PROCESADO DIGITAL DE SEÑALES
Grado en Ingeniería de Tecnologías de Telecomunicación. Universidade de Vigo

Práctica 1. CAD y CDA. Dos sesiones

Dinámica de las prácticas de laboratorio

En los boletines de prácticas habrá una primera sección a la que denominaremos **Pre-Lab**. Todos los alumnos deben haber repasado los conceptos teóricos que se indiquen en este apartado y hacer los ejercicios que se proponen en él antes de asistir a la práctica. Una vez en el laboratorio el profesor hará una breve introducción de los objetivos de la práctica y de forma conjunta con los alumnos resolverá y discutirá en la pizarra los problemas propuestos en Pre-Lab.

A continuación los alumnos resolverán los problemas propuestos en las demás secciones del boletín de prácticas y podrán consultar sus dudas al profesor durante la realización de los mismos.

En resumen:
Antes de llegar al laboratorio: Pre-Lab. Repaso de conceptos teóricos y realización de ejercicios
Al llegar al laboratorio: Alumnos y profesor resuelven y discuten los ejercicios de Pre-Lab
A continuación: Los alumnos resuelven los demás apartados de la práctica

Evaluación:

En los últimos 20 minutos de la segunda sesión, se hará un examen tipo test de esta práctica a través de la web de la asignatura en faitic.

1. Pre-Lab

El paquete matemático MatLab será el empleado en las clases prácticas de la asignatura. Este software es el más utilizado en universidades y centros de investigación para el procesado de señal (de voz, de audio, de imagen) y para simulación de comunicaciones digitales.

El principal propósito de emplear Matlab en PDS es el hacer posible el adquirir los conceptos de forma práctica, complementando de esta forma las explicaciones teóricas de la asignatura. Existen múltiples textos que explican procesado de señal utilizando MatLab con lo que es fácil ampliar o completar los conocimientos adquiridos en esta asignatura y estas prácticas acudiendo a ellos.

1.1 Objetivos

En asignaturas de primer curso ya se ha empleado Matlab en el laboratorio, por lo que el Pre-Lab de esta práctica se dedica a refrescar los operadores básicos. Los objetivos de esta práctica son:

a) Refrescar los comandos básicos y sintaxis. Ayuda de la aplicación.
b) Escribir y editar scripts y ejecutarlos desde la ventana de comandos.
c) Grabación de señales de audio. Aliasing.
d) Rotulación adecuada del eje horizontal en segundos para simular la forma de onda de una señal analógica.

Los contenidos de esta práctica los vais a usar constantemente tanto en esta asignatura como en futuras asignaturas. **En caso de dudas, recurre a esta práctica (enunciado y resultados).**

1.2. Contenidos teóricos a repasar

Para la segunda sesión de la práctica debes tener al día los siguientes contenidos teóricos:

- Muestreo, frecuencia de muestreo.
- Teorema de Nyquist-Shannon y aliasing.

1.3. MatLab. Tutoriales

MatLab (MATrix LABoratory) es una aplicación informática de cálculo y simulación que dispone de módulos especializados en múltiples disciplinas científicas y técnicas. En una primera aproximación podemos ver MatLab como una potente calculadora matricial que además permite realizar representaciones gráficas tanto en dos como en tres dimensiones.

MatLab proporciona un lenguaje de programación interpretado de alto nivel que permite realizar cálculos muy complejos con pocas líneas de código. Su manejo básico se realiza desde una consola (ventana de comandos), en la que aparece un indicador ("»"), donde se escribirán las instrucciones o nombres de funciones y se pulsará retorno de carro para ejecutarlas.

La Figura 1.1 muestra la ventana de entorno de trabajo cuando se abre el MatLab. La ventana principal se denomina *Command Window*. Observamos que la ventana izquierda nos muestra el contenido del directorio actual (*Current Directory*) que nos informa de los ficheros y carpetas que contiene el directorio actual de trabajo. Este directorio de trabajo puede modificarse pinchando en el icono "**...**" de la *Toolbar.* En el ejemplo de la Figura, tenemos un subdirectorio llamado PR1, una función *.m y un fichero *.wav. Si pinchamos sobre la carpeta PR1, el directorio actual cambiará a ése nuevo; si pinchamos sobre la función, se editará para introducir cambios en ella y si pinchamos sobre el archivo *.wav, se abre una ventana con información de los datos contenidos. Si deseamos crear una nueva carpeta, pincharemos el icono *New Folder* de la ventana del *Current Directory*.

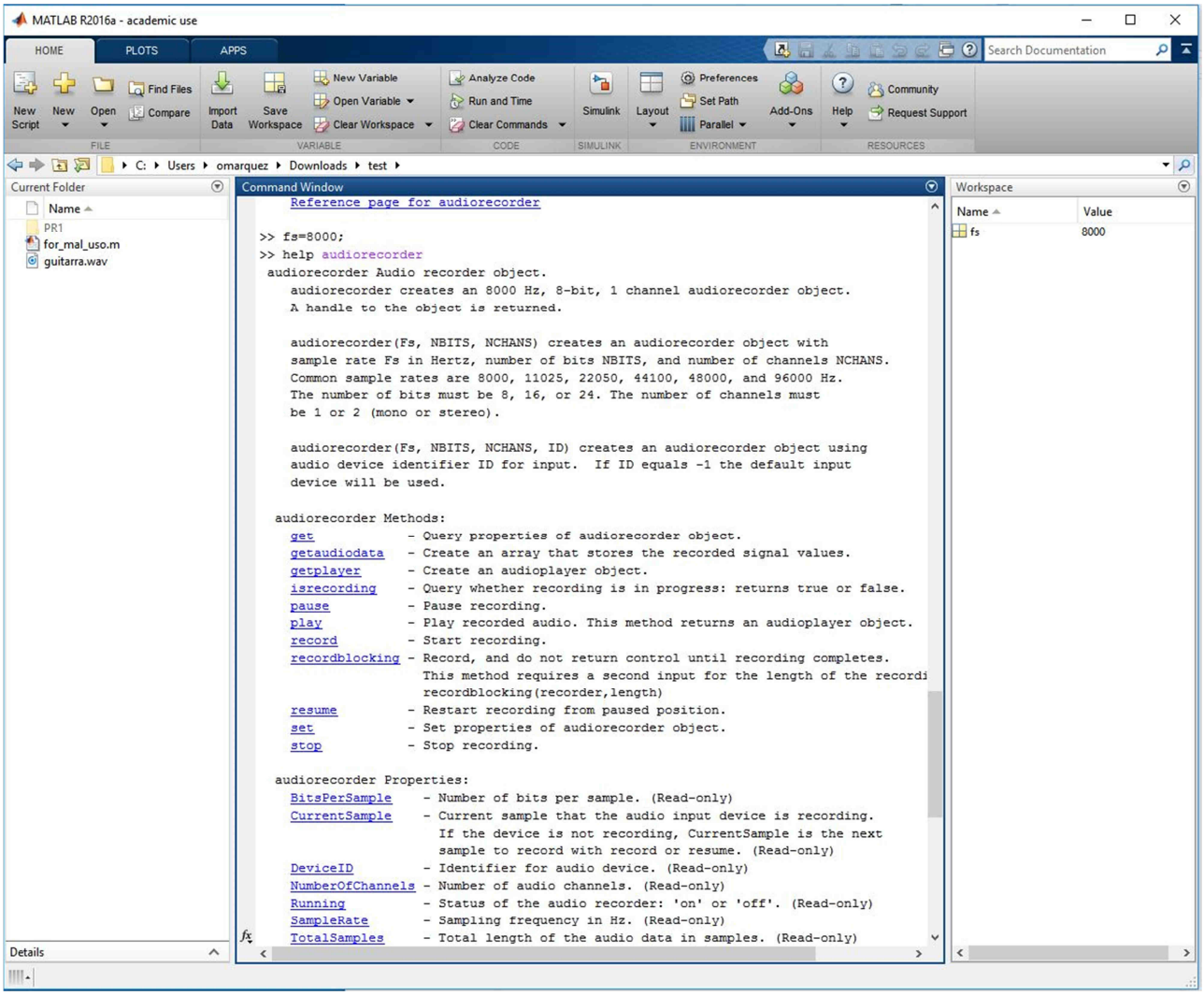

Figura 1.1: Ventana de MatLab R2016a.

La ventana derecha es la de *Workspace* que nos muestra las variables actuales que estamos utilizando en el espacio de trabajo de MatLab y nos da información del nombre de cada variable, de su tamaño así como del tipo de datos que almacena.

Esta visualización que nos muestra el Matlab por defecto, omite una ventana con el histórico de los comandos tecleados (*Command History*) y que puede activarse en la opción *layout* de la barra de herramientas. Esta ventana nos muestra los comandos introducidos hasta el momento. Seleccionando con el ratón alguno de los comandos introducidos anteriormente y pinchando dos veces sobre él, ese comando volverá a ejecutarse. Una manera todavía más cómoda de invocar comandos sin necesidad de teclearlos consiste en activar la línea de comandos (>>), y recuperar los comandos ejecutados anteriormente pulsando repetidamente la tecla de desplazamiento del cursor hacia arriba hasta llegar a la línea antigua que nos interesa. Esta manera es muy útil porque una vez seleccionado un comando, podemos modificarlo antes de ejecutarlo.

En la web de la asignatura, en el tema 1, hay un apartado denominado "Tutoriales MatLab". **Antes de seguir adelante con este documento, y como parte del Pre-Lab, realiza los tutoriales indicados en dicho apartado**.

1.4 Operaciones básicas con MatLab. Tablas resumen

Algunos operadores básicos se resumen en la siguiente tabla:

Operación	Símbolo	Ejemplo
Suma	+	7+6
Resta	-	3-5
Producto	*	2.5*8.9
División	/ o \	56/7=7\56=8
Potencia	^	5^(1/4)
Función Exponencial	exp(x)	exp(-6)

Para generar un vector se pueden utilizar alguno de los métodos que se resumen a continuación:

Definición de vectores y matrices	Comentarios
x=[1 exp(0) 4*7 5]	Los elementos son expresiones que dan un resultado numérico. Deben separarse por espacios o comas y delimitarse mediante corchetes.
x=1:5	Equivale a x=[1 2 3 4 5]
x=1:0.5:3	Equivale a x=[1 1.5 2 2.5 3]
x=1:0.6:3	Equivale a x=[1 1.6 2.2 2.8]
A=[1 2 3; 4 5 6]	Los elementos de una fila se separan mediante espacios en blanco o comas; las diferentes filas se separan mediante punto y coma

En la tabla anterior se puede observar la comodidad que aporta el operador ":". Concretamente la expresión j : k es equivalente a [j, j + 1, j + 2, ..., k] ; si k < j el resultado es una matriz vacía, de dimensiones 0x0; la expresión j : i : k es equivalente a [j, j+i, j+2i, ..., k]. Para más detalles escribe `help colon` en MatLab.

Para acceder a los elementos de un vector se emplean paréntesis. Es importante saber que los índices de los vectores de MatLab se empiezan a contar en 1. Es decir x(0) no está definido. Lo mismo ocurre con las matrices. A continuación se indican algunos ejemplos:

Accediendo a los elementos de un vector o matriz	**Comentarios**
x=[1 exp(0) 4*7 5]; a=x(4)	En la variable a se almacena el cuarto elemento de x (a=5)
A=[1 2 3; 4 5 6] A(2,1)	A(2,1) permite acceder al elemento de la matriz correspondiente a la segunda fila y la primera columna (A(2,1)=4)
A=[1 2 3; 4 5 6] B=A(:,1)	En este caso el operados ":" sirve de comodín para indicar todas las filas. Por tanto B sería la primera columna de A (B=[1;4])
A=[1 2 3; 4 5 6; 7 8 9; 10 11 12] B=A((1:3),[1 3])	(1:3) indica que queremos acceder a los elementos de las filas 1, 2 y 3. Por otra parte [1 3] nos indica que queremos acceder a las columnas 1 y 3. Como resultado B=[1 3; 4 6; 7 9]

Una vez definidos un conjunto de vectores y escalares es sencillo realizar operaciones entre ellos y definir nuevos vectores. MatLab, por defecto, opera con los datos usando funciones matriciales, así el producto X*Y está definido como el producto de matrices y sólo tiene sentido si el número de columnas de X es igual al de filas de Y. De la misma manera las operaciones suma y resta sólo se pueden realizar si las matrices tienen el mismo número de filas y columnas. La única excepción se da cuando hay matrices 1x1, que MatLab trata como escalares. Un escalar se puede sumar o multiplicar por una matriz de cualquier tamaño.

Por otra parte, con un punto precediendo a los operadores del producto (.*), de las divisiones (./ .\) y de la potencia (.^) estas operaciones se llevarán a cabo elemento a elemento. A continuación se especifican algunos ejemplos:

Operaciones entre vectores fila y escalares x=[0 2 4] y=[1 3 5] a=2 b=-3			
Operación	**Tipo de operación**	**Resultado**	**Condición**
z=a+x	Elemento a elemento	z=[2 4 6]	Ninguna
z=b*y	Elemento a elemento	z=[-3 -9 -15]	Ninguna
z=x+y	Matricial	z=[1 5 9]	x e y del mismo tamaño
z=x*y	Matricial	Error!!	x e y matrices cuyo producto esté definido
z=x.*y	Elemento a elemento	z=[0 6 20]	x e y del mismo tamaño
z=x.^y	Elemento a elemento	z=[0 8 1024]	x e y del mismo tamaño
z=x./y	Elemento a elemento	z=[0 2/3 4/5]	x e y del mismo tamaño
z=exp(x)	Elemento a elemento	z=[1 exp(2) exp(4)]	Ninguna

Otra operación común con matrices es la obtención de la matriz traspuesta:

```
MatrizTraspuesta= X.'
```

En el caso de querer obtener la traspuesta conjugada:

```
MatrizTraspuestaConjugada= X'
```

Los símbolos para los operadores relacionales y lógicos en Matlab son los siguientes:

Operadores relacionales		**Operadores lógicos**	
<	menor que	&	AND
<=	menor o igual que	|	OR
>	mayor que	~	NOT
>=	mayor o igual que		
==	igual		
~=	no igual		

1.5 Ejercicios para resolver antes de llegar al laboratorio:

Antes de realizar la práctica en clase, es obligatorio resolver el cuestionario "Test1_PreLab" de la página web. Este test no es puntuable para la evaluación de la asignatura pero no se permitirá empezar la práctica a aquellos alumnos que no lo hayan resuelto previamente.

2. Práctica de Laboratorio

2.1 Captura de sonidos desde Matlab y expresiones sencillas

Con el comando audiorecorder de Matlab se crea un objeto mediante el cual se capturan sonidos de la tarjeta de audio del pc, a la que previamente se han conectado el micrófono y los auriculares. Con el siguiente comando creamos el objeto y visualizamos sus propiedades:

```
>> CapturaAudio = audiorecorder
```

Este comando es equivalente a:

```
>> CapturaAudio = audiorecorder(8000,8,1)
```

La descripción detallada de las propiedades y parámetros del objeto lo conocemos con la ayuda en línea que nos ofrece matlab:

```
>>help audiorecorder
```

a) Vamos a pronunciar la palabra ***Martes*** y capturarla con el método `recordblocking` que nos ofrece `audiorecorder`. El tiempo de captura será de 4 segundos:

```
>> disp('Empieza a hablar.')
>> recordblocking(CapturaAudio,4);
>> disp('Fin de la grabación');
```

Ahora escuchamos y visualizamos los datos:

```
>> play(CapturaAudio);
>> voz=getaudiodata(CapturaAudio);
>> whos
>> plot(voz)
```

Explica el significado de cada una de las variables introducidas y los argumentos de entrada y salida de las funciones empleadas.
¿Cuál es la diferencia entre finalizar la expresión con ";" o sin él?

El operador ":" de Matlab simplifica muchas expresiones y es extremadamente útil en la programación de funciones de procesado de señal.

b) Si visualizamos la señal digitalizada, `plot(voz)`, observamos en la forma de onda que gran parte del vector es ruido. Utilizando el operador ":" crea un nuevo vector `voz_2` con las primeras 20.000 muestras de `voz`.

```
>> voz_2 =          % completa el código
```

c) Ejecuta las siguientes expresiones, escúchalas y explica qué datos contienen `voz_3` y `voz_4`.

```
>> fs = 8000; %Hz
>> voz_3 = voz(end:-1:1); y=audioplayer(voz_3,fs); play(y);
>> voz_4 = voz(1:2:end); y=audioplayer(voz_4,fs); play(y);
>> y=audioplayer(voz_4,fs/2); play(y)

%Las 3 líneas anteriores pueden ser sustituidas por las siguientes
>> voz_3 = voz(end:-1:1); sound(voz_3,fs);
>> voz_4 = voz(1:2:end);  sound(voz_4,fs);
>> sound(voz_4,fs/2);
```

d) Teniendo en cuenta el `help` de las funciones `play`, `sound` y `soundsc`, explica qué hacen los siguientes comandos. Por precaución, separa los auriculares de los oídos.

```
>> [valor_max, indice] = max(abs(voz))
```

```
>> voz_normalizada = voz/valor_max;
>> sound(voz_normalizada, fs)
>> soundsc(voz_normalizada*30,fs)
>> sound(voz_normalizada*30,fs)
```

e) Relaciona la primera expresión del apartado anterior con el comando:

```
>> find(abs(voz)==max(abs(voz)))
```

2.2 Almacenado del workspace

Si accedes a la ventana Workspace, o desde la ventana de comandos tecleas `who`, verás que todas las variables creadas hasta el momento están disponibles. Cuando cerramos la sesión de Matlab, todas estas variables se pierden. Si deseamos almacenar algunos resultados para utilizarlos en una nueva sesión, disponemos de la función `save` que guarda las variables que le indiquemos en un fichero de nombre *nombre.mat*.

a) Teniendo en cuenta el `help` de las funciones `save` y `load`, explica qué hace la siguiente secuencia de comandos.

```
>> save datos_pr1 fs voz voz_3
>> clear fs voz_3
>> who
>> load datos_pr1
>> save datos_pr1
```

Tened cuidado al emplear `clear` porque si se ejecuta sin argumentos, borra todas las variables que tenemos en el workspace.

Las funciones audioread y audiowrite nos permiten leer y almacenar vectores de sonido en el formato que elijamos, según el nombre que le demos al fichero de salida, por ejemplo: fichero.wav.

b) Almacena en un fichero *martes.wav* el vector `voz_2`. Escúchalo desde Windows para comprobar que se ha guardado correctamente.

2.3 Manejo de matrices y vectores

a) Comenzaremos cargando el fichero de audio de 2 canales, *guitarra.wav*:

```
[audio, fsaudio] = audioread('guitarra.wav');
```

Escucha la matriz `audio`.

b) Crea los vectores columna `audio_r` y `audio_l`, que contengan respectivamente el canal derecho (columna 2 del vector `audio`) y el canal izquierdo (columna 1 del vector `audio`). Si escuchas cualquiera de estos dos vectores, se reproducen por los dos auriculares como señal mono.

```
>> % completa el código:
>> audio_r =
>> audio_l =
>> sound(audio_r, ?);
>> sound(audio_l, ?);
```

c) Si deseamos escuchar solamente el canal derecho y por el canal izquierdo no escuchar nada, debemos crear una matriz `audioR`, cuya segunda columna es `audio_r` y cuya primera columna tiene todos los elementos puestos a cero. Crea `audioR`, `audioL` y escúchalos con la función `soundsc`.

```
audioR = audio;
audioR(:, ?) =  ?;
...
```

2.4 Scripts *.m y gráficas

Un fichero *.m de tipo script es un fichero de texto que contiene secuencias de comandos de Matlab. Cuando es invocado desde la ventana de comandos, se ejecuta como si los comandos se hubieran tecleado línea a línea en dicha ventana. La gran ventaja de estos ficheros es la posibilidad de hacer correcciones o cambiar datos sin necesidad de volver a teclear varias líneas de comandos. Además, quedan almacenados en un fichero para sucesivas sesiones.

Abre el editor, pinchando en el icono de documento en blanco de la esquina superior izquierda y teclea las expresiones que se te van pidiendo en el siguiente apartado.

a) **El comando subplot(mnp) divide la ventana en una matriz de m x n gráficas y deja activa la gráfica p.** Escribe el comando `figure(1)` y divide esa figura en dos ventanas horizontales, dibujando en la superior el vector `audio_r` y en la inferior `audio_l`. Cada gráfica debe ir con título.

```
>> figure(1)
>> subplot(211),plot(audio_r)
%Completa el código
...
```

Salva este fichero con el nombre P1_ejer2_4_a.m en una carpeta nueva llamada **Practica2** que crearás en el directorio raíz de vuestra cuenta de trabajo (revisa el apartado 1.3). Para ejecutar el script, o bien, tecleamos su nombre en la ventana de comandos `>>P1_ejer2_4_a`, o bien, desde el propio editor pincha en el icono *Run*.

Para dibujar gráficas, las funciones básicas de Matlab son `plot`, `stem`, `subplot`, `figure`, `title`, `xlabel`, `ylabel`, `grid`, `hold`, `clf`, `axis`.

b) Sigue escribiendo en el script anterior, o crea uno nuevo para los siguientes apartados. Escribe el comando `figure(2)` y divide esa figura en dos ventanas horizontales, dibujando en la superior los 10 primeros valores de `audio_r` con la función `plot`; ídem para la inferior, utilizando `stem`. Emplea `grid` en las dos gráficas.

Observamos que el comando `plot` une las amplitudes del vector simulando una señal continua. Si deseamos asignar al eje horizontal la variable tiempo en segundos, necesitamos crear un vector `t` que contenga los instantes temporales en los que la señal analógica ha sido muestreada. Para ello, debemos tener en cuenta que hemos tomado un intervalo t = [0, T_L) segundos de la señal x(t), que es la música analógica de partida (antes de su grabación en un fichero digital wav). La hemos muestreado cada $T_s = 1/f_s$ s para dar lugar a L=int(T_L f_s) muestras de la secuencia x[n] = x(nT_s) en el intervalo n ϵ [0, int(T_L f_s)) = [0, L) = [0, L-1]. Finalmente hemos convertido esta secuencia en el vector `x` de Matlab (en este ejercicio, vector `audio_r`) de longitud `L = length(x)` elementos y con índices `i=[1,L]`. Estas equivalencias temporales se muestran en la Figura 1.2.

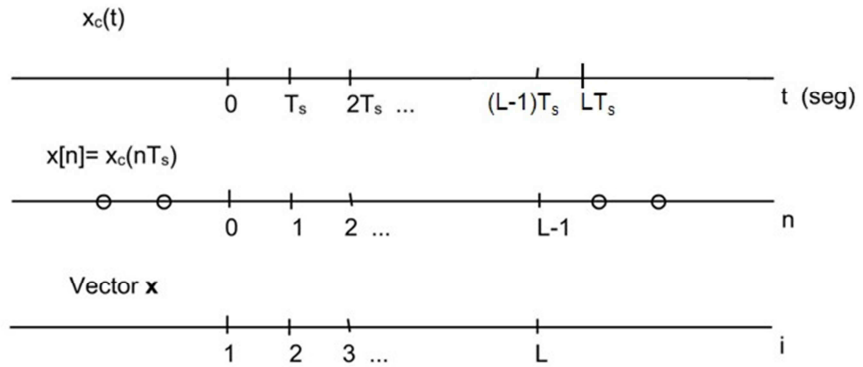

Figura 1.2 Equivalencias entre tiempo continuo, discreto y los índices de un vector de Matlab

Edita un script Pr1_ejer_2_4_c con los comandos que vayas a usar en los siguientes apartados.

c) En primer lugar, vamos a dibujar las primeras 10 muestras del vector audio_r como si fuera una secuencia temporal que comienza en n=0

```
L = length(audio_r)
nn = 0: L -1;
subplot(211), stem(nn(1:10), audio_r(1:10));
title('Secuencia audio_r[n]'); xlabel('n'); grid
```

Para simular el dibujo de la señal analógica, creamos un vector tt:

```
tt = (0: L -1)/fsaudio;
subplot(212), plot(tt(1:10), audio_r(1:10));
title('Señal audio_r(t)'); xlabel('t (s)'); grid
```

d) Dibuja con plot el tramo de `audio_l(20001:28000)`, rotulando el eje horizontal en segundos, adecuadamente.
e) ¿Cuántos segundos dura la señal guitarra.wav?

Los comandos de Matlab para redondear un valor a un número entero son `floor`, `ceil`, `round`.

f) Supón que tenemos 3.75238923 segundos de voz y la muestreamos a fs=8 kHz. ¿Cuántas muestras tenemos? Emplea el `help` de Matlab para averiguar cuál de los comandos `floor`, `ceil` y `round`, servirá para este cálculo.

2.5 Fórmula de Euler

MatLab dispone de una serie de variables muy útiles para operaciones aritméticas: `pi`, `Inf`, `NaN`, `eps`, `i`, `j`, `ans`. El significado de la variable `NaN` es Not a Number. La variable `eps` muestra la resolución numérica de MatLab, mientras que las variables `i` y `j` toman por defecto el valor de sqrt(-1) y se emplean para las operaciones con números complejos. Por su parte, `ans` guarda el último resultado obtenido. Vamos a operar en este apartado con la variable `j`.

En primer lugar, edita un script con nombre Pr1_ejer_2_5.m con los comandos que vayas empleando en los distintos apartados.

a) Genera el vector `x` con 100 puntos de un coseno de frecuencia w0=pi/3 y fase=pi/5. Dibújalo con el comando `stem`, en color azul. A partir de la gráfica, determina el período fundamental:

```
n = 0:99;
w0 = pi/3;
fase = pi/5;
x = cos(w0*n+fase);
stem(n,x)
```

b) Para comprobar la fórmula de Euler, generamos 2 vectores complejos:

```
x1 = exp(j*(w0*n+fase));
x2 = conj(x1);
xcos = (x1+x2)/2;
hold on
stem(n,xcos,?)
```

Dibuja `xcos` en color rojo, superpuesto a la gráfica anterior, usando el comando `hold`. Este comando permite conservar una gráfica en pantalla para visualizar en la misma ventana gráfica un vector diferente. El comando `hold` actúa sobre la última ventana gráfica activa. Si tecleamos `hold off`, la gráfica actual sólo se mantendrá en pantalla hasta un nuevo comando `plot`.

c) Repite los apartados anteriores para comprobar la expresión de Euler de un seno de fase=0.

2.6 Operaciones elemento a elemento

En primer lugar, edita un script con nombre Pr1_ejer_2_6.m con los comandos que vayas empleando en los distintos apartados.

a) Genera los vectores `x0` y `x1` con 100 puntos (n=0:99) de dos cosenos de frecuencias w0=pi/3 y w1 = pi/6, respectivamente, y fase=0. Dibújalas y determina sus períodos fundamentales a partir de las gráficas.
b) Para generar el vector que contenga el producto de las dos sinusoides anteriores, tenemos que realizar esta operación elemento a elemento:

```
y = x0.*x1;
```

Dibuja este vector y determina su período fundamental.

c) Prueba las siguientes expresiones. Usando el comando `size`, explica sus resultados:

```
y1 = x0*x1';
y2 = x0'*x1;
y3 = x0*x1;
```

2.7 Escuchar tonos y aliasing

En primer lugar, edita un script con nombre Pr1_ejer_2_7.m con los comandos que vayas empleando en los distintos apartados.

a) Vamos a generar un vector con 6 frecuencias para generar 6 sinusoides discretas distintas:

```
w0 = 2*pi./[40 20 10 5 4 2];  % rad
```

Para escuchar las sinusoides desde Matlab, tomamos una frecuencia de reconstrucción fs=8000Hz. Sabiendo que la relación entre frecuencia analógica y digital viene dada por $\widehat{\omega}_0 = 2\pi f_0/f_s$, calcula las frecuencias analógicas correspondientes del anterior vector.

b) Completa los siguientes comandos para poder escuchar uno a uno los tonos de frecuencias calculadas anteriormente:

```
n = 0:9999;
fs = 8000;
for i=1: length(?)
       x = cos(w0(?)*n);
       soundsc(x,fs)
       pause
end
```

c) Repite el apartado anterior para las frecuencias

```
w1 = 2*pi./[40 20 10 5 4 2] + 2*pi;
```

y explica el fenómeno que ha ocurrido.

2.8 Uso del bucle for

El bucle `for` en Matlab es muy ineficiente cuando puede ser reemplazado por el operador dos puntos. Vamos a comprobarlo con el siguiente ejemplo.

a) Escribe en la ventana de comandos la línea: `a=1:100000`

b) Escribe ahora

```
for i=1:100000
      b(i)=i;
end
```

c) Los vectores `a` y `b` son idénticos pero habrás notado que la generación de `b` consume mucho más tiempo que la de `a`. Para comprobar el tiempo que se tarda en generar cada uno de estos vectores, abre el script for_mal_uso.m y analiza su código. Los funciones `tic` y `toc` sirven para calcular el tiempo que transcurre entre la llamada de la primera y de la segunda. Ejecuta este script y observa la diferencia de tiempo de generación de ambos vectores que, dependiendo del ordenador, llega a varios órdenes de magnitud.

Recuerda este ejercicio siempre que vayas a programar con Matlab.

2.9 DTMF (Dual Tone Multifrequency)

En telefonía, el sistema de **marcación por tonos**, también llamado sistema multifrecuencial o **DTMF** (Dual-Tone Multi-Frequency) identifica a cada número del teclado por una señal que consiste en la suma de dos tonos de distinta frecuencia. La tabla que asigna la combinación de frecuencias a cada digito es la siguiente:

DTMF	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Por tanto, si marcamos un 4 en el teclado de un teléfono DTMF, la señal que se genera es de la forma $x(t)=(\cos(2\pi 770t)) + \cos(2\,\pi\, 1209t))$.

a) Supón que la señal anterior se muestrea a 8192Hz. Escribe una expresión para la secuencia discreta x[n] obtenida y representa gráficamente en un papel su espectro.

b) Escribe el código MatLab para generar la secuencia x[n] del apartado anterior, asumiendo que la señal continua que se muestrea está definida entre t=0 y t=0.5s. ¿Cuántas muestras tiene x[n]?

c) Escribe el código MatLab para representar gráficamente con `stem` el vector *x,* correspondiente a las 100 primeras muestras de x[n].

d) Reproduce el vector *x* empleando como frecuencia de reconstrucción la misma que se definió para el muestreo.

e) Supongamos ahora que la señal x(t) correspondiente a la pulsación del 4 se muestrea a una frecuencia de 1979Hz. ¿Consideras que esta frecuencia es adecuada? ¿Por qué?

f) Escribe una expresión para la secuencia discreta y[n] obtenida muestreando x(t) a 1979Hz y representa gráficamente en el papel su espectro. ¿Qué ha ocurrido por efecto de la baja tasa de muestreo?

g) Escribe el código MatLab para generar el vector *y* correspondiente a 0.5 s de la señal x(t) del apartado anterior.

h) Escucha mediante el comando *sound* las secuencias *x* e *y* ¿Aprecias la diferencia?

i) Asumiendo que las señales que genera el teclado del teléfono tengan duración infinita, ¿a qué frecuencia las muestrearías si deseas enviarlas en formato digital para poder reconstruirlas posteriormente sin perder información? Debes definir una frecuencia de muestreo que sea adecuada para cualquiera de los números o símbolos del teclado telefónico.