REPRESENTACIONES TIEMPO/FRECUENCIA (ESPECTROGRAMA)

APELLIDOS:	NOMBRE:
APELLIDOS:	NOMBRE:

Fecha de entrega: hasta el 9 de enero

En esta parte vamos a estudiar una extensión muy importante en la práctica del análisis de Fourier, el llamado análisis de Fourier local. Este es el caso más sencillo de las llamadas representaciones tiempo/frecuencia. Hasta ahora hemos visto que una señal tiene una representación temporal ($s(t)$) o en frecuencias ($S(\omega)$). Sin embargo es posible trabajar con representaciones conjuntas tiempo/frecuencia que nos en información simultánea (aunque con limitaciones) sobre lo que está pasando en ambos dominios.

Consideremos un ejemplo práctico donde se aprecia claramente la necesidad de este nuevo enfoque. En la práctica de modulación indicamos que la modulación AM estudiada era solamente una de las posibilidades, modificar el parámetro de amplitud de la portadora. Consideremos el caso de modulación de la frecuencia (FM) para el caso particular en el que la señal a transmitir es binaria, presentando sólo dos valores, típicamente 0 y 1. Técnicamente es lo que se denomina por las siglas FSK (Frequency Shift Key).

En este tipo de modulación se eligen 2 frecuencias f_0 y f_1 y dependiendo del valor de la señal binaria se manda una u otra. Este tipo de modulación era usado comunmente por los *modems* para la transmisión de datos binarios a través de la línea telefónica, usando frecuencias audibles. La siguiente función `bin_fm` simula dicha modulación:

```
function [w,bb]=bin_fm(b,T,F,f0,f1)
% Genera una onda modulada en frecuencia a partir de b[] (una ristra de 0/1)
% T : duración bit en segundos  (por defecto 1/50 sec -> 50 bits/sec)
% F : frecuencia de muestreo (por defecto 4000 Hz)
% f0 y f1 : frecuencias para un 0 y 1 respectivamente (por defecto 200 y 250 Hz)
% Como segundo valor devuelve una onda cuadrada generada a partir de
% b[], muestreada a frecuencia F y donde cada bit dura T segundos.
```

Veamos como funciona esta función. Con la función `txt2bin` generamos un vector de 0/1 correspondiente p.e. a la representación en código ASCII de un cierto mensaje de texto:

```
b = txt2bin('Tratamiento digital');
```

A continuación le aplicamos la función anterior `bin_fm(b)` visualizando el resultado con `show2()`:

```
F=2000;
[bf bb]=bin_fm(b,1/50,F);
show2(bb,bf,F)
```


Para intentar demodular el mensaje, y dado que la modulación consiste en un cambio de la frecuencia parece obvio recurrir a la TF (o a su version discreta, la DFT) para analizar la señal recibida `bf[]`:

```
ver_tf(bf,F,'r','semi');  % Pintamos solo frecuencias "positivas"
```

Determinar usando `ginput(2)` cuáles son las dos frecuencias más importantes presentes? ¿Podemos reconstruir el mensaje original a partir de la gráfica anterior? ¿Por qué?

La solución a este problema es sencilla pero muy versatil y potente. Consiste en "localizar" el análisis de Fourier. Para ello basta multiplicar la señal por una cierta ventana centrada en la zona en la que estamos interesados, como paso previo a la TF. Formalmente:

$$S(t,\omega) = \int_{-\infty}^{\infty} s(\tau)v(\tau - t)e^{-i\omega\tau}d\tau$$

y su correspondiente versión discreta para secuencias:

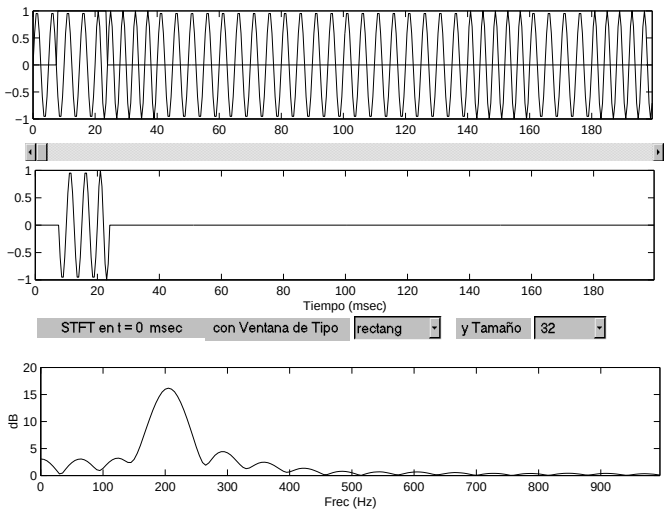
$$S(n, e^{i\omega}) = \sum_{m=-\infty}^{\infty} \left(s[m]v[m-n]\right) e^{-i\omega n}$$

El parámetro t (n para secuencias) indica dónde situamos la ventana $v(t)$ (y por lo tanto que zona de la señal estudiamos) mientras que el parámetro de ω (k) corresponde a las frecuencias presentes en dicha zona (ya no en toda la señal). Se observa que el resultado es una señal de 2 variables, ya que para cada posición de la ventana t tenemos todo un espectro (la TF de la señal ventaneada). La función anterior recibe muy diversos nombres, como por ejemplo transformada local de Fourier, transformada de Fourier de ventana deslizante, análisis local de Fourier, etc. Típicamente se suele trabajar con el módulo de las anteriores funciones, que es lo que se denomina espectrograma. Además, la función $v(t)$ suele tener un soporte finito para darnos solo informacion de una zona controlada.

La siguiente función, `stft()` (de Short Time Fourier Transform) nos va a permitir visualizar estos conceptos:

```
function stft(x,F)
% Muestra de forma gráfica el resultado de una transformada local de Fourier sobre la señal x[],
% muestreada con frecuencia F Hz.  Permite seleccionar diferentes tipos y tamaños de ventana.
```

Analicemos las primeras 800 muestras (400 msec) de la señal anterior modulada en frecuencia `bf[]` usando `stft(bf(1:800),F)`:



La fila superior muestra la señal original y de forma superpuesta la ventana que estamos usando. En la fila intermedia mostramos la señal "ventaneada", y en la fila inferior la correspondiente TF de dicha señal ventaneada, que se puede representar en amplitud o en escala logarítmica (dB). Con el cursor podeis elegir interactivamente el punto de aplicación de la ventana. En los menus se puede seleccionar también el tipo de ventana y escoger su anchura.

Comprobad como ahora si podeis "demodular" el mensaje moviendo la posición de la ventana y observando donde se situa el pico en frecuencias en el correspondiente espectro. Seleccionar un ancho de ventana similar a los 20 msec de nuestros bits y pulsar en la flecha del *slider* viendo como el pico de frecuencias va saltando entre 200 y 250 Hz según tengamos un 0 o un 1 en el mensaje.

Podemos experimentar con el tamaño de la ventana. Probad a incrementar/decrementar el valor de M (y por consiguiente el soporte temporal de la DFT, M/F). ¿Qué sucede si se hace demasiado grande (tamaño $\simeq 50-100$ msec) ? ¿Y si es demasiado pequeña (tamaño $\simeq 8$ msec)? ¿Se puede en esos casos determinar fácilmente el bit correspondiente?

Recordad que la máxima resolución alcanzable al hacer una DFT dependia del tiempo de observación y no de la frecuencia de muestreo. En este caso el tiempo de observación para cada DFT es el ancho de la ventana $T_{obs} = A_v$, por lo que la máxima resolución en frecuencias será del orden de su inverso:

$$\Delta f \simeq \frac{1}{A_v}$$

Por otro lado el ancho de la ventana determina la resolución temporal (Δt). Si dos frecuencias caen dentro de la ventana (separación menor de A_v segundos), no podremos determinar que frecuencia aparece antes o despues (las vemos juntas al hacer la TF). Por lo tanto:

$$\Delta t \simeq A_v$$

¿Qué ocurre con el producto de Δf y Δt? ¿Qué pasa si disminuimos el tamaño de nuestra ventana? ¿Y si lo aumentamos?

Este límite nos va a aparecer siempre que manejemos representaciones conjuntas tiempo-frecuencia. De hecho podemos considerar a las típicas representaciones en el tiempo $s(t)$ o en la frecuencia $S(\omega)$ como casos extremos de una representación conjunta:

Tiempo	$\Leftrightarrow$	Conjunta	$\Leftrightarrow$	Frecuencia
$s(t)$	$\Leftrightarrow$	$S(t,\omega)$	$\Leftrightarrow$	$S(\omega)$
$\Delta t = 0$		Δt		$\Delta t = \infty$
$\Delta \omega = \infty$		$\Delta \omega$		$\Delta \omega = 0$

Una de sus ventajas de la modulación FSK es su alta resistencia al ruido. Haciendo un `load incognita` cargamos una secuencia de 8 bits (con 20 msec de duración cada bit) modulados con FSK que representan una letra. Llevan añadido un gran nivel de ruido por lo que a simple vista no sereis capaces de discriminar las frecuencias presentes. Sabiendo que los ceros se representan con $f_1 = 200$ Hz y los unos por $f_2 = 250Hz$ hallar, usando una ventana de Hanning de unos 20 msec, de qué letra estamos hablando (la función `bin2txt()` convierte un array de bits en su correspondiente caracter ASCII).

b0	b1	b2	b3	b4	b5	b6	b7	Caracter

Marcación por tonos en la red telefónica

El esquema anterior es también válido para analizar señales con modulaciones de frecuencia más complicada. Este es el caso de la marcación por tonos en la red telefónica conmutada.

Al contrario que en FSK donde sólo teniamos dos frecuencias y sólo una u otra estaban presentes en la señal en un momento dado, cada vez que pulsamos una tecla de nuestro telefono se genera una señal constituida por la suma de dos tonos o frecuencias puras, de acuerdo a la siguiente tabla:

F_1 / F_2	1224 Hz	1368 Hz	1512 Hz
704 Hz	1	2	3
792 Hz	4	5	6
872 Hz	7	8	9
956 Hz	#	0	*

Así por ejemplo, la señal enviada al pulsar el 5 consta de dos frecuencias de aproximadamente 790 y 1370 Hz. Hemos escrito en *Matlab* un par de funciones que nos permiten reproducir estos tonos:

```
function res = gen_dig(d,F)
% >> res = gen_dig(d,F)
% Genera el par de tonos correspondiente a la marcacion del digito d muestreado a F Hz.
```

La segunda función `marca` recibe un string con un número de telefono, lo descompone en digitos individuales y llama a `gen_dig` combinando todos los digitos:

```
function telef=marca(num,F)
% >>  telef=marca(num)
% Recibe la cadena num, p.e. num='1234' y genera un tono de marcado telefonico muestreado a F Hz.
```

Un número de teléfono se generaría por ejemplo con

```
F=8000; tel=marca('112',F);
```

Los valores de duración y espaciado de los tonos están dentro de las tolerancias admitidas por la red telefónica, por lo que un `sound(tel,F)` ante el micrófono de nuestro teléfono provocará una llamada.

Al igual que en el caso anterior está claro qué para decidir qué número se ha marcado y actuar en consecuencia es preciso analizar la señal de tonos de una forma similar a como se hizo antes.

Sin embargo, la presentación anterior tenía el inconveniente de que sólo se podia visualizar una TF local al mismo tiempo. Otra posibilidad dado el caracter de representación 2D (t, ω) de la TF local sería visualizarla como una imagen, donde uno de los ejes representase tiempo y el otro frecuencia:

el color o nivel de gris de la imagen representaría el módulo de la correspondiente TF. De está forma se podría evaluar de un sólo vistazo la evolución de la diversas frecuencias en una señal en el tiempo. Para ello dividiremos la señal total en segmentos de M muestras (correspondientes a un tiempo $\Delta t = M/F$. Por razones de eficiencia escogeremos dicho M una potencia de 2. Cada segmento será ventaneado y su DFT calculada. Ahora cada DFT representa un análisis de Fourier del intervalo Δt correspondiente.

A continuación organizaremos dichas DFT como columnas de una matriz, de forma que moverse a traves de una fila equivale a estudiar el comportamiento de una frecuencia dada en el tiempo, mientras que al seguir una columna estamos estudiando las diversas frecuencias presentes en un intervalo de tiempo dado. Implementar esto en *Matlab* es especialmente sencillo, dada su caracter vectorial:

```
F=6144; telef=marca('5551234',F);
M=512;                          % Tamaño de la DFT -> 512/6144 = 80 msec
N=floor(length(telef)/M);       % Numero de DFTs a llevar a cabo
s=reshape(telef(1:N*M),M,N);    % Organizamos en matriz  M filas x N cols
W=hanning(M); s=diag(W)*s;      % Aplicamos ventana a todas las filas
S=abs(fft(s));                  % DFT sobre las columnas de s y modulo
S=20*log10(S+eps);              % Representación log en dBs
S=S(1:M/2,:);                   % Retenemos las M/2 frecuencias no redundantes
image(S);                       % Pintamos S como una imagen
```

¿Que representa en la imagen obtenida el eje X? ¿Y el eje Y?. Para un cierto valor del eje X ¿qué nos encontramos en el eje Y?

Ahora solo faltaría poner los ejes bonitos, etc, sabiendo que cada columna de nuestra nueva representación supone un avance de Δt en el tiempo y que el eje en frecuencias va desde 0 hasta F/2 (la mitad de la frecuencia de muestreo) en saltos de $\Delta f = 1/\Delta t$. La función `stft2d()` hace todo esto

```
function stft2d(data,M,F)
% Pinta espectrograma 2D de una señal 1D data() muestreada a F Hz
% usando una ventana de M muestras y un ventaneado de Hanning
```

La función `stft2d()` tras presentar los resultados entra en un bucle donde podemos pinchar usando el boton izquierdo del ratón y nos indicará la posición en la que hemos pinchado. Para salir, pulsar boton derecho. Si ahora llamamos a la función anterior con un cierto numero de telefóno

```
load telef            % Carga telef[] conteniendo un cierto número
sound(telef,F);
stft2d(telef,512,F);  % Espectrograma con M=512 (unos 80 msec para esta F)
```

al estar los ejes propiamente escalados, usando el ratón podemos determinar las sucesivas frecuencias presentes y los correspondientes dígitos de nuestro número incognita:

F1									
F2									
Dígito									

Espectrograma de señales de voz: Modelo simple de sonidos vocálicos

Vamos ahora a aplicar la herramienta anterior a señales de voz. De hecho, el uso de espectrogramas empezó en los años 40 aplicado al estudio de señales de voz. Nosotros aplicaremos el espectrograma a un caso especialmente simple de la voz humana, los sonidos vocálicos.

En su modelo más simple un sonido vocálico ('a','e', 'i', 'o', 'u') es producido cuando un tren de pulsos de periodo T (generado por la vibración de las cuerdas vocales) pasa a través del tracto vocal (constituido por boca, laringe, etc.) colocado en una posición dada para pronunciar una p.e. una 'a'. El efecto del tracto vocal es simulado como un sistema lineal invariante (invariante mientras no alteremos ninguno de sus componentes, imaginaos pronunciando una 'aaaaaaaa...').

Si llamamos a la respuesta de impulso del tracto vocal $h(t)$ la señal de voz finalmente emitida será:

$$v(t) \;=\; \sum_k \delta(t-kT) \;* h(t) \;=\; \sum_k h(t-kT)$$

Esto es, visto en el tiempo, un sonido vocalico debe consistir en la repetición del mismo patrón cada T segundos. La forma de ese patrón estará relacionada con la respuesta del tracto vocal, es decir, con como pongamos la lengua, boca, etc. al pronumciar dicha vocal. Comprobemoslo con la función `edit_audio` que nos permite echar un vistazo al aspecto de una señal unidimensional.

```
[v1 fs]=wavread('vocal_a.wav');
edit_audio(v1,fs);
```

La señal muestra los cinco sonidos vocalicos separados por silencios. Podemos hacer zoom in/out haciendo click con el ratón. Un click del botón izquierdo nos permite ampliar una zona, mientras que con un `Alt` click reducimos la zona mostrada. Pinchando en el menu `Mide` y posteriormente en dos puntos de la señal nos pinta la distancia (en msec) sobre el eje de tiempos.

Hacer zoom en la zona de la 'i' por ejemplo (tercer caso) y observar su aspecto. ¿Se aprecia la repetición de un patrón? Repetir para p.e. la 'a' y la 'u' ¿Qué sucede en este caso?.

Se trata ahora de que estimeis la distancia entre repeticiones (el parámetro T en la fórmula anterior). Para ello centraros en una vocal dada (p.e. la 'o') y hacer zoom hasta que en pantalla se vean cinco o seis repeticiones. Usando el menu Medir estimar la distancia entre dos pulsos repetidos (para reducir el error medir la distancia entre 4 o 5 y dividir). ¿Cuál es el valor de T obtenido? Repetir para otra vocal, p.e. la 'i'. ¿Se mantiene más o menos dicho valor de T?

Usando el menu `Carga` cargar la otra señal `vocal_s.wav`, que contiene la pronunciación de otro locutor. Estimar el valor de T para el nuevo locutor en las vocales 'o' y 'i'.

Consideremos ahora la situación en el dominio de frecuencias. Si $h(t)$ es la respuesta de impulso del tracto vocal y $H(\omega)$ es su correspondiente función de transferencia el espectro de la señal de voz será

$$\begin{array}{rcll} v(t) & = & \sum_k \delta(t-kT) \;* h(t) & = \quad \sum_k h(t-kT) \\ V(\omega) & = & \sum_k \delta(\omega-kW) \;\times H(\omega) & \quad \text{con } W = 2\pi/T \end{array}$$

Esto es, el espectro de un sonido vocálico se verá como una serie de pulsos separados por una frecuencia $W = 2\pi/T$ (inversa del T estimado antes) multiplicados por una cierta envolvente $H(\omega)$ (la función de transferencia de nuestro tracto vocal).

La separación entre pulsos $W = 2\pi/T$ o $F = 1/T$ Hz no depende de que vocal estemos pronumciando: nuestras cuerdas vocales vibran más o menos igual al decir 'a' o 'i'. Al parámetro F se le denomina tono o *pitch* de la voz.

Por el contrario lo que determina la diferencia entre una 'a' y una 'e' es fundamentalmente la posición que adoptan mis organos fonadores, es decir la configuración de mi tracto vocal, caracterizada por su función de transferencia $H(\omega)$.

Observemos las pronunciaciones de la secuencia 'aeiou' por dos locutores distintos usando nuestra herramienta para visualizar espectrogramas:

```
[v1 fs]=wavread('vocal_a.wav');    % Locutor 1
[v2 fs]=wavread('vocal_s.wav');    % Locutor 2
sound(v1,fs);
sound(v2,fs);                    % Si teneis sonido
figure(1); stft2dcol(v1,256,fs);
figure(2); stft2dcol(v2,256,fs); % Ventana de 256 muestras, unos 30 msec para esta fs
```

Observamos en ambas ventanas como se van sucediendo los cinco sonidos en el tiempo. Observar en uno u otro caso la presencia de franjas horizontales que corresponden a los pulsos de los que hemos estado hablando. ¿Cual de los dos locutores presenta el mayor espaciado entre franjas? ¿Quién tiene un tono de voz más agudo?

Notad que aunque por supuesto hay diferencias entre la 'a' o la 'i' de uno y otro locutor si que se aprecia un parecido en el contenido general de frecuencias (la "envolvente") para una misma vocal. Se trata ahora de estimar aproximadamente en el dominio de frecuencias el tono o *pitch* (F) en hertzios de uno y otro locutor (y ver si es coherente con nuestra estimación de T anterior).

Los pulsos se pueden apreciar mejor usando nuestra antigua versión unidimensional `stft(v1,fs)` con una escala lineal. Usar una ventana de Hanning y un tamaño grande de ventana (unos 60 msec) lo que permite ver los pulsos más definidos al aumentar la resolución en frecuencias. Posicionar el cursor de forma que la ventana caiga sobre una vocal (por ejemplo la 'i') y llamar a `ginput(4)` para pinchar con el raton en 4 pulsos sucesivos para determinar asi el "salto" entre pulsos.

	F1	F2	F3	F4
Frecuencia de pulsos				
$\Delta F = (F_k - F_{k-1})$				

Locutor 1: ΔF media estimada (tono) :

Repetir para la vocal 'o'.

	F1	F2	F3	F4
Frecuencia de pulsos				
$\Delta F = (F_k - F_{k-1})$				

¿Hay diferencias en el tono obtenido en uno y otro caso? ¿Es coherente con el espaciado T entre pulsos medido antes para las vocales consideradas?

Repetir con el segundo locutor usando `stft(v2,fs)` para determinar su tono en la vocal 'a'.

	F1	F2	F3	F4
Frecuencia de pulsos				
$\Delta F = (F_k - F_{k-1})$				

Locutor 2: ΔF media estimada (tono) :

De los dos locutores habia uno masculino y otro femenino. ¿Cual creeis que era la chica de los dos?

DECODIFICACIÓN FSK: simulación realista (entrega opcional)

La aplicación de la FFT para la decodificación de una señal FSK no se utiliza en la práctica. La FFT es demasiado "fina" para esta aplicación donde lo único que nos interesa es discriminar entre dos frecuencias f_0 (0 lógico) y f_1 (1 lógico).

Vamos a examinar en este apartado una implementación más realista (aunque no del todo) de un decodificador FSK. Los parámetros de la codificación son los siguientes:

```
fs=15000;            % frecuencia de muestreo (15 Khz)
f0=1200; f1=1800     % 0 lógico = 1200 Hz ,  1 lógico = 1800 Hz
T=1/300;             % Tamaño bit = 1/300  => 300 bits por segundo
```

¿Será posible en principio discriminar entre las dos frecuencias? ¿Por qué?

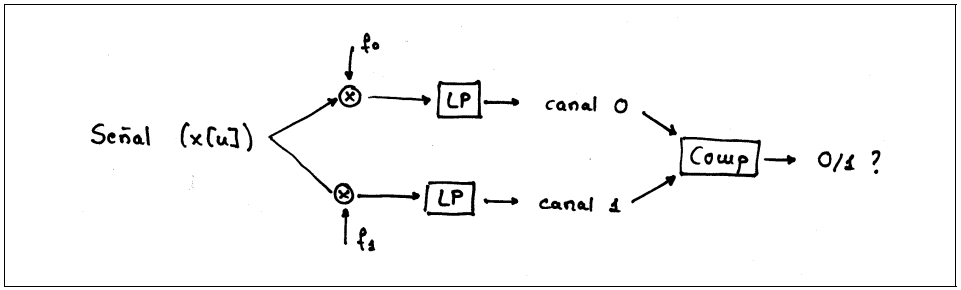
¿Cuántas muestras caben en la duración de un bit de datos? calcular dicho valor e introducirlo en una variable denominada `samples_bit`.

La señal codificada FSK se carga usando `load comm`. Se trata de un texto ASCII de unos 10-15 líneas. Su tamaño, duración y numero de bits en el mensaje serán:

```
load comm               % Cargamos mensaje binario codificado FSK
L=length(comm)          % numero muestras en comunicación
T_total=L/fs;           % duración comunicacion (en segundos)
nbits=round(T_total/T); % numero bits en el mensaje
```

Con un `sound(comm,fs)` podeis escuchar como sonaría nuestro comunicación en la línea telefónica. Visualizar las primeras 800 muestras de la señal FSK. ¿Podeis (visualmente) decodificar la primera letra (8 bits) del mensaje? ¿Cuál es?

El esquema que emplearía un decodificador FSK (simplificado) sería el siguiente:



La señal de entrada se procesa en dos canales, donde es multiplicada por unas frecuencias correspondientes a $f_0 = 1200$ y $f_1 = 1800$ Hz. Para simularlas en matlab haríamos:


```
t=[0:L-1]/fs;   % base de tiempos
c1 = cos(2*pi*f0*t); c2 = cos(2*pi*f1*t);
```

Sabiendo que si multiplicamos 2 frecuencias obtenemos las frecuencia diferencia y la frecuencia suma:

$$\cos(\alpha)\cos(\beta) = \frac{(\cos(\alpha-\beta)+\cos(\alpha+\beta))}{2}$$

indicar en la siguiente tabla que frecuencias apareceran en ambos canales cuando en la entrada tenemos un 0 (1200 Hz) o un 1 (1800 Hz):

Datos de Entrada	Frecuencias en Canal 1 (1200)	Frecuencias en Canal 2 (1800)
0 lógico (1200 hz)		
1 lógico (1800 hz)		

Usando la función `lp_200` en cada canal llevamos a cabo un filtrado pasobajo que elimina aquellas frecuencias por debajo de unos 200 Hz. Tras la aplicación de dicho filtro indicar en la siguiente tabla en que canal tendremos respuesta ante una entrada 0 ó 1:

Datos de Entrada	Respuesta Canal 1 (1200)	Respuesta Canal 2 (1800)	Respuesta discriminante
0 lógico (1200 hz)			
1 lógico (1800 hz)			

Verificar graficamente la deducción anterior mostrando simultaneamente ambos canales con `show2` (adjuntad gráfica):

```
canal_1 = lp_200(comm.*c1); canal_2 = lp_200(comm.*c2);
show2(canal_1,canal_2,fs);
```

Finalmente nuestro decodificador calcula una función discriminante simplemente restando la salida de ambos canales: `disc=canal_2 - canal_1;`

Indicar en la última columna de la gráfica anterior que signo podemos esperar en el discriminante en función del bit de entrada. Comprobarlo gráficamente con `show2(comm,disc,fs);` (adjuntad gráfica).

Con todo esto, para decodificar la señal solo tengo que muestrear la señal del discriminador cada T=1/300 segundos (cada `samples_bit` muestras) en el momento adecuado (centro del bit), ver si es positiva/negativa y declarar un 0/1 según corresponda. Con esto obtenemos un mensaje de 0/1 con una longitud de `nbits` que se convierte a texto usando `bin2txt`.

Implementar lo anterior usando no mas de tres líneas (razonablemente cortas) de *Matlab* (recordad como diezmabamos una señal en la practica anterior) y reproducir el mensaje descifrado.

Eye diagrams

Al contrario que en el ejemplo anterior, en la vida real no sabemos donde están las fronteras de los bits (nuestro receptor podría empezar a irecibir datos a mitad de un bit) por lo que tampoco sabríamos donde muestrear. Una herramienta usada por los ingenieros de comunicaciones para ver gráficamente el instante óptimo de muestreo (y otras muchas cosas) es el gráfico conocido como "eye diagram" (por razones que se harán evidentes al verlo).

Se trata de dividir nuestra señal discriminante `disc[]` en segmentos del tamaño de un bit (T=1/300 sec, `samples_bit` muestras) y pintarlos todos superpuestos. Aunque esto podría hacerse con un bucle la forma más sencilla es reordenar nuestra señal discriminante `disc[]` como una matriz de `samples_bit` filas y `nbits` columnas usando:

```
disc = reshape(disc,samples_bit,nbits)
```

De esta forma cada columna son las muestras del discriminante durante la duración de un bit y cada fila son muestras del discriminante separadas por la duración de un bit. Para pintar ahora el "eye diagram" a partir de la matriz `disc[,]` basta hacer: `plot(disc)`. La función plot aplicada a una matriz pinta todas las columnas de la matriz solapadas.

Adjuntar el diagrama obtenido. ¿Por qué se llama "eye diagram"?

Viendo el diagrama podemos ver qué momento es el óptimo para muestrear, aquel donde el ojo esté más abierto. ¿En que muestra correspondería en este caso?

Vamos a recrear el diagrama anterior para una nueva señal, obtenida añadiendo ruido a la señal FSK original de la siguiente forma:

```
randn('state',sum(100*clock()));          % Reseteamos semilla de aleatorios
N=0.3; comm_noise = comm + N*randn(1,L);  % Añadimos ruido normal gaussiano de varianza N=0.3
show2(comm,comm_noise,fs);                % Visualizamos
```

Adjuntar el "eye diagram" para diferentes niveles de ruido, `N=0.3`, `N=0.8`, `N=1.0`, `N=1.5`. ¿Qué se aprecia en los diagramas al aumentar el nivel de ruido? ¿Cuándo pensais que podemos empezar a encontrar bits mal identificados? ¿Por qué?

Decodificar (y adjuntar) la comunicación FSK decodificada para un ruido de N=1.0 y N=1.5. ¿Se aprecian errores en la decodificación?

Notad que con la nueva disposición de `disc` como matriz, tomar una muestra cada bits es tan simple como extraer una fila de dicha matriz.