



PROCESADO DIGITAL DE SEÑAL

PROYECTO ESPECÍFICO - LABORATORIO 2 (SSo)

Señales Sonido

OBJETIVOS

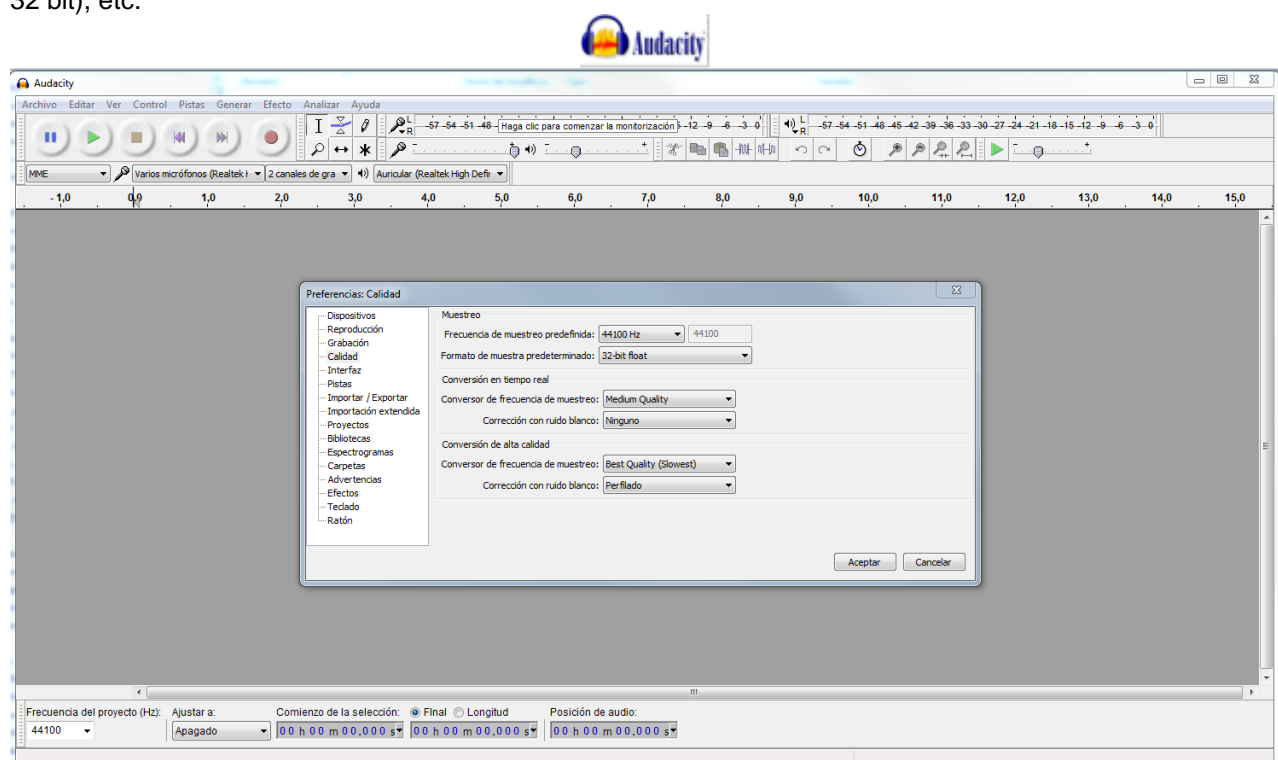
Familiarizarse con el manejo del sonido mediante MATLAB y en concreto:

- Captura, visualización y reproducción del sonido.
- Almacenamiento en ficheros y carga.
- Síntesis de sonidos y percepción.
- Estimación de frecuencia fundamental mediante correlación.

CAPTURA, VISUALIZACIÓN Y REPRODUCCIÓN DE SONIDO EN MATLAB

En las últimas versiones de MATLAB, si tenemos instalada una tarjeta de sonido y un micrófono se pueden grabar y reproducir sonidos utilizando las funciones **record**, **audiorecorder**, **player** y **audioplayer**. Para ello Matlab crea unos objetos, llamados **audiorecorder**, y **audioplayer**, con unos parámetros predefinidos de **fs** (frecuencia de muestreo, por defecto 8000Hz) y de **nbits** (nº de bits, por defecto 8). Estos parámetros pueden modificarse en función de las necesidades de una aplicación concreta.

Sin embargo, para realizar la grabación podemos utilizar cualquier tipo de software específico como, por ejemplo, Audacity. Este software permite exportar los sonidos grabados a formatos estándares como wav, mp3, etc.; con diferentes niveles de calidad en función de los parámetros seleccionados. Por ejemplo, frecuencia de muestreo (8000, 11025, 22050, 44100, 48000, ó 96000 Hz), formato de la muestra (16, 24, ó 32 bit), etc.



Tras realizar la grabación y la exportación (por ejemplo, a formato .wav) mediante Audacity o similar, podemos leer desde Matlab el fichero de sonido utilizando la función `audioread` y reproducirlo de la siguiente manera:

```
[frase, fs] = audioread('aufrase.wav');

fs
figure, plot(frase)

%player=audioplayer(frase, fs);
%play(player)

soundsc(frase, fs);
```

En relación con este último detalle, aunque en la introducción dijimos que MATLAB utilizaba un tipo de datos único, esto no es cierto. MATLAB puede utilizar diferentes tipos de datos para almacenar los sonidos, según las necesidades. Por defecto utiliza el tipo `double`, pero admite otros tipos porque es conveniente en algunas situaciones. Como las señales de sonido pueden ser grandes, puede ser conveniente especificar el tipo adecuado (`uint8` para trabajar con 8 bits, `int16` con 16 bits). Por defecto se usan 16 bits y se guarda utilizando el rango de valores $[-1 \ 1]$. Un inconveniente importante de los tipos `uint8` e `int16`, en función de la versión de MATLAB utilizada, es que **no admiten operaciones aritméticas**.

Data Type	Sample Value Range
<code>int8</code>	-128 to 127
<code>uint8</code>	0 to 255
<code>int16</code>	-32768 to 32767
<code>single</code>	-1 to 1
<code>double</code>	-1 to 1

Visualización

Como haríamos con cualquier otro tipo de señal unidimensional, hemos visualizado `frase` con la función `plot()`. Cuando trabajamos con sonido, como se trata de una señal analógica digitalizada, al visualizarla puede ser interesante utilizar para referenciar las muestras un vector de tiempos en lugar de un vector de índices. Para ello basta conocer la frecuencia de muestreo `fs` con que se ha digitalizado la señal. El tiempo entre muestra y muestra será $T_s = 1/fs$. Por ejemplo, para dibujar la señal capturada en función del tiempo, bastaría hacer:

```
N=length(aufrase); %nº de elementos de la señal frase
nn = 0:(N-1);
fs = 44100;
tt = nn/fs;
plot(tt, frase)
```

Como sabemos, el comando `plot` genera automáticamente los ejes del gráfico adaptados a la señal completa. Si queremos ver ampliado un fragmento de la señal, puede hacerse el **zoom** utilizando los botones que aparecen en la barra de menús de la figura.



También, podemos especificar los ejes mediante el comando `axis`, pasándole como argumento un vector con los valores `xmin`, `xmax`, `ymin` e `ymax`.

```
axis([1 1.5 -20000 20000])
```

Si queremos volver a los ejes iniciales, basta hacer:

```
axis auto
```

Por último, recordar que en ocasiones conviene mostrar claramente las muestras sin dejar que `plot` una los puntos de forma automática. Por ejemplo:

```
plot(tt, frase, 'ro')  
  
plot(tt, frase, 'b--', tt, frase, 'ro')
```

En las últimas versiones de MATLAB, la posibilidad de etiquetar, dimensionar, elegir los colores, etc...de las figuras que se representan puede seleccionarse desde la misma ventana gráfica mediante submenús que se activan en dicha ventana.

Reproducción

Para reproducir el sonido se pueden utilizar distintas funciones de MATLAB como ya se ha visto pero una forma simple es utilizar la denominada `soundsc` con el nombre del vector a reproducir y la frecuencia de muestreo que se utilizó en la digitalización como argumentos (proviene de versiones anteriores y **sólo funciona con el tipo de datos `double`**).

ALMACENAMIENTO EN FICHEROS Y CARGA

Al igual que cualquier variable o conjunto de variables, en MATLAB los vectores de sonido se pueden almacenar en ficheros (`.mat`) mediante la función `save` y leerse posteriormente mediante `load`.

Por ejemplo:

<code>save frase frase fs;</code>	Guarda <code>frase</code> y <code>fs</code> en el fichero <code>frase.mat</code>
<code>clear</code>	borra todas las variables del entorno
<code>load frase</code>	Carga el contenido del fichero <code>frase.mat</code>

Estos comandos ofrecen también la opción de trabajar con ficheros `-ascii` que podrían provenir de otra aplicación. Como ejemplo, el fichero `whale.dat` contiene en formato `ascii` (se puede leer con un editor de texto) un fragmento del sonido emitido por una ballena (11542 muestras a 22050 Hz)

```
load whale.dat      %Carga el contenido whale.dat en la variable whale  
plot(whale)  
soundsc(whale, 22050); % Normaliza al rango (-1,1)
```

También podemos cargar ficheros de sonido tipo `.mat`, como por ejemplo `gong.mat`, o `handel.mat`. Leedlos y escuchad el sonido asociado:

```
load gong.mat  
%Carga el contenido gong.mat en la variable y siendo Fs frec. muestreo  
soundsc(y,Fs)  
  
load handel.mat  
%Carga el contenido handel.mat en la variable y siendo Fs frec. muestreo  
soundsc(y,Fs)
```

Por último, MATLAB permite leer y escribir ficheros de sonido en formato `.wav` que es muy común en Windows. Las funciones son `audiowrite` y `audioread` (`wavwrite` y `wavread` en anteriores versiones de MatLab). En los ficheros `.wav` además de las muestras de la señal de sonido, se almacena la frecuencia de muestreo a la que ha sido realizada la digitalización. En el siguiente ejemplo, se lee un fichero que contiene una melodía, se representa gráficamente y se escucha:

```
[melo, fs] = audioread('melodia.wav');  
fs                                     %frecuencia de muestreo (11025)  
plot(melo);  
%crea un objeto de audio para poder ser escuchado  
%a una frecuencia de muestreo fs  
player=audioplayer(melo,fs);  
play(player);  
% también se puede utilizar el commando soundsc o sound  
soundsc(melo,fs);
```

De nuevo el valor por defecto de `fs` es 8000Hz.

SÍNTESIS DE SONIDOS Y PERCEPCIÓN

La superposición de sonidos es lineal. Esto quiere decir que basta sumar señales para combinar varios sonidos. **Prueba a sumar las señales frase y melo y escuchar el resultado.** Recordad que en MATLAB para sumar señales tienen que ser de la **misma longitud**, y, por tanto, tendrás que ampliar la más corta con ceros para poder sumarlas. Además, las dos deben ser de tipo `double`. Una vez sumadas las señales ya no pueden separarse.

Al superponer sonidos o al generarlos artificialmente hay que tener cuidado de que los valores obtenidos estén dentro del rango correcto (`[-1 1]` para el tipo `double`) porque si no `soundsc` truncará estos valores y deformará el sonido.

Vamos ahora a **generar sonidos artificiales** para experimentar como percibimos los humanos los sonidos. Debemos tener en cuenta que, para ello, vamos a asignar valores a las muestras, como si estuviéramos digitalizando una señal analógica que luego la tarjeta de sonido se encargará de reproducir. Por tanto, deberemos tener en cuenta la frecuencia de muestreo que queremos utilizar.

Sinusoides puros

Por ejemplo, para generar una senoide de 440 Hz que dure 1 segundo, contando con que vamos a reproducir el sonido a 22.050 Hz, bastaría hacer lo siguiente:

```
fs = 22050;
tt = 0:1/fs:1;
s = sin(2*pi*440*tt);
plot(tt(1:500),s(1:500))
soundsc(s,22050)
```

%Vector con instantes de muestreo
%Señal sinusoidal con f=440 Hz y fs=22050Hz
%Dibujamos un trozo.

Para hacer más pruebas de forma cómoda, vamos a escribir una función muy sencilla que hace algo parecido. Para ello, en la carpeta de alumnos se dispone de una función guardada en el fichero **tone.m**

```
function y = tone(f, phi, d, fs)
%TONO Genera sonido sinusoidal puro
% y = tone(f, phi, d, fs) genera muestras de una señal sinusoidal de
% frecuencia f en hercios, desfase phi en radianes, duración d en
% segundos y muestreada con frecuencia fs.
%
% tone(f, phi, d, fs) sin argumento de salida hace sonar la señal

tt = 0:1/fs:d;
s = cos(2*pi*f*tt + phi);

if nargin == 0,
    soundsc(s, fs)
else
    y = s;
end
```

Ahora ejecuta:

```
help tone
tone(440, 0, 1, 22050)
```

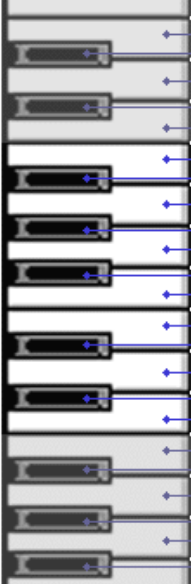
Escucha señales correspondientes a diferentes frecuencias. Por ejemplo: 50, 100, 500, 1000, 2000, ... Hz.

Podrás apreciar varias cosas:

- El tono o altura percibido depende de la frecuencias de la onda.
- Dependiendo de la frecuencia se percibe diferente volumen a pesar de que las señales tienen la misma amplitud (se supone que la sensibilidad máxima se da entre 1 kHz y 5 kHz).
- A mayor frecuencia los incrementos de frecuencia deben ser mayores si queremos percibir subidas de tonos similares.

Recuerda que si la frecuencia de la señal generada aumenta y se acerca al valor crítico de **fs/2** ocurrirá un efecto de **aliasing** y la frecuencia aparente bajará en lugar de subir.

En música las notas de la escala corresponden a frecuencias concretas, (por ejemplo 440 Hz corresponde a la nota LA). En la figura puedes ver parte de la escala.



E	Mi	659.26
D# (Eb)	Re # (Mi b)	622.25
D	Re	587.33
C# (Db)	Do # (Re b)	554.37
C	Do	523.25
B	Si	493.88
A# (Bb)	La # (Si b)	466.16
A	La	440.00
G# (Ab)	Sol # (La b)	415.30
G	Sol	392.00
F# (Gb)	Fa # (Sol b)	369.99
F	Fa	349.23
E	Mi	329.63
D# (Eb)	Re # (Mi b)	311.13
D	Re	293.66
C# (Db)	Do # (Re b)	277.18
C	Do	261.63
B	Si	246.94
A# (Bb)	La # (Si b)	233.08
A	La	220.00
G# (Ab)	Sol # (La b)	207.65
G	Sol	196.00
F# (Gb)	Fa # (Sol b)	185.00

Lee el código de los ficheros `e1.m`, `e2.m`, `e3.m` y `e4.m` que encontrarás en el servidor, ejecútalos y veras lo fácil que es generar música.

Ejemplo2: http://www.youtube.com/watch?feature=player_detailpage&v=dprwlAft16c#t=180

Ejemplo3: <https://www.youtube.com/watch?v=96KHBFDdEiA&app=desktop>

Combinaciones de armónicos:

En `e5.m` se generan, para la frecuencia fundamental $f_0=440$ Hz, los armónicos **s1, s2, s3, s4, s5**, es decir sinusoidales de frecuencias 440 Hz, 880 Hz, 1320 Hz, ... y se escuchan por separado y sumadas con diferentes pesos. Podrás observar que la señal combinada es periódica, con el **mismo periodo fundamental** que s1 y al escucharla se percibe como un solo sonido con el mismo tono que s1 (f_0) pero con distinto **timbre**. En el fichero `e6.m` tienes otro ejemplo de señales formadas por combinación de armónicos.

De todos modos, esto es demasiado simplificar. La percepción de la frecuencia fundamental (*pitch*) de un sonido periódico es muy subjetiva. Normalmente corresponde a la frecuencia con más peso en la onda, pero hay casos en los que puede no ser así, y además el oyente puede percibir más de un tono dependiendo del contexto y del entrenamiento.

Señales no periódicas:

Genera una señal no periódica, por ejemplo un ruido aleatorio, con la función `rand` y escúchala.


Por tanto, podemos decir que la percepción del timbre depende de la forma de la señal, pero es más correcto decir que depende de los armónicos que la forman. Para comprobarlo, escribe un programa que dibuje cuatro periodos de manera adecuada y haga sonar las dos señales siguientes:

$$s_1(t) = 0.5 \cos(2\pi 220 t) + 0.5 \cos(2\pi 440 t)$$

$$s_2(t) = 0.5 \cos(2\pi 220 t) + 0.5 \cos(2\pi 440 t + \pi/2)$$

Puedes ver que en $s_2(t)$ aparece un desfase en uno de los armónicos. Esta pequeña diferencia hace que su forma sea bastante diferente. Pero ¿suenan diferentes?

P2_1_armonicos.m



ESTIMACIÓN DE FRECUENCIA FUNDAMENTAL MEDIANTE CORRELACIÓN

En el apartado anterior hemos realizado síntesis de señales de sonido y ahora vamos a intentar lo contrario, el **análisis** de una señal para determinar algunas de sus características.

Hemos visto que en señales cuasiperiódicas, el tono percibido parece corresponder a la frecuencia o periodo fundamental de la señal. Este tono es muy importante en música, porque a cada nota le corresponde una frecuencia determinada.

Para comprobar esto, vamos a utilizar el sonido cargado antes del fichero **melodia.wav**, que contiene un fragmento de una melodía interpretada al piano.

Primero visualiza la señal total. Observando la amplitud de la señal, apreciarás claramente los instantes en que se pulsan las teclas. Con la ayuda del zoom analiza fragmentos de señal más pequeños (hazlo marcando las muestras). **Podrías medir (contando muestras) el periodo correspondiente a cada nota.**

Por ejemplo:

Si visualizas un fragmento de la **primera nota**, veras que el periodo es aproximadamente **27** muestras. Como la frecuencia de muestreo es de 11.025 Hz: ¿Cuál es la **frecuencia fundamental** de esa nota? ¿A qué nota podría corresponder?

Repite el procedimiento para otras notas (por ejemplo, la segunda y la cuarta).

P2_2: No requiere de fichero .m

FRECUENCIA FUNDAMENTAL Y AUTOCORRELACIÓN

Vamos a realizar ahora este cálculo de la frecuencia fundamental pero usando las propiedades de la **correlación**. Para ello se escribirá un programa en MATLAB que estime la evolución de la frecuencia fundamental del sonido utilizando la correlación para **obtener el periodo fundamental de fragmentos** de la señal.

Como se ha visto en clase, el comando **xcorr(x,y)** de MATLAB permite calcular la correlación de dos señales x e y:

$$\bullet \text{ Correlación: } r_{xy}(n) = \sum_{k=-\infty}^{+\infty} x(k)y(k-n) \quad \text{xcorr(x,y)}$$

Mientras que el comando **xcorr(s)** permite calcular la **autocorrelación** de una señal **s**. En el caso de que la señal **s** sea cuasiperiódica podemos obtener, a partir del resultado, una estimación del periodo fundamental y, por tanto, de la frecuencia.

Si queremos utilizar esta característica para obtener la evolución de la frecuencia fundamental en el tiempo (esto es, la melodía) de una grabación musical, tenemos que tener en cuenta que debemos ir aplicando la autocorrelación a fragmentos de la señal suficientemente pequeños, para que la frecuencia no varíe significativamente. Ahora bien, ¿entre qué valores se encontrará el periodo de las notas que estamos calculando? Por simplificar, vamos a suponer que sabemos que la melodía sólo contiene notas de las **tres escalas centrales** del teclado. Eso supone un rango de frecuencias aproximado de: **125 Hz – 1000 Hz**.

Podemos calcular el rango de periodos mediante la relación $T=N \cdot T_s \rightarrow N = f_s/f$ con $f_s = 11025$ m/sg,

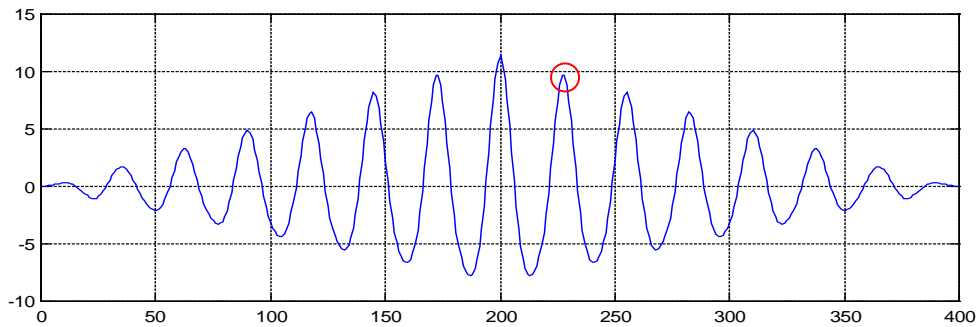
$$\begin{aligned} f = 125 \text{ Hz} & \rightarrow N = 11025/125 \approx 88 \text{ muestras} \\ f = 1000 \text{ Hz} & \rightarrow N = 11025/1000 \approx 11 \text{ muestras} \end{aligned}$$

Para detectar el periodo mediante correlación, debemos incluir en el fragmento por lo menos **dos periodos** de la onda. Por tanto, una buena elección del tamaño del fragmento podría ser **200** ($2 \cdot 100$ muestras).

Ejemplo:

En la figura puedes ver el resultado de aplicar la autocorrelación a un fragmento de 200 puntos de la señal **melo** escogido al azar entre las muestras 3000 y 3199.

```
Rxx = xcorr(melo(3000:3199)); plot(Rxx), grid
```

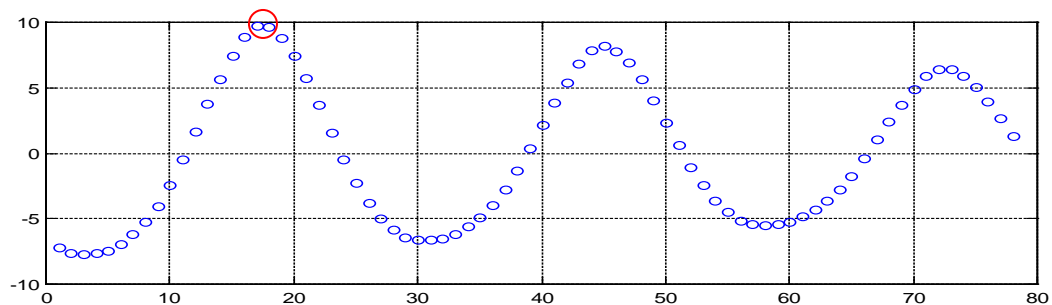


Observando esta figura se tiene que:

- La autocorrelación produce un vector de 399 elementos ($2 \cdot 200 - 1$)
- La posición $n=0$ de la señal $r_{xx}(n)$ corresponde al elemento 200 del vector resultado.
- La autocorrelación es par, es decir simétrica respecto del eje vertical.
- Se observa claramente que en el fragmento existe periodicidad. Para determinar el periodo fundamental podríamos medir la **distancia entre el pico de máxima altura y el siguiente (círculo)**.

Como en este caso el periodo está acotado, para medirlo podemos analizar sólo la zona que nos interesa (de $n=11$ a $n=88$).

```
plot(Rxx(211:288),'o'), grid
```



Puede verse que el periodo es aproximadamente de 27 muestras (máximo en el elemento 17 del gráfico, pero debemos sumarle 10 del desplazamiento realizado en la representación).

Para hacer este cálculo podemos utilizar la función `max` de MATLAB que nos da el valor máximo de un vector y su posición:

```
[rmax imax] = max(Rxx(211:288))
N0 = imax + 10 %posición del máximo
```

Para calcular la frecuencia correspondiente basta multiplicar por el periodo de muestreo ($T_s=1/f_s$) y calcular su inversa o bien hacer

```
f0 = fs/N0 %frecuencia asociada siendo N0 el periodo medio en muestras.
```

En este caso:

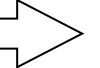
```
f0 = 11025/N0 %frecuencia asociada
```

Visto esto, lo que se pretende que hagas en esta práctica es un programa que muestre la utilización de esta técnica y os permita comprobar su utilidad para obtener la melodía de una música grabada y también sus limitaciones.

El programa tiene que realizar las siguientes tareas:

- Leer del fichero la señal de sonido 'melodia.wav'.
- Dividir la señal de sonido en fragmentos y para cada uno de ellos utilizar la autocorrelación para determinar el periodo y frecuencia fundamentales.
- Dibujar la evolución de frecuencias, que corresponde a la melodía.
- Generar una señal de sonido mediante una sinusoidal de la misma frecuencia para cada fragmento. Se obtendrá una señal con la misma melodía que la original.

Como guía puedes utilizar el fichero `pitch_xcor.m` como punto de partida y completarlo basándote en lo que acabamos de exponer.

`P2_3_pitch_xcor.m` 

Verás que el resultado es bastante aceptable, a pesar de que los cálculos se han hecho de la manera más sencilla posible.

Podrían añadirse algunas mejoras:

- Calcular el periodo promediando la separación de varios picos de la autocorrelación.
- Comprobar que el pico detectado en la autocorrelación es suficientemente alto, para distinguir los momentos en que la señal no es periódica (ausencia de notas).
- Filtrar el resultado para evitar oscilaciones.
- Incluir la frecuencia de muestreo como parámetro, para poder manejar otros valores. Esto influiría en Nmin y Nmax.

De todos modos, se ha obtenido un resultado aceptable porque la señal está formada por notas aisladas y analizando la señal en el dominio del tiempo puede encontrarse periodicidad.

Para comprobar que esto es así, carga la señal del fichero `melodia2.wav`, escúchala, visualiza un fragmento y aplícale el programa `pitch_xcor`. Verás que en este caso el método no sirve para nada.