n)$ parecen iguales?

0.4. Exponenciales y exponenciales complejas

Ahora vamos a generar una exponencial del tipo:

$$Y(n) = a^n, \quad n_0 \leq n \leq n_0 + L \quad (0.6)$$

Para ello vamos a definir una función que defina esta operación. En MatlabTM, para poder reutilizar funciones, podemos crearlas en ficheros que tengan el mismo nombre que la función. En este ejemplo, vamos a crear la función `genexp` en el fichero `genexp.m` con el siguiente código:

```
function y = genexp(a,n0,L)
%uso: Y = genexp(A,N0,L)
%A: entrada escalar
%N0: instante de comienzo (entero)
%L: longitud de la senal generada (entero)
%Y: senal de salida Y de longitud L

%Mostramos un error si nos dan una longitud negativa o nula
if (L < 1)
    error(' GENEXP : Longitud no positiva')
end

%Construimos un vector de tiempos desde n0 y longitud L
nn = n0 + [0 : L-1]; %Notese como hago un vector de 0 a L-1
%y luego le sumo a cada termino n0

%Generamos una senal exponencial con base a de longitud L
```

```
%comenzando en n0
y = a.^nn;
end
```

La última orden, en la que aparece “.” es una modificación de la instrucción “elevado a” “^”. Cuando introducimos un punto delante de alguna operación (por ejemplo, “./” o “.*”), estamos aplicando esa operación elemento a elemento.

Empleando esta función, vamos a construir una función exponencial de base 2 definida para $-8 \leq n \leq 8$

0.4.1. Exponencial decreciente

La exponencial decreciente (con $|a| < 1$) es una señal extremadamente útil en el tratamiento digital de señales, como veremos a lo largo de la asignatura. En muchas ocasiones hay que sumar los valores de la secuencia exponencial $a^n * u(n)$. Para un intervalo finito esta suma tiene una expresión compacta:

$$\sum_{n=0}^{L-1} a^n = \frac{1 - a^L}{1 - a} \quad (0.7)$$

Utiliza ahora la función *genexp.m* para generar distintas exponenciales decrecientes (emplee bases a menores que 1, positivas y negativas). Verifica que la fórmula anterior se cumple para tres valores distintos de a y L (intenta utilizar valores distintos a los de sus compañeros).

0.4.2. Exponenciales complejas

Aunque en el mundo real las señales tienen valores reales, en ocasiones es conveniente, para procesar e interpretar dichas señales, transformarlas en señales de valores complejos. Las exponenciales complejas son un caso particular muy importante de señales complejas por su relación con la transformada de Fourier y la representación compacta de armónicos.

Como debemos saber, a través de la fórmula de Euler una exponencial compleja se relaciona con las funciones seno y coseno de la siguiente forma:

$$x(n) = re^{j(\omega n + \theta)} = r(\cos(\omega n + \theta) + j\sin(\omega n + \theta)), \quad -\infty < n < \infty \quad (0.8)$$

Podemos comprobar que esto realmente se cumple a través de las instrucciones `real` e `imag` de Matlab™, que se quedan con la parte real e imaginaria de un número (si tienes dudas, consulta la ayuda). Para ello, recurriremos al siguiente código:

```
%Definimos el intervalo de tiempo [0 50]
n=0:50;
%Construimos la exponencial compleja
x=exp(j*n/3)

%Representamos la parte real y la imaginaria
figure(); subplot(2,1,1); stem(n,real(x)), title('Parte Real')
subplot(2,1,2); stem(n,imag(x)), title('Parte Imaginaria')
```

Al ejecutar esta orden obtendremos dos figuras. Describe que observas en cada una de ellas:

Ahora, vamos a comprender mejor la utilidad de estas señales complejas. Para ello, vamos a modificar el código anterior para obtener una exponencial compleja con $r = 3$, $\omega = \pi/10$ y $\theta = 0$ en el intervalo $0 \leq n \leq 20$. Posteriormente, vamos a representar la parte real en función de la parte imaginaria de cada uno de los puntos de esta exponencial. Podemos hacerlo como si fuera una animación utilizando la orden `pause`, que para la ejecución de un bucle hasta que pulsemos una tecla del teclado. Para hacerlo, simplemente completemos este código:

```
% COMPLETAR: construccion de la exponencial compleja.
figure();
axis equal % este comando establece la misma proporcion en todos los
    ejes
for i = 1:20
    % COMPLETAR: muestra el i-esimo punto
    hold on; % permite representar mas puntos en los mismos ejes
    pause()
end
```

Una vez que lo hayamos representado, piensa y responde a las siguientes preguntas. **¿A que forma geometrica se asemeja la figura que obtiene?**

¿A que distancia de la coordenada (0,0) están todos los puntos?

Si considerásemos una representación polar de los puntos (radio y angulo en el plano complejo) ¿que relación hay entre ω y los angulos de los puntos?

Para terminar, hagamos un truco gracioso:

```
> why
```

Prueba a repetir varias veces esta última instrucción, coméntala con tus compañeros, y habremos terminado esta práctica.

Práctica 1

Teorema del muestreo

Sírvete de lo aparente como indicio de lo inaparente.

(Solón de Atenas)

Nota: para esta práctica necesitarás unos auriculares.

Objetivos:

- Descubrir las diferencias entre señales analógicas y digitales.
- Comprender el teorema de muestreo y sus implicaciones en el dominio del tiempo y de la frecuencia.
- Experimentar el muestreo y fenómenos como el *aliasing*.

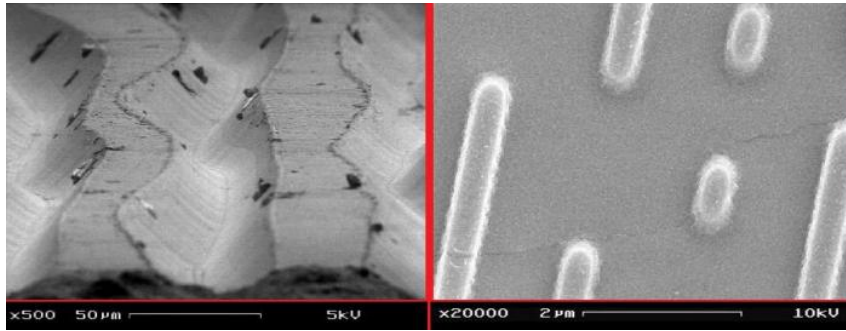
1.1. Introducción

En esta práctica (1,5 sesiones) vamos a estudiar el teorema de muestreo y sus implicaciones en el dominio del tiempo y de la frecuencia, analizando fenómenos como el *aliasing*.

Comencemos por una pregunta para reflexionar sobre algo que probablemente todos conocemos:

¿Qué diferencia un disco de vinilo de un CD?

Una pista, en la siguiente imagen de microscopio:



Con las diferencias que has encontrado intenta dar una respuesta a las preguntas: **¿Cómo definirías una señal digital? Encuentra al menos dos características que diferencian las señales analógicas de las digitales.**

1.1.1. ¿Por qué usar señales digitales?

Vamos a ponernos un poco trascendentes. Hoy día casi todo el procesamiento de señales, ya sea sonido, imagen u ondas cerebrales, se realiza tras una digitalización de las mismas. Podría parecer que añadir un paso más a un proceso lo haría mucho más complicado, y sin embargo es la opción preferida. Como cualquier cambio, éste obedece a una serie de razones. **Piensa y enumera algunas razones que justifiquen el uso de señales digitales frente a las analógicas.**

1.1.2. ¿Que es el muestreo?

El primer paso en la digitalización de una señal es el proceso de **muestreo**. Éste consiste en tomar muestras de una señal analógica a una frecuencia o tasa de muestreo constante F_s , para cuantificarlas posteriormente. En la Figura 1.1 se ilustra el proceso, mostrando una señal analógica, un tren de impulsos y finalmente, la señal muestreada.

Con la ayuda de la Figura 1.1 intenta llegar a una definición más “matemática” del muestreo, utilizando la fórmula del tren de impulsos con un periodo de muestreo T_s . Pista: un tren de pulsos de amplitud unidad.

$$\sum_{k=-\infty}^{\infty} \delta(t - kT_s)$$

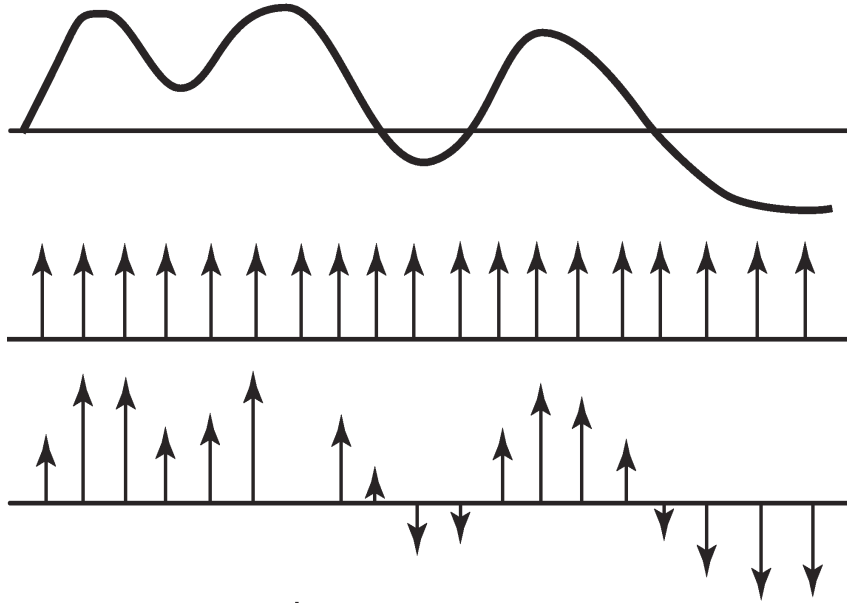


Figura 1.1: Operación de muestreo de una señal analógica.

Mediante una simple operación has obtenido una representación discreta de una señal analógica, que luego se puede cuantizar –esto es, codificar la amplitud de cada impulso utilizando una secuencia de n bits– y almacenar en nuestro disco duro. Como puedes ver hemos tirado a la basura una cantidad infinita de señal (todo lo que hay entre dos muestras), ¿qué implicaciones tiene esto? Lo veremos en el siguiente apartado.

1.2. Realización guiada

En esta sección se van a realizar diferentes pasos sobre el papel y en el ordenador –con MatlabTM– para replicar el proceso de muestreo en el ordenador. También analizaremos los resultados que obtendremos para llegar a un conocimiento más profundo del muestreo.

En primer lugar, definimos la señal $s(t) = A \cos(2\pi Ft + \theta)$. Con los valores $A = 1$, $F = 3$ y $\theta = 0$, se pretende realizar el muestreo con un periodo T_s . El proceso de muestreo será el siguiente:

$$s(n) = s(t)|_{t=nT} = A \cos(2\pi FnT_s + \theta) = A \cos(2\pi fn + \theta) \quad (1.1)$$

Ahora te toca simular el proceso en el ordenador. Para ello, debes seguir los siguientes pasos:

1. Definir dos vectores en MatlabTM, t_1 y t_2 , que contendrán los valores de tiempo t en los que se quiere muestrear, desde 0 hasta 2 segundos, en intervalos $T_{s1} = 1ms$ y $T_{s2} = 20ms$.

2. Evaluar la función $s(t)$ –no confundir con la $s(n)$ – con los valores dados y guardarlos en las variables x1 y x2.
3. Representar la función x1 con el comando plot y x2 con stem en la misma ventana. Para hacerlo, basta con escribir en la ventana de comandos:

```
plot(t1,x1,'r')
hold on
stem(t2,x2,'b')
hold off
```

Con esta estrategia habrás simulado una función continua x1 y su muestreo x2 en la misma ventana. Observa el resultado porque es un paso que tiene muchas implicaciones.

1.2.1. Frecuencia Normalizada

La primera implicación del muestreo es que en la ecuación 1.1 aparece un valor f . Dicho valor $f = F/F_s$ se llama **frecuencia normalizada**, y tiene un rango $-1/2 < f < 1/2$ (motivado porque la frecuencia angular normalizada está en el rango $-\pi < \omega < \pi$).

Para practicar, calcula la frecuencia normalizada de dos señales $s_1(n)$ y $s_2(n)$, con frecuencias $F_1 = 2kHz$ y $F_2 = 3kHz$, a una tasa de muestreo $F_s = 6kHz$.

$$f_1 = \boxed{}, f_2 = \boxed{}$$

Ahora repetimos el cálculo con una tasa de muestreo $F_s = 4kHz$.

$$f_1 = \boxed{}, f_2 = \boxed{}$$

¿Qué ocurre con la frecuencia normalizada? ¿Hay algo que no resulte intuitivo? ¿Hay alguna que esté fuera del rango $-1/2 < f < 1/2$?

1.2.2. El aliasing

Cuando, como en el caso anterior, obtenemos una frecuencia normalizada que está fuera del rango $-1/2 < f < 1/2$, se produce un efecto no deseado: el **aliasing**. Para ayudar a comprender el aliasing, vas a visualizar este efecto en Matlab™. Para ello, sólo tienes que realizar los pasos 1-3 anteriores, creando las señales x3, x4 y x5 muestreando a diferentes tasas: $F_{s3} = 10Hz$, $F_{s4} = 6Hz$ y $F_{s5} = 3,5Hz$.

¿Que puedes observar en las representaciones discretas de estas nuevas señales? Vamos ahora a representar la señal x5 sin utilizar x1 como referencia. **¿Puedes identificar la señal original de la cual se deriva?**

El caso de la señal x5 es precisamente un ejemplo de aliasing. La señal que visualmente has reconstruido tiene una frecuencia mucho menor que la original. Esto ocurre así porque la fórmula que se utiliza para el cálculo de la frecuencia normalizada no es del todo correcta. Cada frecuencia real produce infinitas frecuencias imágenes al muestrear, siguiendo la fórmula:

$$\omega = \pm 2\pi \left(n + \frac{F}{F_s} \right) \quad \text{ó} \quad f = \pm \left(n + \frac{F}{F_s} \right) \quad (1.2)$$

Así pues, la frecuencia $F = 3kHz$ del apartado 1.2.1, muestreada a $F_s = 4kHz$ daría las frecuencias imágenes $-1,25, -0,75, -0,25, 0,25, 0,75, 1,25, 1,75$, etc. Y de ese modo, la frecuencia que finalmente vemos (u oímos), es la frecuencia que está dentro del rango al reconstruir, en nuestro caso $f = \pm 0,25$, y la oíríamos entonces como una frecuencia real de $F = f \times F_s = 0,25 \times 4kHz = 1kHz$, en lugar de los $3kHz$ originales.

1.2.3. El teorema del muestreo

La necesidad de reconstruir una señal de forma unívoca, y evitar este tipo de problemas es la que motiva el **teorema de muestreo de Nyquist-Shannon**. Es un teorema fundamental en la teoría de la información, particularmente en telecomunicaciones, y dice así:

Si una señal continua, $S(t)$, tiene una banda de frecuencia tal que f_m sea la mayor frecuencia comprendida dentro de dicha banda, dicha señal podrá reconstruirse sin distorsión a partir de muestras de la señal tomadas a una frecuencia f_s siendo $f_s > 2f_m$.

Piensa en el caso alternativo: ¿qué hay que hacer con la señal original si queremos muestrear a una frecuencia F_s dada?

No respetar el teorema del muestreo cuando se captura una señal en el tiempo da lugar a señales digitales que no representan la realidad de forma correcta. Un divertido ejemplo lo tienes en <https://www.youtube.com/watch?v=R-IVw80KjvQ> (y su explicación en <https://www.youtube.com/watch?v=AYQAKwCxScc>).

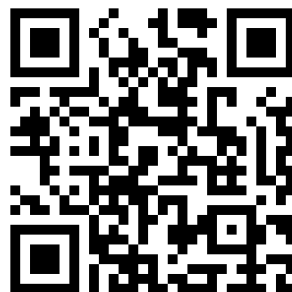


Figura 1.2: Escanea este código QR para ir directo al enlace.

Ahora, **indica cuales de los periodos y frecuencias de muestreo utilizadas al inicio ($T_{s2} = 20ms$) y en el apartado 1.2.2 ($F_{s3} = 10Hz$, $F_{s4} = 6Hz$ y $F_{s5} = 3,5Hz$) respetan el teorema de muestreo, y por qué.**

1.2.4. Visión Espectral

¿Te acuerdas de Fourier?



Figura 1.3: Este señor es Fourier.

Sí, ese señor con cara de tan buena persona es Jean-Baptiste Joseph Fourier, matemático y físico francés al que le debemos la **Transformada de Fourier**, o lo que es lo mismo, una descomposición de funciones periódicas utilizando series trigonométricas. Como recordaras, llamamos **espectro** a una representación en escala de frecuencias de la amplitud de cada componente frecuencial (o espectral), que debidamente sumadas, nos permitirían reconstruir la señal original.

Mas adelante estudiaremos en detalle la transformada de Fourier, su aplicación a señales en tiempo discreto y su version discreta. Por el momento, asumiremos que la funciones `fft` y `abs`, nos permiten obtener el espectro y su módulo, respectivamente, de una señal muestreada. Es importante recordar que esto no es del todo correcto, de hecho, para visualizar el módulo del espectro tendremos que emplear la siguiente secuencia de instrucciones:

```
espectro = abs(fft(x, 1024))
Fs = 1/Ts
frecuencias = 0:Fs2/(length(espectro)-1):Fs
plot(frecuencias, espectro)
```

Por el momento no entraremos en qué estamos realmente haciendo con esas instrucciones y simplemente asumiremos que van a calcular numéricamente el espectro de la señal x desde la frecuencia 0 a F_s .

Comprueba a qué frecuencias aparecen las deltas (recuerda el espectro de una señal coseno) en el espectro de las señales x_3 , x_4 , x_5 . ¿Qué tiene de particular el caso de x_5 ?

1.2.5. Experiencia auditiva

Es posible que ya tengas una intuición sobre las implicaciones del muestreo, así como los conceptos teóricos del teorema del muestreo afianzados. Antes de terminar, vas a experimentar estas propiedades de forma sensorial, acercándonos al mundo cotidiano: con el oído.

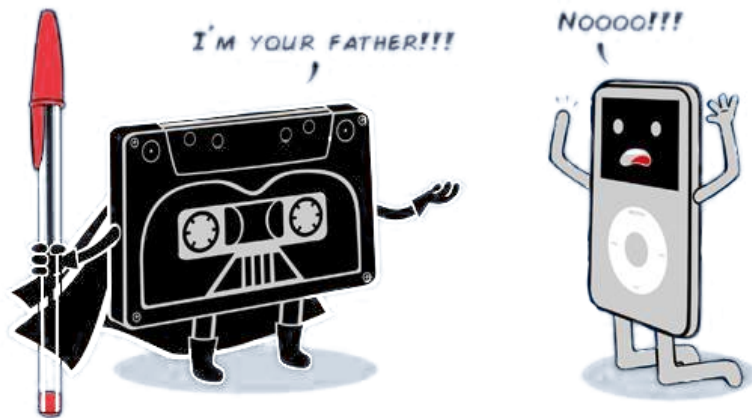
Probablemente sabrás que el oído humano es capaz de oír entre $20Hz$ y $20kHz$. Durante todos estos años, se han establecido unos estándares de calidad para diferentes medios en los cuales varía la frecuencia de muestreo. Tenemos por ejemplo, la calidad CD ($F_s =$

44100Hz), cinta de Casette ($F_s = 22050Hz$), radio de baja calidad ($F_s = 11025Hz$) o línea telefónica ($F_s = 8000Hz$).

Si queremos escuchar una señal coseno con frecuencia $F = 7000Hz$, **de entre los anteriores, ¿cuáles son los estándares que respetan la frecuencia de Nyquist-Shannon?**

Reproduce ahora cada una de las señales x2 a x5, muestreadas a diferentes valores F_s , con el comando `soundsc(señal,Fs)`. **¿Se oye lo mismo en todos los casos?**

De acuerdo con lo visto anteriormente, **¿a qué frecuencia se situa el tono en cada uno de los esquemas de calidad?**



Práctica 2

Convolución de Señales Discretas

Nota: para esta práctica necesitarás unos auriculares.

Objetivos:

- Comprender y visualizar el concepto de convolución.
- Realizar la programación de la función estudiada.
- Desarrollar un método de reverberación por convolución.

2.1. Introducción

En esta práctica (1 sesión) vamos a ver una operación fundamental en el procesamiento de señales: la convolución. Aunque en este momento pueda parecer una operación algo exótica, más adelante veremos su enorme importancia. La convolución entre dos señales habitualmente se representa de la forma:

$$y(n) = x(n) * h(n) \quad (2.1)$$

En el dominio discreto de las señales digitales esto es equivalente a:

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) \quad (2.2)$$

Esta sumatoria se conoce como *suma de convolución*. A diferencia de su homóloga en tiempo continuo, la convolución de señales discretas finitas es fácilmente implementable, ya que no requiere de resolver una integral. La convolución se utiliza ampliamente en el procesamiento de señales, entre otras cosas para filtrar una señal $x(n)$ utilizando la respuesta al impulso de un filtro $h(n)$ (lo veremos más adelante en teoría).

2.2. Realización guiada

2.2.1. Visualización de la convolución

En primer lugar, vamos a utilizar el script `visualiza_conv.m` que se proporciona, para visualizar cómo se realiza la operación de convolución. Para ello, escribimos el siguiente código:

```
nn = -5:5;
cuadrada = zeros(length(nn),1);
cuadrada(5:10) = ones(6,1);
triangular = zeros(length(nn),1);
triangular(1:5)=1:5;
triangular(6:9)=4:-1:1;
visualiza_conv(cuadrada, triangular, nn, None);
```

El código genera una señal cuadrada y una triangular, y las convoluciona (la visualización avanza un paso cada vez que pulsemos una tecla). **¿Dónde está el máximo de la convolución?**

¿Qué ocurre con las funciones cuando se produce este máximo? ¿Qué implicaciones puede tener?

2.2.2. Función de convolución

Ahora, para tener una mejor comprensión de la convolución, vamos a implementarla en una función de Matlab™. Como ya vimos en la práctica 0, una función en Matlab™ se declara con la cabecera y el pie siguientes:

```
function y = convolucion(h,x)
    % Texto de la funcion
end
```

En este caso, el archivo se llamara `convolucion.m`, y para poder utilizarlo debemos estar en la misma carpeta en la que se contiene. Ya sabemos que la función de convolución tiene la forma que vimos en la ecuación 2.2. Ahora, piensa en cómo podemos traducir esta fórmula a una secuencia de comandos en Matlab™.

Una pista: las sumatorias suelen poder escribirse como un bucle for. **¿Qué variables tenemos en la fórmula? ¿Cuántos bucles for necesitaremos entonces para implementar la convolucion?**



Figura 2.1: Iglesia de Schellingwoude (Alemania), donde se grabó la señal `Iglesia.wav`.

¿Qué longitud tendrá el vector final?

Con estos datos, ya estás listo para empezar a programar. Es posible que necesites crear una matriz auxiliar y luego realizar una operación final. **Programa ahora la función de convolución.**

2.2.3. Aplicación: Reverberación por Convolución

La convolución se utiliza muy frecuentemente en procesamiento de audio, tanto para filtrado como para otro tipo de efectos, por ejemplo la *reverberación*. La reverberación es un proceso que se produce cuando escuchamos una señal original, seca, reflejada en numerosas paredes, cuyas reflexiones nos llegan en tiempos diferentes, y normalmente menores de 100 ms.

Como hemos visto antes en la operación de convolución, podemos “transformar” una señal original utilizando otra. De ese modo podríamos utilizar una señal que represente la “reverberación” (más adelante en teoría hablaremos sobre esto) de una sala, para dar forma a la señal original, mediante la convolución de ambas.

Para ello, **cargamos las señales que queremos utilizar mediante la función** `audioread`:

```
[senal, Fs] = audioread('archivo.wav');
```

Mediante esta función obtenemos un vector `senal`, que contiene la señal $x(n)$ y la frecuencia de muestreo a la que está grabada, `Fs`. Vamos a cargar la señal original `freddy.wav`, y las señales que representan la reverberación `Hall.wav` e `Iglesia.wav`. **Ahora, escucha cada señal por separado utilizando la función** `soundsc`:

```
soundsc(senal, Fs)
```

Ahora, podemos meter a Freddie Mercury en cada ambiente mediante la convolución de la señal original con los diferentes archivos de reverberación que tenemos:

```
y1 = conv(senal, hall);  
y2 = conv(senal, iglesia);
```

Y los volvemos a reproducir, utilizando el comando `soundsc`. **¿Qué diferencias encuentras entre la señal original y las señales procesadas?**

En el mundo real no escuchamos solo la señal reverberante, sino una mezcla de la señal directa del que habla y la reverberación. Para hacer esto, simplemente normalizamos las señales al valor máximo:

```
ynormalizada = y/max(y)
```

Y posteriormente las sumamos y las escuchamos. Recuerda que el tamaño de los vectores es diferente, por lo que para poder sumar tenemos que igualar su longitud.

Práctica 3

Sistemas Lineales e Invariantes en el Tiempo (LIT)

Objetivos:

- Estudiar los sistemas LIT a partir de su transformada Z.
- Estudio de los parámetros fundamentales de un sistema LIT: función de transferencia, respuesta en frecuencia, y respuestas al impulso y al escalón.

3.1. Introducción

En esta práctica (2 sesiones) procederemos al estudio de sistemas lineales e invariantes en el tiempo (LIT, LTI por sus siglas en inglés) por medio del uso de la transformada Z para describir dichos sistemas y la obtención de la función del sistema $H(z)$, y su particularización a la circunferencia de radio unidad para obtener la respuesta en frecuencia $H(\omega)$.

Como hemos visto en clase, los sistemas lineales de coeficientes constantes se pueden describir en el dominio del tiempo por su ecuación en diferencias:

$$\sum_{k=0}^N a_k y[n-k] = \sum_{l=0}^M b_l x[n-l]; a_0 = 1$$

Respuesta al impulso

Alternativamente, el sistema en el dominio del tiempo también queda descrito por su respuesta al impulso:

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

3.1.1. Respuesta en frecuencias

Si se analiza el sistema en el dominio transformado, por medio de la transformada Z a la ecuación en diferencias, se obtiene la función de transferencia del sistema $H(z)$:

$$H(z) = \frac{\sum_{l=0}^M b_l z^{-l}}{\sum_{k=0}^N a_k z^{-k}}$$

Dicha función de transferencia queda caracterizada por sus polos y ceros.

Particularizando $H(z)$ en la circunferencia de radio unidad, $z = e^{j\omega}$, se obtiene la respuesta en frecuencia del sistema:

$$H(\omega) = \frac{\sum_{l=0}^M b_l e^{-jl\omega}}{\sum_{k=0}^N a_k e^{-jk\omega}}$$

donde $H(\omega)$ es una función periódica cada 2π .

3.2. Realización guiada

Para comenzar el estudio de sistemas LIT mediante MatlabTM, vamos a estudiar el sistema descrito por la siguiente ecuación en diferencias:

$$y(n) = 1,52y(n-1) - 0,8y(n-2) + x(n) + 1,2x(n-1) + x(n-2)$$

> Crea los vectores b y a que contengan respectivamente los coeficientes de $x(n)$ y de $y(n)$.

Cálculo de la respuesta al impulso

En el caso de su cálculo mediante MatlabTM, los valores concretos de $h(n)$ para $n = 0, 1, 2, \dots$ se obtienen al introducir $\delta(n)$ como señal de entrada al filtro, descrito a partir de los coeficientes a_k y b_l . Para ello se emplea la función `filter(b,a,x)` de MatlabTM. Dicha función permite aplicar un filtro digital, caracterizado por los vectores de coeficientes a y b , a una señal de entrada $x(n)$ dada. Si la señal $x(n)$ es $\delta(n)$, entonces $y(n)$ será la respuesta al impulso $h(n)$. Hay que tener en cuenta que la función `filter()` devuelve tantas muestras en y como muestras haya en x , es decir, la respuesta al impulso se trunca a la longitud del vector x .

> Crea un vector impulso unidad de longitud 128. Genera los 128 primeros puntos de la respuesta al impulso del filtro mediante `filter()`. Representa gráficamente los valores como una señal de tiempo discreto. **¿Qué ocurre?**



Cálculo de la respuesta al escalón

De igual modo se puede estudiar la respuesta al escalón del filtro. Dado que el escalón equivale a una señal en continua, la respuesta permite intuir la naturaleza del filtro (paso bajo o paso alto).

> Crea un vector escalón de longitud 128. Genera la respuesta al escalón del filtro. Representa gráficamente los valores como una señal de tiempo discreto. **¿Qué le ocurre al escalón inicial? ¿Qué valor tiene la señal a la salida cuando alcanza un valor constante (régimen permanente)?**



La parte variable de la respuesta hasta que se alcanza el régimen permanente se denomina respuesta transitoria.

> Visualiza la respuesta transitoria $y_{trans} = y(n) - y_{perm}$ y determina la longitud del transitorio. **¿Por qué ocurre esto?**



Cálculo de la función de transferencia (polos y ceros)

Una manera muy sencilla de visualizar los polos y ceros a partir de la función de transferencia (coeficientes) es mediante la función `zplane()`. Además de la visualización, se pueden calcular los valores numéricos de dichos polos y ceros mediante la función `tf2zp()`. Alternativamente, la función `zp2tf()` permite determinar la función de transferencia a partir de los polos y ceros del sistema.

> Determina los polos y los ceros de la función de transferencia y represéntalos gráficamente. **¿Es estable el sistema?**



Cálculo de la respuesta en frecuencia

En el caso de su cálculo mediante MatlabTM, se emplea la función `freqz()` para hallar la respuesta en frecuencia, ya que la evaluación de la transformada Z en la circunferencia de radio unidad es equivalente a la transformada de Fourier de tiempo discreto. La sentencia `[H,W] = freqz(b,a,N,'WHOLE')` proporciona la respuesta en frecuencia de un filtro para N frecuencias equidistantes (en H se devuelven N muestras de la respuesta en frecuencia, y en W se devuelven los N valores equidistantes de ω). Si no se usa la opción `'WHOLE'`, sólo se evalúa la mitad superior de la circunferencia de radio unidad (desde la frecuencia 0 a π) que, como sabemos, es suficiente para caracterizar filtros con coeficientes reales.

> Calcula la respuesta en frecuencia del sistema y represente su módulo y fase. **¿Cuáles son las características del filtro? ¿De qué tipo es?**

Cálculo para otros sistemas

Repita los cálculos de los apartados anteriores con las siguientes ecuaciones en diferencias:

1. $y(n) + 1,02y(n-1) + 0,64y(n-2) = 0,64x(n) + 1,02x(n-1) + x(n-2)$

2. $y(n) - 0,66y(n-1) + 0,8y(n-2) = x(n) - 0,74x(n-1) + x(n-2)$

> **¿A qué tipos de filtros representan dichos sistemas?**

Cálculo mediante la resolución de la ecuación en diferencias

Alternativamente, para la obtención de la respuesta al impulso del sistema, se puede proceder a resolver la ecuación en diferencias. La función `residuez()` permite obtener la respuesta al impulso $h(n)$ por medio del método de los residuos.

Tome el último de los sistemas anteriores:

$$y(n) - 0,66y(n-1) + 0,8y(n-2) = x(n) - 0,74x(n-1) + x(n-2)$$

> Calcula el valor de la respuesta al impulso por medio de la solución de la ecuación en diferencias. Visualízala conjuntamente con el valor calculado anteriormente. ¿Existen diferencias?

Práctica 4

Estudio de filtros digitales mediante FDATool

Objetivos:

- Aprender el manejo de la herramienta FDATool para el diseño y estudio de filtros digitales.
- Estudiar todos los parámetros relevantes de un filtro digital: polos, ceros, respuesta al impulso, respuesta en frecuencia, etc.

4.1. Introducción

En esta práctica (1 sesión) procederemos al estudio de filtros digitales mediante FDATool. FDATool es una interfaz gráfica que ayuda al diseño y caracterización de filtros digitales, permitiendo estudiar gran variedad de filtros digitales de modo sencillo, a partir de sus parámetros fundamentales de diseño, como son: banda de paso, frecuencia de corte, rizado de la banda de paso, atenuación de la banda de rechazo, etc. La imagen Fig.4.1 muestra el aspecto visual de FDATool.

La herramienta ejecuta internamente las funciones y rutinas de diseño de filtros de MatlabTM que se encuentran en la librería de procesamiento de señal, permitiendo simplificar al usuario la complejidad que todas estas funciones puedan conllevar. En nuestro caso, nos servirá para afianzar múltiples conceptos sobre sistemas LIT y filtrado digital de modo visual, dinámico y sencillo.

4.2. Realización guiada

Para la realización de la práctica, se va a proceder a estudiar distintos tipos de filtros digitales a partir de sus parámetros de diseño. El estudio de dichos filtros permitirá obtener las respuestas en frecuencia, respuestas en fase, diagramas de polos/ceros, respuestas al impulso, al escalón, etc.

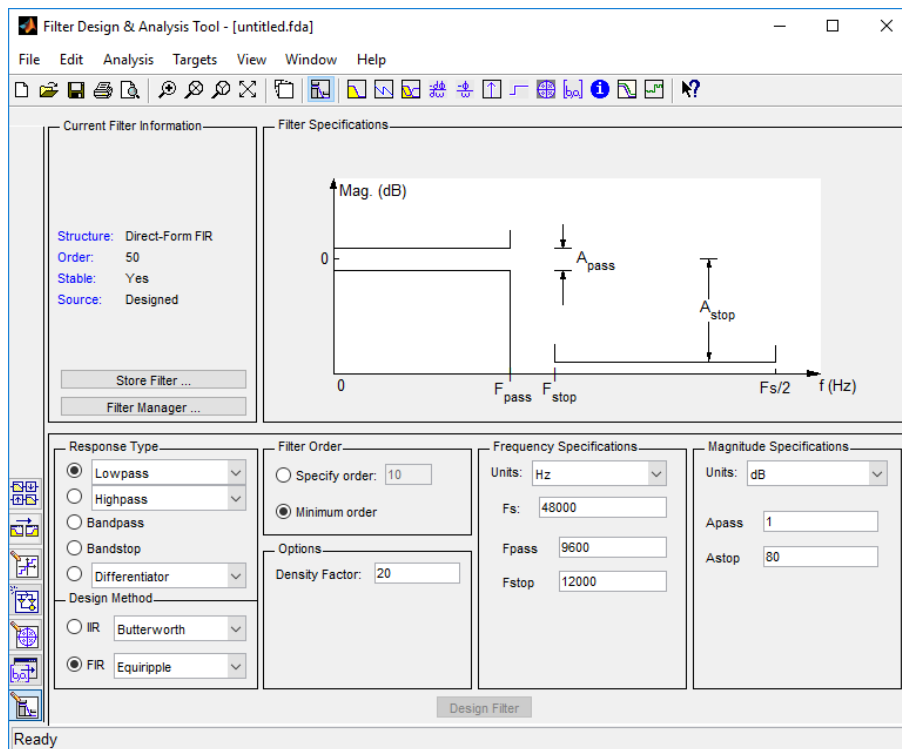


Figura 4.1: Aspecto visual de FDATool.

Inicialmente nos planteamos el diseño de un filtro paso bajo que cumpla la siguiente máscara para su respuesta en frecuencia:

- Frecuencia de Muestreo: 10 KHz
- Máxima atenuación en la banda de paso: 1.5 dB
- Mínima atenuación en la banda atenuada: 40 dB
- Frecuencia de corte de la banda de paso: 2.4 KHz
- Frecuencia de corte de la banda atenuada: 3 KHz

> **Ejecuta FDATool** escribiendo `fdatool` en la ventana de ordenes. Establece todos los parámetros de diseño del filtro anterior y calcula el filtro para los tres tipos siguientes de filtro IIR: Butterworth, Chebycheff tipo I y elíptico. Determina cuál de ellos proporciona el filtro de **menor orden** que cumple la máscara de transferencia anterior. **¿Qué implicaciones tiene que el orden de un filtro sea cada vez mayor?**

> Visualiza el Diagrama de Polos y Ceros de la función de transferencia de cada uno de los filtros obtenidos. **¿Dónde están situados en cada uno? ¿Cuál es el número de polos y ceros en cada filtro? ¿Qué implica eso para el orden del filtro?**

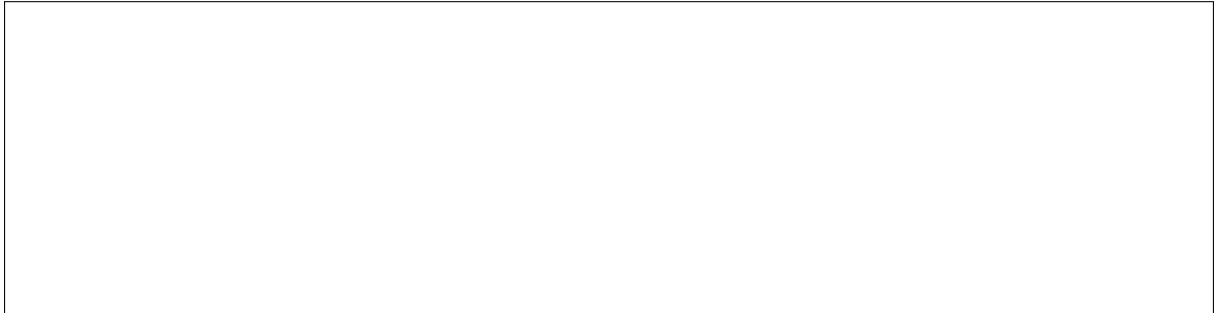
> **Visualiza el módulo de la respuesta en frecuencia** de cada uno de los filtros obtenidos **¿Qué efecto provocan los ceros en el módulo? ¿Y los polos?**

> Haz zoom para ampliar el detalle de la banda de paso y apreciar el rizado (en caso de que exista). Para los filtros que posean rizado en la banda de paso, **¿qué relación encuentras entre el número de máximos y/o mínimos de rizado de la banda de paso y el orden del filtro**, en cada uno de esos filtros?

> Ahora visualiza **la fase de la respuesta en frecuencia** de cada uno de los filtros obtenidos **¿Qué efecto provocan los ceros en la fase?** ¿En qué filtro la respuesta en fase presenta discontinuidades? Indica qué valor de fase tiene dicha discontinuidad y el porqué.

Para estudiar la linealidad de los filtros se debe analizar el **retardo de grupo ¿Cómo se calcula (fórmula)?**

> Visualízelo para los distintos filtros ¿Cuál de ellos tiene un comportamiento menos ‘no lineal’? ¿Qué relación encuentra entre la gráfica del retardo de grupo y el módulo de la respuesta en frecuencia?

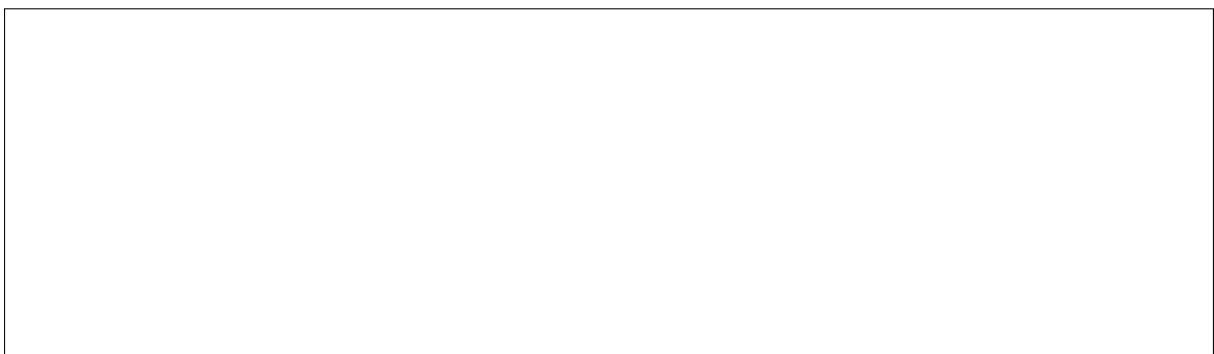


Ahora elegimos un filtro **elíptico** fijando el orden del filtro en valores $N=5$ (impar) y $N=6$ (par), con los mismos parámetros iniciales (salvo la frecuencia de corte de la banda atenuada, que ahora será calculada por MatlabTM en función del orden seleccionado para el filtro).

> Visualiza el módulo de la respuesta en frecuencia y el diagrama de Polos y Ceros para cada uno de los filtros obtenidos. **¿Qué diferencias encuentra? ¿En qué afectan al comportamiento de los filtros?**



> Visualiza la respuesta al impulso de cada uno de los dos filtros. **¿Qué relación encuentra entre la posición de los polos de la función de transferencia y la duración del transitorio?**, ¿y con la rapidez de oscilación de la respuesta al impulso? (por ejemplo, toma como referencia el número de oscilaciones y el tiempo que tarda la respuesta al impulso en atenuarse por debajo de 0.03 dB de amplitud).



> Finalmente, visualiza la respuesta al escalón de cada uno de los dos filtros. ¿Cuánto vale la señal de salida (amplitud) una vez pasado el régimen transitorio en los dos filtros? (haz zoom para ver el detalle). **¿A qué puede deberse ese diferente comportamiento entre ambos filtros?** (como guía, relaciona la contestación con la respuesta en frecuencia de ambos filtros y el diagrama de polos y ceros).



Práctica 5

Filtrado Digital

Nota: para esta práctica necesitarás unos auriculares.

Objetivos:

- Estudiar el diseño de filtros FIR e IIR.
- Emplear dichos filtros para el filtrado de una señal de audio real.

5.1. Introducción

En esta práctica (2 sesiones) se pretende profundizar en el filtrado digital por medio del diseño de filtros y su aplicación para el filtrado digital de una señal real.

Como se ha visto en clase, los sistemas LIT (lineales e invariantes en el tiempo) pueden funcionar como filtros selectivos en frecuencia. Estos sistemas pueden tener una respuesta impulsiva finita (FIR) y, por tanto, implementarse mediante convolución, o bien una respuesta impulsiva infinita (IIR), siendo entonces necesaria la ecuación en diferencias recursiva del sistema. En esta práctica veremos cómo diseñar filtros FIR e IIR en Matlab™, cómo caracterizar la respuesta en frecuencia de ambos y compararlos, y cómo emplearlos en el filtrado digital de señales.

5.2. Realización guiada

Para la realización de la práctica, se dispone de una señal que se corresponde con un audio del canto de un canario, registrado con ruido de ambiente. Por la propia naturaleza de la señal sonora, esta señal posee componentes frecuenciales útiles (sonido del canario) en un determinado rango de frecuencias, siendo el resto correspondiente al ruido de ambiente. Por ello, el filtrado paso banda de la señal supondrá una reducción considerable del ruido de ambiente que afecta a la señal de audio.

5.2.1. La señal de audio

La señal de audio que se va a emplear se encuentra en el fichero `canario.wav`, suministrado como material de esta práctica. Para su lectura en MatlabTM como una señal, basta con cargar la señal de audio en MatlabTM mediante `wavread()`. Además de procesar la señal, en todo momento podremos escucharla por medio de la función `sound()`.

> Carga la señal, **identifica su frecuencia de muestreo** y reproducéla.

Una vez cargada la señal, podemos visualizar tanto la señal en el dominio original como la respuesta en frecuencia de toda la señal. Para ello:

> crea el vector de tiempo a partir de F_s y el vector de frecuencia normalizada.

> Realiza la transformada de la señal mediante la función `fft()` y **visualice conjuntamente mediante un subplot tanto la secuencia temporal como el módulo de su transformada**. Recuerda que para representar correctamente el eje horizontal, debes calcular el vector de tiempo ($t=nT$) y el vector de frecuencia normalizada (f).

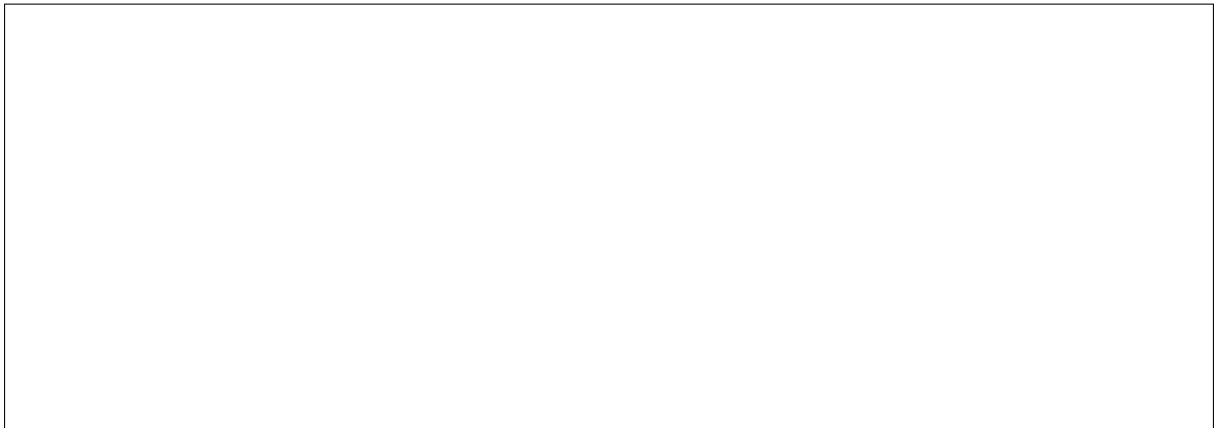
> Una vez representado **¿Puedes identificar el rango de frecuencias de la señal y distinguirlo del ruido?** En ese caso, **indica el rango de frecuencias normalizadas (f) de la señal** $[f_{min}, f_{max}]$. Recordando la relación existente entre f y F a través de F_s , **identifica también el rango de frecuencias (F) de la señal** $[F_{min}, F_{max}]$.

Si lo que deseamos es **ver cómo evoluciona la respuesta en frecuencia de la señal según avanza en el tiempo**, se debe realizar el **espectrograma** de la señal, en nuestro caso con la función `spectrogram()` de MatlabTM. El espectrograma nos muestra cómo evoluciona el contenido en frecuencias de una señal a lo largo del tiempo. Para ello, se toman segmentos cortos de la señal y a cada uno se le calcula la DFT, mostrando únicamente la magnitud del espectro (en dB). Este valor se muestra usando colores en una imagen, en donde el eje vertical se refiere al tiempo (segmento empleado) y el horizontal a la frecuencia. Los colores fríos representan una magnitud pequeña, mientras que los cálidos muestran un valor alto.

> Estudia la función `spectrogram()` y aplícala sobre la señal con los siguientes parámetros:

- Segmentos de la señal de 64 muestras.
- 48 muestras solapadas.
- 64 puntos de muestreo en frecuencia.

Identifica la señal de audio en el espectrograma. Observa que ahora se distingue muy bien la señal del resto de ruido, y que el contenido en frecuencia de la señal se encuentra en el rango determinado previamente.



5.2.2. Diseño del filtro FIR

Para el filtrado de la señal, en este apartado vamos a proceder al diseño de un filtro FIR de fase lineal paso banda, diseñado mediante ventanas. En este método los coeficientes del filtro digital se obtienen como: $b(n) = w(n)h(n)$, $0 \leq n \leq M - 1$. Donde $w(n)$ es una ventana de longitud M ; y $h(n)$ es la respuesta impulsiva ideal del filtro. Esta respuesta se obtiene de la transformada de Fourier inversa de la respuesta en frecuencia ideal.

El filtro que vamos a diseñar contará con 101 coeficientes. Inicialmente emplearemos una **ventana de Hamming**.

> Construye este filtro siguiendo los siguientes pasos:

- **Identifica las frecuencias angulares de corte inferiores y superiores** ($[\omega_{min}, \omega_{max}]$), a partir las frecuencia normalizada de corte inferior y superior en los valores determinados anteriormente para la señal de audio que queremos filtrar, esto es, $[f_{min}, f_{max}]$.

- **Genera la respuesta impulsiva** $h(n)$ para $M = 101$ muestras, a partir de la respuesta impulsiva de un filtro ideal paso banda, desde $-(M-1)/2$ a $(M-1)/2$. Recuerde que la respuesta impulsiva de un filtro paso banda es la siguiente:

$$h_d(n) = \begin{cases} \frac{\omega_{max}}{\pi} \frac{\sin n\omega_{max}}{n\omega_{max}} - \frac{\omega_{min}}{\pi} \frac{\sin n\omega_{min}}{n\omega_{min}} & n \neq 0 \\ \frac{\omega_{max} - \omega_{min}}{\pi} & n = 0 \end{cases} \quad (5.1)$$

- **Multiplica dicha secuencia por una ventana de Hamming** de igual longitud: $b(n) = w(n)h(n)$ (para crearla, emplee la función de MatlabTM `hamming()`).
- **Obtén la respuesta en frecuencia**, $H_{FIR}(\omega)$, a partir de los coeficientes, por medio de la función `freqz()`

> **Visualiza tanto la respuesta impulsiva**, $b(n)$, **como en frecuencia**, $|H_{FIR}(\omega)|$, del filtro diseñado.

Para corroborar que hemos diseñado correctamente nuestro filtro, vamos a emplear la función `fir1()` (dicha función genera automáticamente un filtro diseñado por ventanas),

> **Calcula la respuesta impulsiva mediante `fir1()` y obtén su respuesta en frecuencia. Visualiza superpuestas las respuestas en frecuencia de ambos filtros** (el creado por nosotros y el creado a partir de `fir1()`). **¿Son idénticos?**

Una vez así, estamos en disposición de **filtrar nuestra señal**. Para ello emplearemos la función `filter()`.

> Filtra la señal de audio original y reproduce la señal filtrada. **¿Qué diferencias percibes con respecto a la señal original?**

Cambio de ventana

Para la creación del filtro FIR, se pueden emplear **otras ventanas**: Hanning, rectangular, triangular, etc.

> Crea un nuevo filtro en el que se reemplaza la ventana de Hamming por una ventana triangular (función `triang()`). **Visualiza y compara las respuestas en frecuencia de ambos filtros.**

> Filtra la señal de audio original con el nuevo filtro y reproduce la nueva señal filtrada. **¿Nota diferencias con la señal filtrada anterior?**

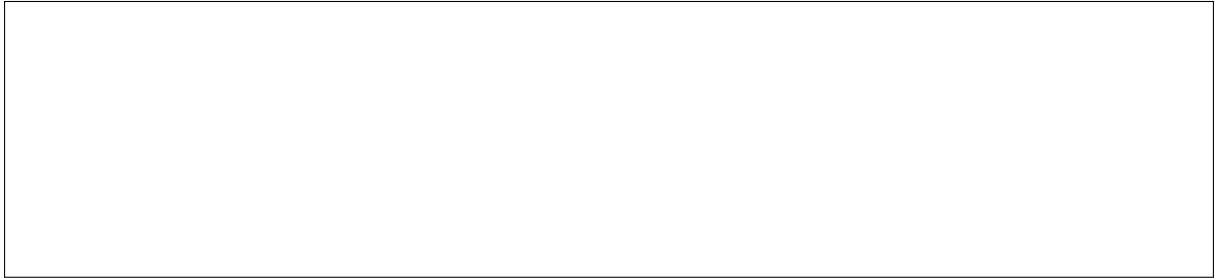
5.2.3. Diseño del filtro IIR

En este apartado vamos a proceder al diseño de un filtro IIR paso banda para el filtrado de la señal. Para ello seleccionaremos un filtro analógico de Butterworth paso banda y mediante transformación bilineal lo convertiremos en digital. Para ello emplearemos la función `butter()` de MatlabTM, que realiza el diseño del filtro analógico y obtiene el correspondiente filtro digital mediante dicha transformación bilineal.

> **Diseña el filtro IIR paso banda propuesto** mediante la función `butter()`, con frecuencias inferior y superior de corte $[f_{min}, f_{max}]$. El filtro debe tener orden 6.



> **Representa el módulo de la respuesta en frecuencia del filtro** obtenido, $|H_{IIR}(\omega)|$.



> Filtra la señal de audio original y reproduce la señal filtrada. **¿Qué diferencias percibes con respecto a la señal original y a la filtrada mediante filtros FIR?**



Finalmente, **compara la respuestas en frecuencia de los filtros FIR e IIR diseñados**. Para ello, visualiza en una misma gráfica tanto $|H_{IIR}(\omega)|$ como $|H_{FIR}(\omega)|$. **¿Qué diferencias encuentras entre ambos?**

