MODULACIÓN Y DEMODULACIÓN IQ. APLICACIÓN A EMISORA FM

1) Modulación/Demodulación FM

Modulación/Demodulación de una señal "analógica"

Cargad la señal x con la que vamos a trabajar con **`load x_signal`**. Es una función en forma de sierra con una frecuencia de 5 Hz y duración de 1 segundo, muestreada con una frecuencia de muestreo fs=20000 Hz. Cread un vector columna de tiempos t con 20001 valores desde 0 a 1 separados por el intervalo de muestreo dt=1/fs. Adjuntar la gráfica de la señal x en función del tiempo t.

a) Modulación: para crear la señal FM usaremos modulación IQ. En primer lugar se generan las señales I y Q: I = A·cos(φ) y Q=A·sin(φ). En modulación FM la amplitud A es constante (usad A=1) y la fase φ es la integral de la señal x(t) multiplicada por una constante k:

$$\varphi(t) = k\int_0^t x(s)\cdot ds$$

Si la constante k (rads/seg) es alta la frecuencia instantánea se apartará más de la frecuencia de la portadora y la señal FM tendrá un mayor ancho de banda. En la banda de FM en España, la desviación con respecto a la portadora es de ± 75 KHz, por lo que cada emisora FM "ocupa" unos 150 KHz. Para evitar solapamientos, las emisoras están separadas por 200 KHz en el dial de FM.

La integral de una función x(s) puede aproximarse a partir de sus muestras x[n], usando la regla del rectángulo:

$$\varphi(t) = k\int_0^t x(s)\cdot ds \Rightarrow \varphi(n) \approx k\cdot dt\cdot\sum_{i=0}^{n-1} x[i]$$

donde dt=1/fs es el intervalo de muestreo. Para calcular el sumatorio de la fórmula anterior podemos usar la función cumsum(x) que suma de forma acumulativa los contenidos de la secuencia x[n]. Calculad la fase φ aplicando la fórmula anterior a la señal x. Usad k=2000 y recordad que en esta simulación la frecuencia de muestreo es fs=20000 Hz. Una vez determinada la fase φ calcular las señales I y Q. Haced la gráfica de φ, I y Q en función del tiempo t en la misma figura usando **`subplot(311); plot(t,fase); subplot(312); plot(t,I);`** etc. Usad **`xlim([0 0.2])`** para mostrar las 3 señales solo en el intervalo de t=0 a t=0.2. Adjuntad captura de la figura.

Una vez obtenidas I y Q se combinan con la oscilación de la portadora (cos y -sin) para generar la señal IQ. Para la portadora usaremos una frecuencia fc=2000 Hz. Con la misma base de tiempos t crear un coseno y un seno con la frecuencia de la portadora y usarlos para crear la señal modulada $IQ = I\cdot\cos(2\pi f_c t) - Q\cdot\sin(2\pi f_c t)$

Para comprobar que la señal IQ corresponde a la modulación FM de la señal x[], usaremos la función spectrogram(). Esta función muestra cómo van cambiando las frecuencias de una señal en el tiempo: **`figure; spectrogram(IQ,512,[],[],fs);`**
Adjuntad figura resultante. A partir de la gráfica, ¿cuál es el ancho de banda de la señal FM? Si quisiéramos enviar una señal similar en un canal adyacente, ¿qué frecuencia tendríamos que usar para la nueva portadora?

b) Demodulación: Vamos a proceder a demodular la señal IQ. Para ello, el primer paso es recuperar las señales I y Q por separado a partir de IQ. Vamos a escribir una función que reciba como argumento la señal modulada IQ y devuelve las señales I/Q por separado:

```
function [I,Q] = demodulador(IQ,f_local,ph0,fs)
% Parametros de entrada:
%   IQ : señal modulada
%   f_local:  frecuencia del oscilador local (en Hz)
%   ph0: fase inicial el oscilador local (rads)
%   fs:  frecuencia de muestreo (Hz) de simulación
% Parametros salida: I,Q: señales I y Q separadas.
```

Dentro de la función:

- Hallar (con length) el tamaño N de la señal de entrada y crear una base de tiempos t con el mismo tamaño usando fs como frecuencia de muestreo.

- Crear la oscilación local $\cos(2\pi \cdot f\_local \cdot t + \varphi 0)$ y $\sin(2\pi \cdot f\_local \cdot t + \varphi 0)$, usando la frecuencia local (f_local) y fase inicial ($\varphi 0$) dadas en los argumentos.

- Calcular IQ·cos y –IQ·sin, obteniendo las ramas I y Q del demodulador.

- Finalmente, para eliminar las frecuencias "suma" creadas al multiplicar las dos sinusoides filtraremos las señales I y Q con un filtro paso bajo. Para calcular el filtro usaremos la función **fir1(n,wc)** donde n es el orden del filtro y wc la frecuencia de corte normalizada (dividida por fs/2). En este caso usaremos un filtro de orden 64 (65 coeficientes) con una frecuencia de corte de 1000 Hz:

  ```
  wc=1000/(fs/2); B = fir1(64,wc);
  ```

  Una vez que tenemos los coeficientes B, usad la función filter() de MATLAB para filtrar I y Q. El resultado son las señales I y Q devueltas por la función.

Código de vuestra función una vez completada.

c) Aplicación: Aplicar esta función a la señal IQ obtenida en 1a), usando una frecuencia de 2000 Hz y una fase de 0.0 para el oscilador local (que coinciden exactamente con las usadas al modular):

```
[I2 Q2]=demodulador(IQ,2000,0.0,fs);
```

En la modulación FM, la información de la señal es la derivada de la fase (frecuencia instantánea) de un fasor complejo R cuya parte real e imaginaria son I y Q. Para construir dicho fasor podemos hacer:

```
R = complex(I2,Q2);
```

A partir de R extraemos su fase usando la función **angle(R).**La información de la señal x está en la **derivada** de la fase: para aproximarla podemos restar los valores consecutivos de la fase que acabamos de obtener. Esto lo podemos hacer con la función **diff()** de MATLAB. Haced un plot de la señal recuperada y adjuntar la figura resultante ¿Recuperamos la señal original?

El problema es que **angle** devuelve el valor de la fase siempre en el intervalo $[-\pi, \pi]$. Imaginad que en la muestra n la fase vale justo $-\pi$ y está decreciendo. En el instante siguiente la fase valdrá un poco menos, $(-\pi - \delta\varphi)$, pero la función angle() nos va a devolver su valor equivalente (trigonométricamente) en el intervalo $[-\pi, \pi]$ que sería $(\pi-\delta\varphi)$. Esto estropea la estimación de la derivada, ya que en vez de la diferencia $(-\pi - \delta\varphi) - (-\pi)$ que nos daría la estimación correcta $(-\delta\varphi)$ lo que obtenemos es $(\pi - \delta\varphi) - (-\pi) = 2\pi - \delta\varphi$, introduciendo un salto de 2π respecto al valor correcto.

Una posible forma de solucionar este problema es considerar dos valores sucesivos del fasor, R(n-1) y R(n), cada uno con su amplitud A y fase φ correspondientes:

$$R(n-1) = A(n-1)\cdot\exp(i\cdot\varphi(n-1)) \qquad R(n) = A(n)\cdot\exp(i\cdot\varphi(n))$$

Si multiplicamos R(n) por el conjugado de R(n-1):

$$R(n)\cdot R^*(n-1) = A(n)\cdot e^{i\cdot\varphi(n)}\cdot A(n-1)\cdot e^{-i\cdot\varphi(n-1)}$$
$$= A(n)\cdot A(n-1)\cdot e^{i\cdot(\varphi(n)-\varphi(n-1))} \rightarrow \begin{cases} \text{mod} = A(n)\cdot A(n-1) \\ fase = \varphi(n)-\varphi(n-1) \end{cases}$$

La fase del producto calculado es justo la diferencia de fases buscada (φ(n) - φ(n-1)) y no muestra los saltos de fase de antes. Sabiendo que la función de MATLAB para el conjugado es conj(), escribir el código necesario para recuperar la señal de esta forma (un par de líneas). Adjuntar el código usado y la figura con la señal obtenida (que debe seguir la forma de la señal original)

Una de las ventajas de la modulación FM para señales analógicas es que es muy estable frente a errores en la frecuencia o la fase del oscilador local. Demodulad la señal usando la frecuencia correcta (2000 Hz) pero con una fase de $\pi/2$. Repetid con una frecuencia de 2200 Hz y una fase de 0.0 (desajuste de frecuencia). Adjuntar la gráfica con las señales recuperadas en ambos casos junto con la obtenida con los parámetros de demodulación correcta. ¿Qué efecto tiene un desajuste de la fase del oscilador local? ¿Y de su frecuencia?

Demodulación de una señal binaria (BPSK):

Estudiaremos ahora el problema de la demodulación en comunicaciones de datos digitales. Usando la misma base de tiempos t (1 seg) anterior, creamos un array de 0's y 1's alternados (de duración 0.025 segundos) con **bits = mod(t,0.05)<0.025;**

Multiplicando ese array por π obtenemos la correspondiente fase φ a usar en un esquema de modulación binaria BPSK: 0 (si el bit=0) o π (si el bit=1).

Construir las señales en cuadratura I=cos(φ) y Q=sin(φ) y la señal IQ con la misma portadora de antes (fc=2000 Hz). Demodular luego la señal IQ con una frecuencia local (2000 Hz) y fase (0.0) en perfecto ajuste con las usadas al modular. Tras la demodulación, mostrad las primeras 5000 muestras de las señales I y Q. Adjuntar la gráfica ¿En qué canal podemos ver los bits de la señal?

Como hemos dicho, I y Q pueden verse como la parte real e imaginaria de un fasor que gira en el plano complejo. La función **`show_IQ(I,Q)`** nos muestra la evolución de dicho fasor en el tiempo, como si fuese un osciloscopio al que mandamos las señales I Y Q a sus ejes X e Y**.** Los cuadrados azules marcan los estados ideales donde esperamos ver los bits 0 y 1. Veréis como la señal salta entre los 2 estados.

Vamos a repetir lo que hicimos para la señal analógica, desajustando la frecuencia o la fase del oscilador local. En primer lugar demodular con la frecuencia correcta pero con un desajuste de $\pi/4$ en la fase. Observad las señales demoduladas usando **`show_IQ(I,Q);`** ¿Qué efecto tiene el desajuste de fase en el diagrama IQ?

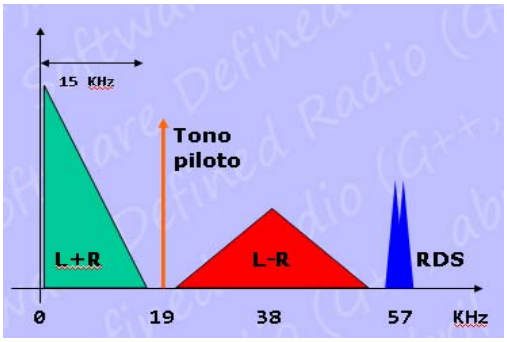
Probad sin el desajuste de fase pero con una frecuencia local ligeramente distinta (2001 Hz) de la original. Describid la evolución de la señal en el plano (I,Q). ¿Cómo afectará a la decodificación de bits?

¿Qué podemos concluir sobre lo crítico que es el correcto ajuste de fase y frecuencia entre emisor y receptor en este tipo de modulación comparada con la modulación FM de una señal analógica?

2) Decodificación de datos reales de una señal real (FM)

Una vez entendidos los conceptos básicos vamos a procesar algunos datos reales.

Como ejemplo usaremos la señal de una emisora FM. En su versión más simple (no estéreo) el proceso de emisión FM es idéntico al que se ha simulado en el 1er apartado de esta práctica. La señal de audio x(t) (limitada a 15 KHz de ancho de banda) se pasa a un modulador FM con una portadora en la banda de FM (89-106 MHz). Sin embargo, como se muestra en la figura adjunta en una señal FM suele haber más información.



Normalmente una emisora FM transmite audio en estéreo. En ese caso se calcula previamente la suma (L+R) y diferencia (L-R) de ambos canales:

- La suma (L+R) se deja tal cual (en el "centro" del canal). De esta forma se mantiene la compatibilidad con receptores no estéreo. Adicionalmente, se añade un tono piloto (frecuencia pura) de 19 KHz para indicar al receptor la presencia de datos en estéreo.

- La señal diferencia (L-R) se modula en amplitud AM con una subportadora de 38 KHz (2x19). Con un ancho de 15 KHz en la señal de audio no tenemos solapamiento entre la señal suma (L+R), el tono piloto (@19 KHz) y la señal diferencia (L-R) (@38 KHz).

- Adicionalmente muchas emisoras emiten un stream de datos (RDS o Radio Data Signal), en otra subportadora a 57 KHz (3x19). Es la componente que se muestra en azul en el gráfico anterior.

a) Decodificación del audio (mono)

En IQ_data_FM.mat tenéis los datos I y Q obtenidos a partir de un USB stick usado como una radio software sintonizado a una emisora de radio en la banda de FM (89-106 MHz). En este caso el sintonizador de la radio ya ha hecho la multiplicación por la portadora (cos/sin) y el filtrado paso-bajo para eliminar las frecuencias dobles. Las señales I y Q se muestrean posteriormente y se envían al PC. Podéis cargar estos datos haciendo:

```
load IQ_data_FM; I=double(I); Q=double(Q);
```

El ADC usado es de 8 bits por lo que los datos se guardaron como enteros (bytes), de ahí la necesidad de convertirlos a double para hacer los cálculos posteriores. En cuanto al muestreo, el ancho de banda de una emisión FM es de unos 100 KHz, por lo que para poder muestrear adecuadamente estas señales I/Q la frecuencia de muestreo debería ser superior a 200 KHz. En este caso se ha usado fs = 247.000 muestras/seg. ¿A cuántos segundos de señal corresponden las muestras cargadas?

Por lo tanto en este apartado disponemos de las señales I y Q ya demoduladas y separadas y nuestra única tarea será extraer la señal a partir de la fase de R, el correspondiente fasor complejo formado a partir de las señales I y Q. Tras recuperar la señal usar la función ver_tf() para visualizar el aspecto de la señal original (sin diezmar) en el dominio de frecuencias: `ver_tf(x,fs,'r','semi');`

Adjuntar gráfico (donde se deben ver las diferentes componentes de la señal FM).

En este apartado estamos sólo interesados en escuchar el audio, por lo que la señal demodulada se filtrará con un filtro pasobajo con una frecuencia de corte del orden de 15 KHz para eliminar todo lo que no sea el canal principal de audio. Para diseñar el filtro usaremos la función fir1() explicada antes. Obtener los coeficientes de un filtro paso-bajo de orden n=150 y frecuencia de corte=15000 Hz. Usando fvtool(), mostrad la función de transferencia del filtro diseñado. Adjuntar la figura con el módulo de la función de transferencia del filtro obtenido. Aplicar este filtro FIR a la señal x y adjuntad una captura en el dominio de frecuencias de la señal filtrada xf.

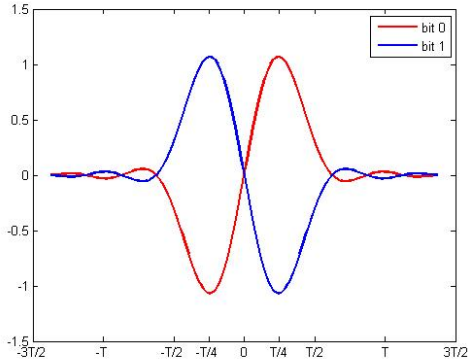
Para escuchar la señal resultante podríamos hacer `sound(xf,fs)` pero este comando puede dar problemas en algunos ordenadores porque la frecuencia de muestreo que estamos usando (fs=247000 Hz) es mucho más alta que las usadas habitualmente con señales de audio y puede que no esté soportada por nuestro sistema de audio.

Para reducir la frecuencia de muestreo, como la señal filtrada ya no tiene contenidos por encima de 15 KHz, podemos submuestrearla quedándonos con **una muestra de cada cinco**. Adjuntad el código para hacerlo. ¿Qué frecuencia de muestreo debéis usar en el comando sound() para escuchar la nueva señal submuestreada?

Por último debemos ajustar el "volumen" de la señal de audio antes de escucharla. La tarjeta de sonido espera valores entre -1 y 1 para las señales de audio. Valores muy pequeños hacen que la señal se escuche muy baja y valores por encima de ±1 provocan distorsiones. Indicad el valor máximo y mínimo de la señal de audio. Reescalar la señal para que su máximo sea 0.9. ¿Se escucha ahora correctamente el audio? Transcribid la primera frase del mensaje.

Decodificación de la señal RDS

En la señal demodulada x, además de los canales de audio, podíamos ver un pico en la frecuencia de 57000 Hz, usada para transmitir un stream de datos RDS (Radio Data System). Estos datos están modulados con un sistema llamado BiPhase Shift Keying Modulation, en el que cada bit (0/1) se codifica con uno de los siguientes símbolos (rojo/azul):



El símbolo del siguiente bit se colocaría en la posición T y el del bit anterior en -T. La señal resultante de la superposición de los bits que forman el mensaje a enviar se usa para modular en amplitud la subportadora de 57 KHz.

Respecto a la separación T, los bits se mandan a un ritmo de 1187.5 bits/segundo (57000/48), lo que corresponde a un periodo T~842 msec. En muestras, como la frecuencia de muestreo usada en nuestra señal es de 247000 muestras/segundo, un bit corresponde a T = 247000/1187.5 = 208 muestras.

Conocida T, según el gráfico anterior, para decidir si un bit es un 0 o un 1 debemos muestrear la señal en las posiciones +T/4 y –T/4 (respecto del centro del bit) y restar los valores obtenidos. Si la resta es positiva tenemos un bit 0 (símbolo rojo) y si es negativa un bit 1 (símbolo azul).

Una complicación adicional es que en estas comunicaciones se usa una codificación diferencial. Los bits no se mandan directamente: el bit enviado es un XOR entre el correspondiente bit del mensaje original y el bit previamente enviado. En el receptor para decodificar el mensaje original se hace un XOR entre los bits consecutivos que se van recibiendo: **`data(k) = rec(k) XOR rec(k-1)`**

La ventaja de esta codificación diferencial es que es irrelevante si durante el proceso de decodificación se intercambian 0's por 1's en los bits decodificados. En ambos casos el mensaje recuperado es el mismo. Esta propiedad es importante porque en algunos sistemas de demodulación los bits recuperados pueden estar invertidos. La codificación diferencial asegura que se recupere el mensaje original en esos casos.

El punto de partida para decodificar los bits del *stream* RDS es la señal x (la que contenía todas las componentes de la señal FM antes de aplicarle el filtro de 15 KHz para aislar el audio). El primer paso es aplicar un filtrado pasobanda, centrado en 57 KHz y con un ancho de banda estrecho, de ±2.5 KHz alrededor de los 57 KHz para asegurarnos que nos quedamos sólo con la señal RDS. Por lo tanto, lo que queremos es un filtro paso-banda con una banda de paso entre 57-2.5 y 57+2.5 KHz.

Para diseñar este filtro, usaremos de nuevo la función B=fir1(n,wc) ya explicada. La diferencia es que wc, en vez de ser una única frecuencia (la de corte) va a ser ahora un intervalo wc=[w1 w2], donde w1 y w2 corresponden a las frecuencias que delimitan la banda de paso. Como antes, las frecuencias w1 y w2 deben darse como frecuencias normalizadas (divididas por fs/2). Usar n=192 para el orden del filtro. Adjuntad código usado para generar el filtro y una captura de |H| usando la fvtool().

Filtrar la señal x con este filtro y llamad x_rds a la señal filtrada. Usando ver_tf() visualizar la señal resultante en el dominio de frecuencias. Debéis ver que solo se conserva el "pico" en 57 KHz. Adjuntad captura.

Haced de x_rds desde la muestra 2050 a la 3100. Adjuntad una captura MARCANDO sobre el gráfico unas líneas verticales dónde creéis están las fronteras entre los bits. ¿Cuál podría ser un criterio automático para detectar el centro de los bits?

Para "limpiar" los bits sólo hace falta eliminar la oscilación rápida (los 57000 Hz de la subportadora) que se aprecia en la gráfica anterior. Como siempre, para eliminar una frecuencia y bajar los bits a banda base, multiplicaremos por una oscilación de la misma frecuencia (57000 Hz).

Aquí lo ideal sería implementar algún tipo de algoritmo adaptativo (buscar en google "Costas filter") para detectar exactamente la frecuencia y fase de la portadora (ya que antes hemos visto lo crítico que podía ser en este tipo de modulación). Sin embargo, para no complicar la práctica, vamos a asumir que la frecuencia nominal (57000 Hz) es la correcta y que la fase inicial es 0. Luego veremos los problemas de este enfoque.

Cread un vector de tiempos t (vector **columna** del mismo tamaño que los datos x_rds) con un intervalo entre tiempos de 1/fs (el intervalo de muestreo) y cread una oscilación con una frecuencia de 57000 Hz usando la función cos(). Multiplicar esta oscilación local punto a punto por la señal x_rds. Usando ver_tf() mostrad el contenido en frecuencias de la nueva señal. Adjuntad captura.

Lo único que nos queda es aplicar un filtrado paso-bajo a la señal para eliminar la doble frecuencia (~ 57000 x 2 = 114 KHz) introducida por la multiplicación anterior y que se observaba en la gráfica anterior. La frecuencia de corte del filtro paso bajo a aplicar se corresponderá con el ancho de banda de la señal RDS (2500 Hz). Usad de nuevo la función fir1() como en los ejemplos anteriores para crear un filtro paso-bajo de orden n=192 con frecuencia de corte de 2500 Hz. Aplicad este filtro a la señal. Adjuntar vuestro código para crear el filtro y aplicarlo a la señal. Adjuntad una captura de la señal resultante en el mismo rango de antes, desde la muestra 2050 a la 3100.

Eye diagrams (determinación de la posición de los bits)

En un caso real no sabemos a-priori donde caen las fronteras de los bits, ya que el receptor puede haber empezado a recibir los datos en cualquier momento durante la duración de un bit. Conocer la posición de los bits es fundamental para saber en qué momento hay que muestrear dichos bits. Un gráfico que nos permite visualizar el instante óptimo de muestreo es el llamado “eye diagram”.

Se trata de dividir la señal con la que hemos terminado antes conteniendo los bits en segmentos del tamaño de un bit y pintarlos todos superpuestos. El ancho de un bit es de 208 muestras, por lo que la señal completa (2392000 muestras) contiene un total de 11500 bits (208 x 11500). Para pintar el eye-diagram podríamos hacer un bucle, pero es mejor convertir la señal (un vector de 2392000 elementos) en una matriz de 208 filas y 11500 columnas con la función reshape:

```
Nbits=11500; BIT_SIZE=208;  M=reshape(vector,BIT_SIZE,Nbits);
```

Cada columna de la nueva matriz M contiene las muestras de un bit. En MATLAB, si hacéis un plot de una matriz se hará un plot de TODAS las columnas (en este caso todos los bits) mostrándonos así el eye-diagram al completo con una sola orden. Adjuntad la figura resultante del plot de M.

Si la gráfica del "eye-diagram" muestra un "ojo" (o en este caso ojos) abierto, esto nos indica que hay una clara diferencia entre los niveles altos (1's) y bajos (0's) de la señal. Esto permitirá una decodificación sin errores. Si el "ojo" está "cerrado", se corre el peligro de no poder diferenciar entre ambos estados ¿Creéis que en este caso se podrán decodificar los bits o tendremos altas probabilidades de error?

En este caso el problema es que la frecuencia nominal usada (57000 Hz) no es del todo correcta. Repetir la demodulación con las siguientes frecuencias ligeramente modificadas para el oscilador local: **`f_local = 57000.5, 56999.5, 57001.0, 56999.0`** ¿Qué frecuencia da mejores resultados? Adjuntar eye-diagram para esa frecuencia. ¿Creéis que usando esta frecuencia podremos decodificar los bits sin errores?

Notad que incluso en el mejor caso parece haber un corrimiento de los bits (el cuello parece que se va desplazando), lo que indica que la frecuencia usada no es del todo correcta (o que la frecuencia/fase del emisor o del receptor tenían una ligera deriva con el tiempo). Esto indica la importancia de usar filtros adaptativos como los que hemos mencionados antes que son capaces de adaptar la fase/frecuencia local a la de la portadora.

Decodificación del mensaje RDS

Finalmente decodificaremos el mensaje contenido en esos 11500 bits. Lo primero es determinar el centro del bit a partir del eye diagram (usando la frecuencia que dio el mejor resultado). Observando el eye-diagram podéis determinar en qué muestra cae aproximadamente el centro de cada bit. ¿Cuál sería esa posición?

Una vez decidido el centro del bit y recordando la forma de codificar un bits, el paso siguiente es muestrear a ±T/4 de dicho centro. ¿A cuántas muestras N corresponde esa distancia de T/4? Se trata de muestrear a N muestras a la derecha y N a la izquierda del centro de cada bit. Si el muestreo en (centro+N) es mayor que el muestreo en (centro-N) se declara un 1 y en caso contrario un 1 (o al revés, ya que sabemos que una inversión de bits no importa en este sistema de comunicación).

Para hacer el muestreo usamos el hecho de que la matriz M está organizada en:

- 11500 columnas. En cada columna están las muestras de un bit.
- 208 filas = 208 posiciones de las muestras dentro de un bit.

Con esta disposición, muestrear los bits en una cierta posición de dentro de cada bit consiste simplemente en extraer la correspondiente fila de la matriz M. Escribir un par de líneas de código que:

- Extraigan muestras de la posición (centro + T/4) y (centro – T/4) de cada bit.
- Las comparen y decidan si los correspondientes bits son 0's o 1's.

Adjuntar vuestro código de muestreo y decisión de bits (1/0).

Los bits que hemos decodificado son los enviados, pero no son los del mensaje original, debido a la codificación diferencial usada en RDS. Para llegar al mensaje original debemos deshacer la codificación diferencial de la forma explicada antes. ¿Cuál sería el código a usar?

Por último, para comprobar que todo ha salido correctamente mandaremos dichos datos a un parser RDS para ver si son decodificados correctamente. He escrito un sencillo programa rds_decode que recibe un array de 1's y 0's como el que habéis decodificado y vuelca por pantalla los principales mensajes. Aplicarlo. ¿Cuál es el nombre y la frecuencia de la emisora FM?

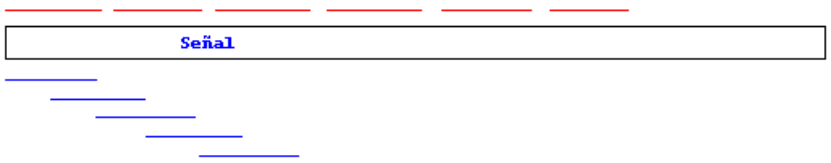
3) Espectrograma o transformada Fourier local

Para conocer el contenido en frecuencias de una señal podemos calcular su DFT (usando el algoritmo rápido de la FFT) y representar p.e. su módulo para ver la importancia relativa de las diferentes frecuencias que forman la señal.

Sin embargo, al analizar una señal de larga duración (o que nos está llegando en tiempo real) no es conveniente esperar a que la señal termine para calcular la FFT de toda la señal. Lo que se suele hacer es ir analizando bloques de muestras según van llegando. A esta técnica se le denomina análisis local de Fourier (ya que el análisis de Fourier se hace de forma local, bloque por bloque, en vez de globalmente).

Esta técnica es útil para ver la evolución de las frecuencias de una señal en el tiempo y es la base de la función spectrogram() de MATLAB que se usó en el primer apartado de la práctica. También se usa en el software que vimos para una SDR para visualizar la actividad en el tiempo de las diferentes frecuencias que estamos escuchando. En este apartado vamos a reproducir su funcionamiento.

El concepto es simple. Se trata de decidir el tamaño N (en muestras) de los bloques usados en el análisis y hacer la FFT de cada bloque.



El caso de los segmentos en rojo correspondería a dividir la señal en segmentos sin solapamiento. Lo normal al analizar una señal por bloques es usar un cierto grado de solapamiento entre bloques, como se muestra en el caso de los segmentos azules.

Se use o no solapamiento, las FFT's de todos los bloques se van organizando en una imagen, donde cada columna es el módulo de la FFT de un bloque. De esta forma moviéndonos de arriba abajo en la imagen nos movemos dentro de la FFT de un bloque, examinando las diferentes frecuencias presentes en esa franja de tiempo. Si por el contrario nos movemos de izquierda a derecha lo que hacemos es cambiar de bloque, lo que corresponde a movernos en el tiempo (cada bloque corresponde a un "trozo" de la señal en distintos instantes de tiempo). La imagen resultante nos da una representación (espectrograma) que nos permite visualizar para una cierta señal la evolución de sus frecuencias con el tiempo

Veamos cómo podemos construir esta imagen o "espectrograma" a partir de la señal IQ del apartado 1a) para intentar reproducir la imagen obtenida entonces con la función de MATLAB. Lo primero es decidir:

1. El tamaño N de los bloque que vamos a usar: esto determinará la resolución en frecuencias de las FFTs que como sabemos corresponden a fk = k·fs/N. Recordad que el nº de frecuencias no redundantes en una FFT de tamaño N es de N/2, que corresponden a las frecuencias cubriendo desde 0 hasta fs/2.

2. Decidir un salto S entre bloques. Típicamente se usa S=N/2 o S=N/4. El salto S determina la resolución en el tiempo, ya que cada bloque corresponde a un trozo de la señal que empieza en el tiempo tn = (n·S)/fs

Una vez decididos los valores de N y S el proceso es el siguiente:

1. Inicializar XX =[]. XX será la matriz o imagen donde iremos guardando los resultados.

2. Inicializar rg=(1:N) que corresponde a las muestras del 1er bloque a procesar

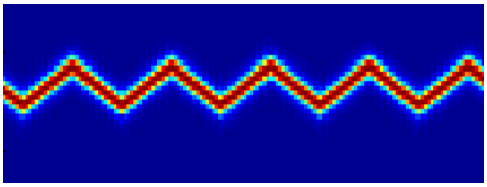
3. Procesar cada bloque hasta "acabar" con la señal. Para ello haced un bucle while mientras que rg(end) sea menor que la longitud de la señal IQ. En cada paso del bucle:

   a) Extraer el correspondiente segmento **`IQ(rg)`** **y** hacer su FFT (X). Opcionalmente podéis ventanear cada segmento (antes de calcular su FFT) multiplicándolo con la ventana creada con **`w=hanning(N)`**.
   b) Guardar X como una columna adicional de XX haciendo **`XX=[XX X];`**
   c) Incrementar rg por el salto S para procesar el siguiente segmento en la próxima iteración

Correr este bucle usando un tamaño de segmento N=256 y un salto S=N/2=128.
Al terminar tendremos una matriz XX de alto 256 (el tamaño N de los bloques) y de ancho 155 (el número de bloques procesados). Cada columna es la FFT de los 155 segmentos analizados.

Calcular el módulo de esta matriz y quedaros solo con las filas que corresponden a las frecuencias no redundantes. Recordad que aunque las FFT tienen un tamaño de N (el alto de nuestra matriz) la mitad de esos coeficientes son redundantes ya que al ser la señal real se verifica que |X[k]|=|X*[N-k]|. Adjuntad vuestro código para crear la matriz XX, calculad su módulo y eliminad las filas redundantes.

Tras eliminar las frecuencias redundantes podemos visualizar el resultado con **`imagesc(log(1+XX))`**. Adjuntad la gráfica resultante, que debe ser similar a la adjunta (el uso del logaritmo es simplemente para mejorar la visualización).



Lo único que le falta a la gráfica anterior es mostrar correctamente las unidades en ambos ejes. Ahora las unidades son sólo índices: en la vertical tenemos los índices k correspondientes a las diferentes frecuencias de la FFT y en el eje horizontal tenemos los índices de los diferentes segmentos. Repetir el comando anterior pero ahora haciendo:

```
imagesc(log(1+XX),'Ydata',Fk,'Xdata',Tn)
```

donde Fk y Tn son dos vectores con las frecuencias (en Hz) correspondiente a cada índice k y los y tiempos (en segundos) correspondientes al inicio de cada segmento. Podéis completar la gráfica usando los comandos **`xlabel('xxxx')`** e **`ylabel('yyyy')`** para etiquetar los ejes, indicando cuál corresponde al tiempo y cuál a la frecuencia.

Adjuntad el código usado para crear los vectores Fk y Tn y adjuntad la gráfica final, que debería mostrar la señal FM centrada en los 2000 Hz de la frecuencia portadora. La única diferencia con el gráfico de la función de MATLAB spectrogram() es que los ejes de la figura están cambiados. En nuestro gráfico el tiempo está en el eje horizontal y las frecuencias en la vertical.

4) Método alternativo para estimar derivada de la fase

Una vez que tenemos las señales I y Q demoduladas, otra forma de hallar la derivada de la fase es usar la relación:

$$\frac{d\varphi}{dt} = I\frac{dQ}{dt} - Q\frac{dI}{dt}$$

Comprobad esta relación repitiendo el ejercicio 1c) usando la fórmula anterior para estimar la derivada de la fase (que es la señal a recuperar). Podemos aproximar la derivadas dI/dt de forma numérica haciendo:

- (I_2-I_1) para el punto inicial
- $(I_{n+1}-I_{n-1})/2$ para los puntos intermedios n=2,3, ...
- (I_N-I_{N-1}) para el último punto.

Haced lo mismo para dQ/dt. Una vez estimadas las derivadas de I y Q usando este método aplicad la fórmula anterior para estimar $d\varphi$. Adjuntad el código usado y la gráfica de $d\varphi$, que debe ser similar a la señal x original (en forma de sierra).

Sabiendo que $I=\cos(\varphi)$ y $Q=\sin(\varphi)$, demostrar la relación anterior. Adjuntad vuestra deducción.