Señales y Sistemas
Laboratorio 2 (4h)
Series de Fourier: señales sinusoidales y fasores.
Filtrado: sistemas DTMF de marcado telefónico

Profesor responsable: Luis Vielva

Curso 2002/2003

Materiales: (a) El alumno debe traer unos auriculares similares a los de un reproductor portatil. (b) Todos los ordenadores están equipados con Matlab y tarjeta reproductora de sonido.

Objetivos: (a) Comprender la relación entre los fasores y las señales sinusoidales, ilustrando su aplicación a las series de Fourier. (b) Ilustrar los fundamentos del filtrado de señales. (c) Aprender cómo utiliza el sistema telefónico señales sinusoidales de distintas frecuencias para indicar qué tecla se ha pulsado. (d) Generar tonos DTMF a partir de la información del símbolo a marcar. (e) Identificar símbolos a partir de señales de audio con tonos DTMF.

Almacenamiento de los ficheros: Todos los ejemplos y ejercicios que se proponen en este laboratorio deben crearse en ficheros y almacenarse en el disco duro del alumno. Para ello, se debe crear una carpeta con el nombre `lab2` en la unidad `X:` al mismo nivel que las carpetas `Ficheros` y `lab1`.

1. Series de Fourier: señales sinusoidales y fasores

Introducción

La exponencial compleja $x(t) = e^{j\Omega_0 t+\phi}$, puede considerarse como un número en el plano complejo —un fasor— con módulo unitario constante y una fase que varía linealmente $\angle x(t) = \phi + \Omega_0 t$. A medida que el tiempo avanza, el fasor rota en sentido antihorario con frecuencia angular Ω_0. Por otra parte, la señal puede expresarse como

$$x(t) = \cos(\Omega_0 t + \phi) + j \sin(\Omega_0 t + \phi),$$

poniendo de manifiesto que la parte real es un coseno y la imaginaria un seno.

Experimento[1]

Conéctate a la página web `gtas.dicom.unican.es/ss/Apuntes.htm`. En la parte inferior de la página encontrarás cinco applets de java para experimentar con los fasores y su relación con las series de Fourier. En todos los applets se representa en la izquierda una señal en el plano complejo y en la derecha la parte imaginaria de la misma señal. El usuario puede interactuar dibujando fasores en el plano complejo y observando su parte real.

La primera ventana permite seleccionar mediante el botón izquierdo del ratón un fasor en el plano complejo, que corresponde a $x(t)$ para $t = 0$. Mediante el botón `Play` se puede generar la señal compleja asociada y su parte real. Experimenta con distintos fasores y observa la relación entre las fases en ambas representaciones.

La segunda ventana es idéntica a la anterior, pero permite dibujar dos fasores, que se corresponden con el armónico fundamental y el segundo armónico de la señal. La señal que se compone de esta forma es

$$x(t) = |a_1| \exp(j\Omega_0 t + \phi_1) + |a_2| \exp(j2\Omega_0 t + \phi_2),$$

donde el fasor n-ésimo es $a_n = |a_n|e^{j\phi_n}$. Es decir, $x(t)$ es una señal cuya serie de Fourier tiene N coeficientes no nulos: los a_n. Observa cómo se suman los dos fasores en el plano complejo y la parte imaginaria. Experimenta con distintos fasores.

La tercera ventana es idéntica a la anterior, pero permite dibujar múltiples fasores, que se corresponden con los distintos armónicos de la señal. Así, el primer fasor que dibujes es la frecuencia fundamental, el segundo que dibujes el segundo armónico, etc. La señal que se compone de esta forma es

$$x(t) = \sum_{n=1}^{N} |a_n| \exp(jn\Omega_0 t + \phi_n),$$

donde el fasor n-ésimo es $a_n = |a_n|e^{j\phi_n}$. Experimenta con distinto número de fasores.

En la cuarta ventana no se pueden definir manualmente los fasores, sino que se pueden seleccionar tres tipos de señales: cuadrada, triangular e impulso. El applet realiza una análisis mediante series de Fourier e identifica tantos armónicos como se le indiquen en la ventana `Terms`. Experimenta con distintos tipos de señales y número de armónicos. Observa la forma de los armónicos para la señal impulso.

La quinta ventana es idéntica a la anterior, pero permite realizar un proceso de filtrado sobre la señal para eliminar el rizado. Experimenta con los distintos tipos de filtrado.

2. Filtrado: sistemas DTMF de marcado telefónico

Introducción

La mayor parte de los teléfonos actuales utilizan un sistema de marcación en el que cada número está identificado por una señal compuesta por la suma de dos sinusoides. Este

[1] No dediques más de unos quince minutos a este apartado. Siempre tendrás disponible los ejemplos en la página web de la asignatura

	f columna (Hz)		
f fila (Hz)	1209	1336	1477
697	1	2	3
770	4	5	6
852	7	8	9
941	$*$	0	#

Cuadro 1: Cada dígito está compuesto por dos señales sinusoidales, con las frecuencias indicadas según su fila y columna.

sistema, denominado DTMF (*Dual Tone Multi Frequency*), ha ido sustituyendo progresivamente al marcado por pulsos que utilizaron los teléfonos de marcado rotatorio. Las dos frecuencias que componen cada uno de los símbolos se organizan en una matriz, tal y como se muestra en la tabla 1. La frecuencia más alta está asociada a la columna del símbolo en el teclado, y la frecuencia inferior a la fila.[2] Mediante las siete frecuencias mostradas, tomadas de dos en dos, se pueden generar los códigos DTMF correspondientes a los doce símbolos (diez dígitos y los caracteres especiales $*$ y #) que están presentes en todos los teléfonos.

El laboratorio consta de dos partes. La primera consiste en generar tonos DTMF en respuesta a una secuencia que representa el símbolo a marcar. La segunda consiste en identificar los números marcados a partir de la señal de audio.

Parte I: Generación de señales DTMF

Las señales DTMF están compuestas por dos sinusoides, cada uno de los símbolos obedece a la expresión

$$s(t) = \sin(2\pi f_1 t) + \sin(2\pi f_2 t) \equiv \sin(\Omega_1 t) + \sin(\Omega_2 t).$$

Construye una macro, `ex1.m`, que cree una figura con dos subfiguras, una superior y otra inferior. En la superior utiliza `ezplot` para dibujar 10 ms de la señal correspondiente al dígito 1. En la inferior, utiliza la función `ezplot` para dibujar 10 ms de la señal correspondiente al dígito 8.

Para trabajar en un sistema discreto, como un ordenador, las señales continuas se deben discretizar. El método de discretización más habitual es el muestreo uniforme, que consiste en generar una secuencia tomando muestras equiespaciadas T segundos —el periodo de muestreo— de la señal continua: $s[n] = s(nT)$. Por lo tanto, la secuencia discreta asociada a la señal DTMF es

$$s[n] = \sin(2\pi f_1 nT) + \sin(2\pi f_2 nT) \equiv \sin(\omega_1 n) + \sin(\omega_2 n).$$

[2]Estas frecuencias se escogieron para evitar armónicos. Ninguna frecuencia es múltiplo de otra, la diferencia entre cada dos frecuencias no es igual a ninguna frecuencia de la tabla. Esta elección hace más sencilla la detección en presencia de distorsiones (no linealidades) en la línea.

Observa la relación entre la frecuencia continua Ω (rad/s) y la frecuencia discreta ω(rad):

$$\Omega = \frac{\omega}{T} = \omega f_s,$$

donde $f_s = 1/T$ es la frecuencia de muestreo.

Construye una función, `seno.m`, que genere y devuelva una secuencia sinusoidal discreta. El formato de la función debe ser el siguiente:

```
function s = seno(f, fs, Duracion)
...
```

donde `f` es la frecuencia continua, `fs` es la frecuencia de muestreo, y `Duracion` es la duración en segundos de la señal continua muestreada.

Construye una macro, `ex2.m`, que cree una figura con dos subfiguras, una superior y otra inferior. En la superior utiliza `ezplot` para dibujar 10 ms de la señal correspondiente al dígito 1. En la inferior utiliza la función `seno.m` que acabas de crear para dibujar las muestras correspondientes a 10 ms de la señal correspondiente al dígito 1 muestreada con una `fs` de 8000.

Construye una función, `tonodtmf.m`, que sirva para generar las muestras correspondientes a un tono DTMF. La función **no** debe invocar a la función `soundsc` para hacer sonar el tono, sino `sólo` devolver un vector con las muestras. La función debe utilizar internamente la función `seno.m` y utilizar `fs=8000`. La sintaxis de la función debe ser la siguiente

```
function x = tonodtmf(Simbolo, Duracion)
```

Donde `Simbolo` es un carácter —por ejemplo '1' o '*'—. Para realizar la asociación entre símbolos y frecuencias, podrías utilizar el siguiente código:

```
s = '123456789*0#';
ff = [697 697 697 770 770 770 852 852 852 941 941 941];
fc = [1209 1336 1477 1209 1336 1477 1209 1336 1477 1209 1336 1477];
i = find(s == Simbolo);
f1 = ff(i);
f2 = ff(i);
```

Construye la siguiente macro, `ex3.m`, y escucha el resultado:

```
for k=['1' '8' '*' '0']
    t = tonodtmf(k,0.5);
    soundsc(t, 8000);
    plot(t);
    pause;
end
```

Construye una función, `telefono.m`, que mediante llamadas sucesivas a la función `tonodtmf.m` construya un vector con los tonos DTMF correspondientes al número de teléfono que se le pase como argumento. La sintaxis de la función debe ser la siguiente:

```
function x = telefono(NumeroDeTelefono)
```

Donde `NumeroDeTelefono` es una cadena de caracteres con los dígitos a marcar. Cada dígito debe tener una duración de medio segundo, con una separación entre dígitos de una décima de segundo.

La función de Matlab `fft` calcula la transformada discreta de Fourier (DFT) de un vector. Genera una función, `espectro.m`, que permita observar un tono DTMF tanto en el dominio del tiempo como en el de la frecuencia. La función debe crear una figura con dos ventanas. En la superior utiliza `plot(x)` para dibujar la señal en el dominio del tiempo. En la inferior, utiliza

```
plot(abs(fft(x)));
```

para dibujar la señal en el dominio de la frecuencia. Además, la función debe reproducir, mediante `soundsc(x, 8000)`, la forma de onda del tono DTMF. La sintaxis de la función debe ser la siguiente:

```
function espectro(Digito)
```

Crea la siguiente macro y llámala `ex4.m`:

```
espectro('1'); pause;
espectro('4'); pause;
espectro('7'); pause;
espectro('8'); pause;
espectro('9'); pause;
```

Observa en las sucesivas gráficas los picos que aparecen en la figura inferior. Relaciónalos con las frecuencias correspondientes a cada dígito que aparecen en la tabla 1. Describe tus comentarios en el propio fichero `ex4.m`

Genera la siguiente macro y llámala `melodia.m`:

```
Simbolos = '11216311219611#9632##9696'
x = Telefono(Simbolos);
soundsc(x, 8000);
```

¿Te suena a algo conocido? Escribe tus comentarios en el propio fichero.

3. Detección automática de dígitos

En esta segunda parte, deberás construir un sistema que a partir de una señal de audio con las muestras correspondientes a los tonos DTMF de cada dígito, devuelva una cadena de caracteres con los dígitos asociados. La señal de audio puede tener códigos DTMF consecutivos, con distintas duraciones cada uno (en función de cuánto tiempo se mantenga pulsado el botón del teléfono para cada dígito) y con distintos espacios de silencio entre ellos. Para Para resolver este problema es mejor dividirlo en dos etapas. En la primera debe construirse una función que analice la señal de audio y la divida en segmentos, uno por cada dígito. La segunda etapa debe tomar cada uno de estos segmentos e identificar a qué dígito corresponde.

3.1. Segmentación de la señal de audio

La forma más sencilla sería buscar aquellas regiones con ceros en la señal de audio, que se corresponderían a silencios entre dígitos. Sin embargo, este método no es muy robusto, pues es muy sensible a la presencia de ruido en la señal de audio. Un método mejor es calcular la energía local de la señal, entendiendo por energía local la que presenta una ventana deslizante de duración `100` muestras. La forma más sencilla de implementar esta operación es aplicar un filtro de media móvil —la respuesta al impulso es uno durante la duración de la ventana— mediante

```
nVentana = 100;
filtro = ones(1, nVentana);
y = filter(filtro, 1, abs(x));
y = y(nVentana:end);
figure(1); plot(x);
figure(2); plot(y);
```

Donde se ha supuesto que `x` es el vector que contiene la señal de audio a segmentar. Observa ambas figuras y diseña un método para detectar los segmentos de la señal.

3.2. Estimación del dígito codificado

Una vez que ya sabes cómo segmentar la señal, lo que resta es aplicar una técnica para estimar el dígito codificado en un tono DTMF. Existen varias formas de realizar este paso. A continuación describimos dos de ellas; selecciona la que más te guste e impleméntala.

3.2.1. Estimación mediante detección de picos espectrales

En este caso vamos a identificar los picos que aparecen en la DFT de la señal de audio para calcular a qué digito corresponden. La idea es sencilla: dado un tono DTMF, calculamos su DFT, obervamos su módulo e identificamos las frecuencias a las que ocurren los

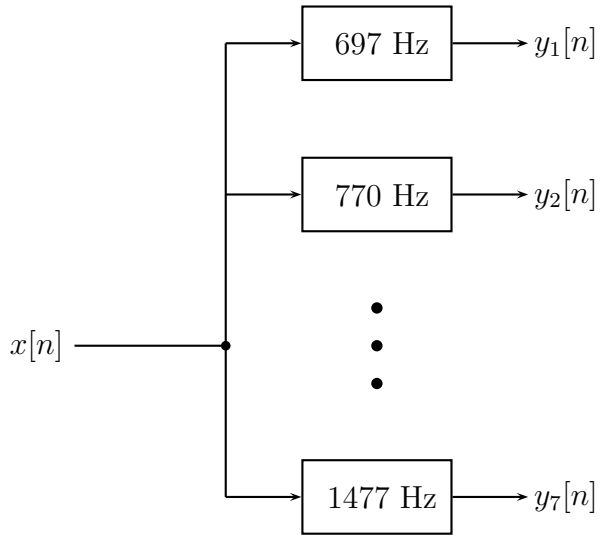

Figura 1: Banco de filtros para el decodificador DTMF. Cada uno de los siete filtros paso banda tiene como frecuencia central una de las siete que componen los símbolos DTMF.

máximos. Con estas frecuencias identificamos a qué frecuencias de la tabla 1 se encuentran más próximas y realizamos una decisión sobre el dígito codificado.

Construye una función `EstimaTonoFFT.m` que devuelva el dígito que se encuentra codificado en el tono DTMF que toma como entrada:

```
function Digito = EstimaTonoFFT(x)
```

Construye una función `EstimaTelefonoFFT.m` que utilice el método de segmentación que realizaste en el apartado anterior para dividir la señal de entrada en códigos individuales. Para cada tono invoca a la función `EstimaTonoFFT.m` para estimar el dígito individual. La sintaxis de la función es la siguiente:

```
function Telefono = EstimaTelefonoFFT(x)
```

3.2.2. Estimación mediante un banco de filtros

Es posible decodificar las señales DTMF utilizando un banco de filtros pasp banda como el que se muestra en la figura 1

Cada uno de estos filtros no es más de un sistema cuya respuesta frecuencial —la transformada de Fourier de la respuesta al impulso— deja pasar las señales alrededor de su frecuencia central y amortigua el resto. Como cada uno de estos subsistemas es un sistema LTI, la salida se obtendrá como el producto de convolución de la entrada por la respuesta al impulso: $h_i[n] = x[n] * h_i[n]$. Esta operación puede implementarse en Matlab mediante la función `filter`:

```
y1[n] = filter(h1[n], 1, x[n]);
      .
      .
      .
y7[n] = filter(h7[n], 1, x[n]);
```

Idealmente, al hacer pasar un código DTMF a través del banco de filtros, dos de las salidas $y_i[n]$ serán prácticamente una señal sinusoidal de amplitud apreciable y las otras cinco serán prácticamente cero. Un método para detectar qué dos frecuencias estaban presentes en la señal $x[n]$ es calcular la energía de las señales de salida.

Para diseñar los filtros pueden utilizarse muchos métodos, pero aquí recurriremos a filtros sencillos cuya respuesta al impulso es simplemente un coseno finito de la forma

$$h[n] = \frac{2}{L} \cos\left(\frac{2\pi f_b n}{f_s}\right), \quad 0 \leq n < L,$$

donde L es la longitud del filtro, f_s es la frecuencia de muestreo, el parámetro f_b define la localización frecuencial de la banda de paso —por ejemplo, escogemo $f_b = 697$ si queremos seleccionar la componente de 697 Hz—. El ancho de banda del filtro está controlado por L: cuanto mayor sea L, menor será el ancho de banda. Utiliza en todos los casos $f_s = 8000$ y $L = 64$.

Genera la macro `ex5.m` con las siguientes líneas para diseñar un filtro paso banda y dibujar tanto su respuesta al impulso como su respuesta frecuencial:

```
fs=8000;
fb=770;
L=64;
n=0:L-1;
h2 = 2/L * cos(2*pi*fb*n/fs);
figure(1); stem(h2);
ww = 0:(pi/256):pi; % solo frecuencias positivas
ff = ww/(2*pi)*fs;
H = freqz(h2, 1, ww);
figure(2); plot(ff, abs(H)); grid on
```

Construye una función `EstimaTonoBanco.m` que devuelva el dígito que se encuentra codificado en el tono DTMF que toma como entrada:

```
function Digito = EstimaTonoBanco(x)
```

Construye una función `EstimaTelefonoBanco.m` que utilice el método de segmentación que realizaste en el apartado anterior para dividir la señal de entrada en códigos individuales. Para cada tono invoca a la función `EstimaTonoBanco.m` para estimar el dígito individual. La sintaxis de la función es la siguiente:

```
function Telefono = EstimaTelefonoBanco(x)
```