Trabajo Práctico Especial de Señales y Sistemas

Identificación de canciones mediante huellas digitales acústicas

Introducción

Imaginen la siguiente situación: se encuentran en un bar, tomando su trago favorito, inmersos en esa interesante conversación cuando de repente . . . “pará, pará, ¿qué es eso que suena?”. Entre el ruido general alcanzan a distinguir esa canción que no escuchaban en tanto tiempo, que habían obsesivamente intentado encontrar pero que, a pesar de ello, nunca llegaron a dar siquiera con el nombre del intérprete. Su corazón se estremece . . . Ahora que la tienen ahí, sonando nuevamente, la situación es bien diferente a la de hace algunos años: toman su teléfono celular, abren alguna de las aplicaciones de reconocimiento de audio que tienen y, en cuestión de segundos, aparece la información completa del tema, la banda, ¡e incluso se atreve a sugerir artistas similares!

Si el protagonista de esta ficción fuera estudiante de Señales y Sistemas, no debería extrañarnos que surjan en él varios interrogantes: ¿cómo hizo el programa para reconocer la canción escuchando sólo 10 segundos?, ¿cómo puede funcionar con el ruido de un bar de fondo? y ¿cómo pudee ser que además identifique tan rápido a seta banda que nadie conoce?

Resulta que todo este proceso se basa en reconocer *huellas digitales acústicas*, una técnica robusta y eficiente que puede entenderse en términos de Señales y Sistemas.

Reconocimiento mediante huellas digitales acústicas

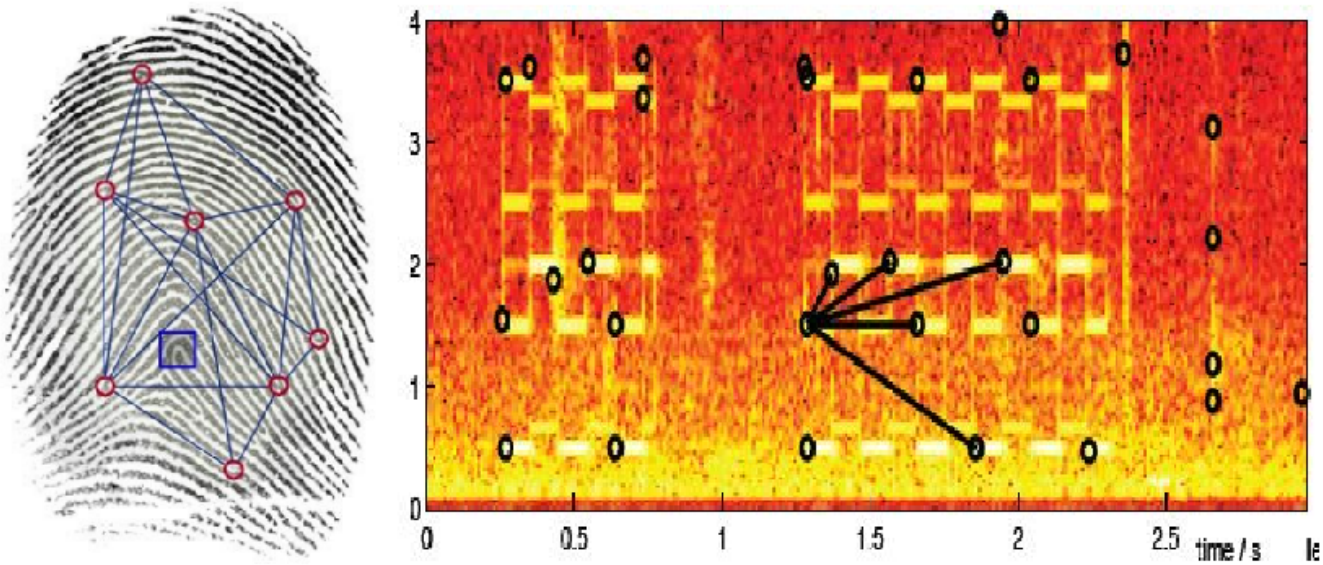
El reconocimiento de audio mediante huellas digitales acústicas es una técnica que busca identificar una pieza de audio, contrastando la información contenida en dicha pieza contra la almacenada en una base de datos de pistas conocidas. Esta técnica comienza a desarrollarse desde el año 2000,

pero es durante la última década que tomé mayor impulso, siendo posible hoy en día encontrar una variedad de aplicaciones para smartphones capaces de identificar casi cualquier pieza de audio en segundos [4].

Existen varias formas dea bordar el problema del reconocimiento, pero todas persiguen la misma finalidad:

- Simplicidad computacional: el reconocimiento debe realizarse en forma rápida.
- Eficiencia en el uso de memoria y buena capacidad de discriminación: existen aproximadamente 100 millones de canciones grabadas [1].
- Robustez ante degradaciones: ruido de fondo, degradación por compresión digital del audio, ecualización por respuesta en frecuencia no plana del lugar y/o parlantes, etc.
- Granularidad: capacidad de reconocimiento utilizando sólo un segmento del audio.
- Alta tasa de aciertos, baja tasa de falsos positivos y de rechazos.

El procedimiento básico consiste en analizar el segmento de audio buscando extraer características particulares en el esapcio tiempo-frecuencia (es decir, en su espectrograma) que sirvan para identificarlo luego. Una característica podría ser, por ejemplo, la potencia que posean las diferentes bandas de frecuencias. Al conjunto de estas características se las denomina *huella digital acústica*.



Este procedimiento de extracción de huellas se utiliza tanto para confeccionar la base de datos de canciones conocidas como para posteriormente reconocer segmentos de audio que se desean identificar consultando la base de datos.

La elección de las características es la base del funcionamiento del método. Deben ser lo suficientemente específicas como para identificar a cada canción unívocamente pero a la vez ocupar poca memoria para permitir realizar la búsqueda en forma rápida y eficiente. También deberán ser relativamente inmunes ante ruidos y distorsiones del audio de manera de lograr un sistema de reconocimiento robusto.

Diferentes características han sido propuestas con el fin de lograr estas especificaciones. Algunos ejemplos que se extraen del dominio tiempo-frecuencia son los MFCC [4] (Mel-Frequency Cepstrum Coefficients, comúnmente utilizados para la representación del habla), SFM [1] (Spectral Flatness Measure, una estimación de cuánto se aproxima una banda espectral a ser un tono o ruido), la ubicación de los picos de potencia del espectrograma (la base del algoritmo de Shazam [9, 8]), la energía de las bandas [6], etc. Además del espectrograma, otras técnicas se han utilizados como *wavelets* y algoritmos de visión [2], algoritmos de machine learning [3], y meta-características [7], útiles en grabaciones muy distorsionadas.

En este trabajo desarrollaremos una implementación simple que utiliza como características el signo de la diferencia de energía entre bandas [5].

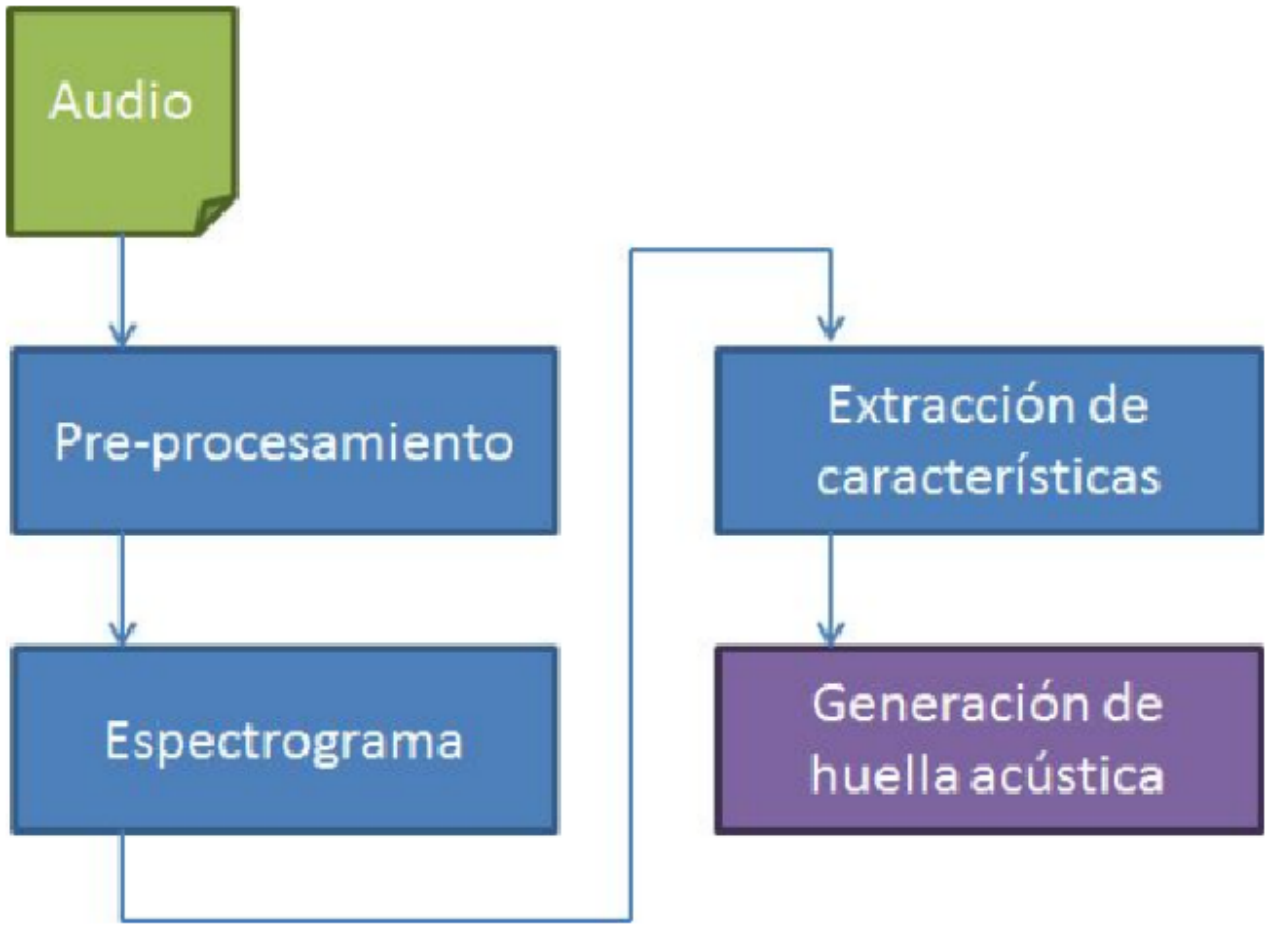
Cabe destacar que esta característica sólo sirve para reconocer las mismas versiones de las canciones que se almacenan en la base de datos: es decir, no está diseñada para reconocer versiones en vivo o interpretaciones diferentes a la original.

Descripción del sistema de reconocimiento

El sistema de reconocimiento consta de dos bloques principales:

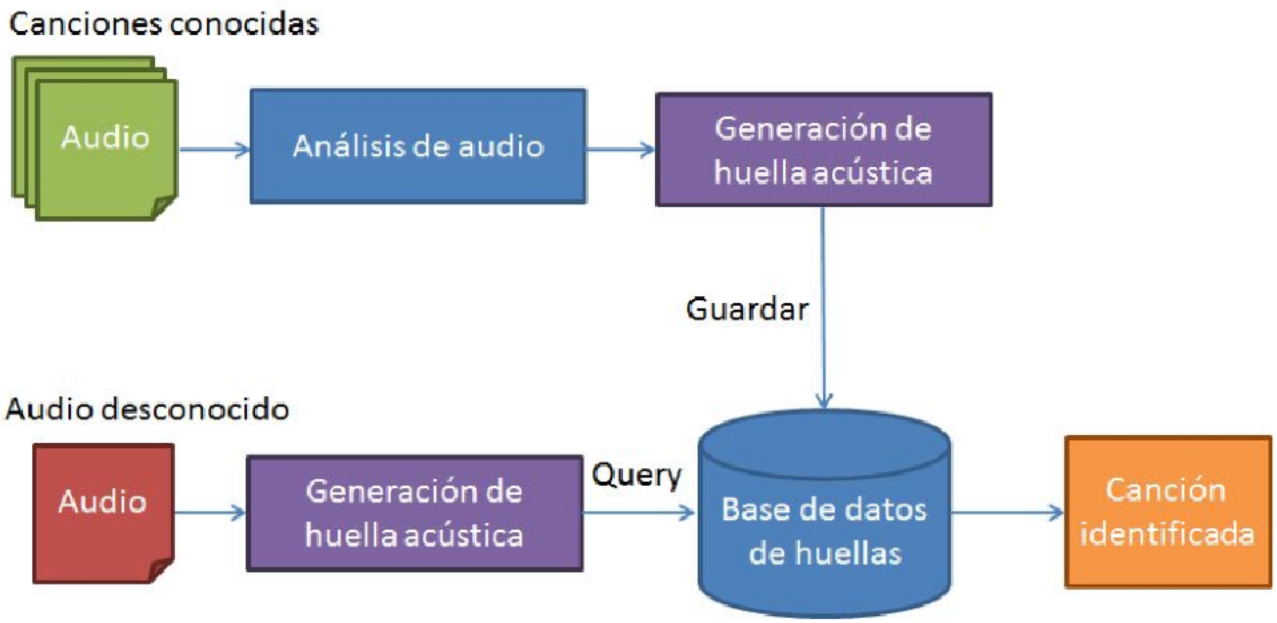
1. El algoritmo para extraer las huellas digitales acústicas.
2. La base de datos que almacena las huellas acústicas de las canciones conocidas y permite realizar búsquedas a partir de la huella acústica de una canción desconocida.

El algoritmo para extraer las huellas acústicas toma como entrada el archivo con el audio, lo pre-procesa, extrae las características, y entrega como resultado la huella acústica.



La base de datos guardará las huellas acústicas de las canciones conocidas. Tiene además un sistema de búsqueda (en inglés query) tal que al realizar un query con una huella acústica dada nos devuelve la canción – más probable – a la cual correseponde.

El esquema del sistema completo se presenta a continuación:



Observe que el algoritmo de extracción de huellas se utiliza tanto en la creación de la base de datos de canciones conocidas como para el reconocimiento de audios desconocidos.

Algoritmo de extracción de huellas digitales acústicas

El algoritmo de extracción de huellas acústicas tiene como finalidad extraer características a partir del espectrograma del audio. Primeramente, se acondiciona el audio con un pre-procesamiento. Luego se realiza un espectrograma del audio pre-procesado y finalmente se extraen las características para cada intervalo de tiempo.

1. Pre-procesamiento del audio

Se comienza convirtiendo el audio, que suele ser estar en estéreo, a un audio monocanal, promediando los canales.

Luego, se reduce su frecuencia de muestreo, aprovechando que son las frecuencias bajas las que contienen la información más útil para la extracción de características. En general, el audio está muestreado a 44100 Hz (calidad de CD), pero para este algoritmo de reconocimiento basta con tenerlo muestreado a 1/8 de su frecuencia original, 5512.5 Hz. Esto permite trabajar con menos muestras, aliviando la carga computacional. Para realizar los siguientes ejercicios, utilice el archivo `Pink.ogg`.

Ejercicio 1)

Cargue la pista de audio. Verifique que la variable cargada es una matriz con 2 columnas, cada una correspondiendo a un canal de audio. Promedie los canales utilizando al función mean para tener 1 solo canal y grafique una porción de la señal resultante. Escuche el audio resultante para verificar el resultado.

```
44100
```

```
loadaudio (generic function with 1 method)
```

```
pink = 441000×2 Array{Float32,2}:
        0.581058   0.717792
        0.946098   0.893093
        0.86118    0.783726
        0.760788   0.722609
        0.795677   0.691608
        0.73639    0.664538
        0.689886   0.773019
        ⋮
        0.0486554  0.0473221
        0.0863577  0.114512
        0.126865   0.173023
        0.166574   0.225978
        0.182585   0.245573
        0.124588   0.171235
```

```
pinkSound =
```

0:00 / 0:10

```
pinkMono =
  Float32[0.649425, 0.919596, 0.822453, 0.741699, 0.743643, 0.700464, 0.731453, 0.807
```

```
pinkMeanSound =
```

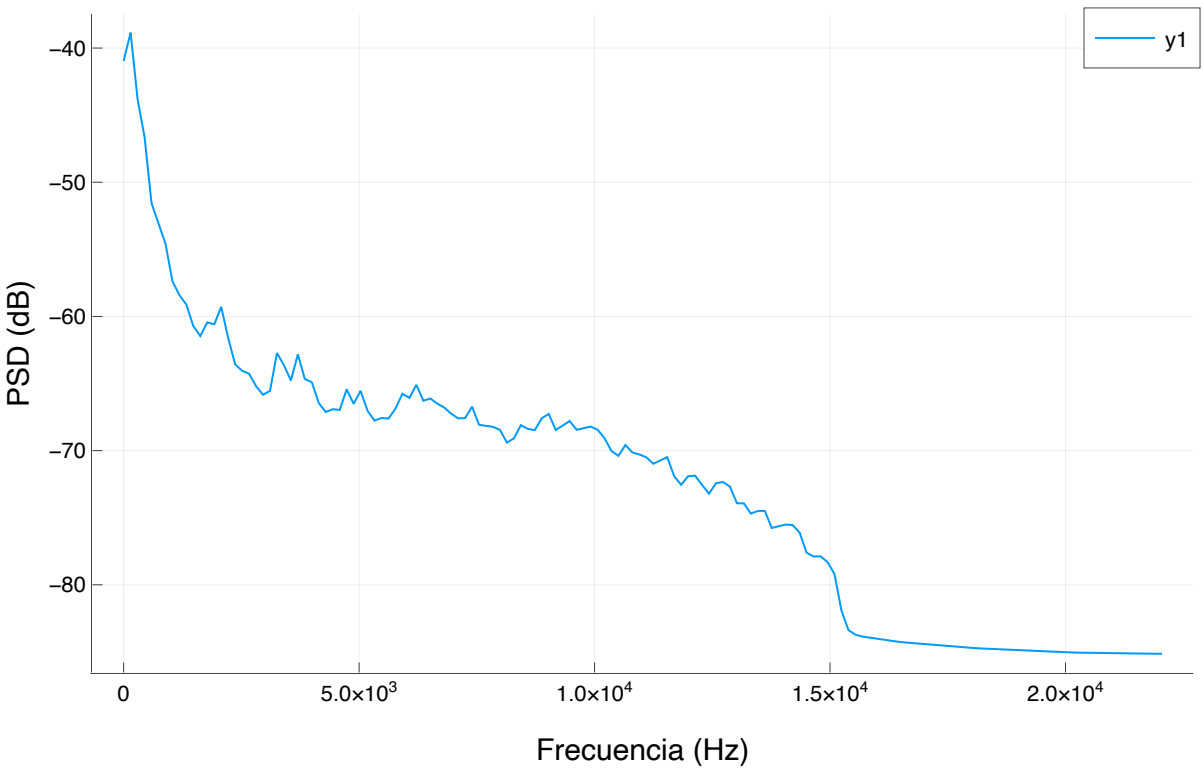
0:00 / 0:10

Ejercicio 2)

Mediante la función `fft`, realice un análisis espectral de la señal mostrando la densidad espectral de potencia (PSD) en función de la frecuencia en Hz. Verifique que la mayor PSD se concentra en las bajas frecuencias. Utilice escala logarítmica para mostrar esultado.

La PSD, S_{XX}, la puede estimar como $S_{XX}(\omega) = 1/T|X(\omega)|^2$, donde T es la duración del segmento temporal que usa para la estimación, o mejor aún, dividiendo al audio en segmentos, realizando la estimación anterior con cada uno, y finalmente promediándolas.

```
psd =
  Float32[-40.9493, -38.8508, -43.8107, -46.626, -51.5645, -53.0943, -54.5677, -57.38
```

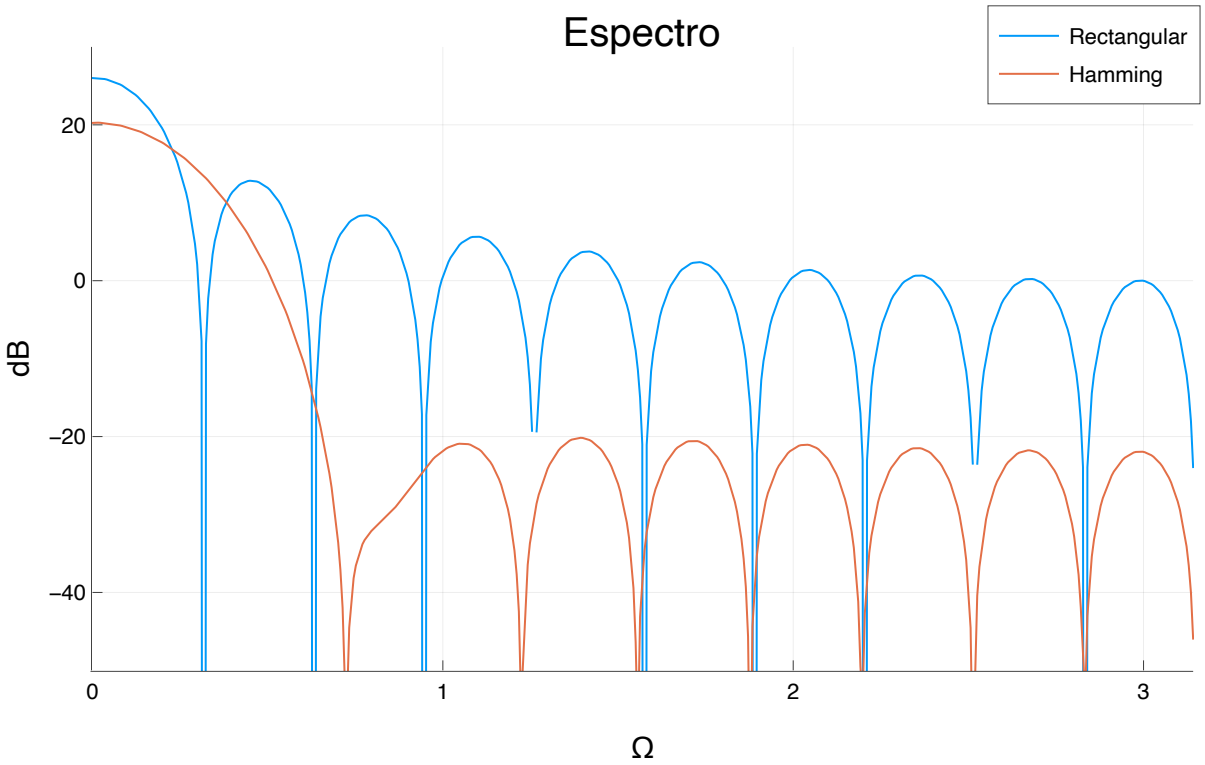


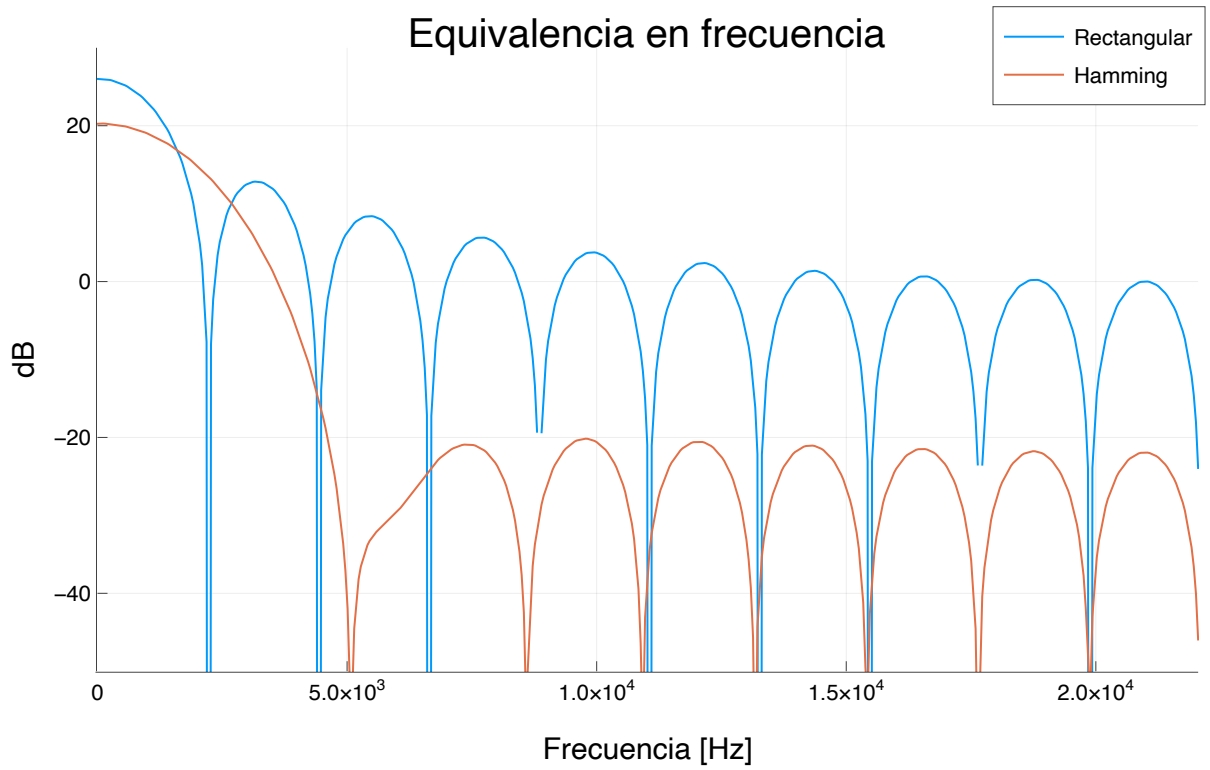
Como se observa, la frecuencias mas bajas tienen mayor potencia.

Ejercicio 3)

Mediante la función `fft`, obtenga el espectro de una ventana rectangular y una de Hamming de igual duración y grafique su potencia en escala logarítmica en un mismo gráfico para compararlos. ¿Cuál es la resolución en frecuencia de cada ventana? Al realizar un espectrogama, ¿qué ventaja tiene utilizar la ventana de Hamming frente a la rectangular?

La ventana de hamming tiene lobulos laterales significativamente menores respecto a los lobulos laterales de la ventana rectangular, aunque el lobulo principal es aproximadamente el doble que la de la ventana rectangular. Al tener lobulos laterales mas grandes, tendremos una mayor amplificacion en las frecuencias. Esto provoca que en el espectograma se muestren frecuencias que no correspondan.



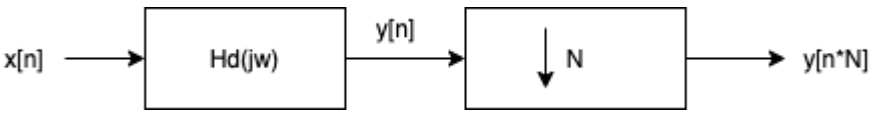


Observando los gráficos, podemos notar que el lóbulo principal de la ventana rectangular tiene una resolución de aproximadamente 2.5 KHz y la de la ventana de hamming 5KHz

Ejercicio 4)

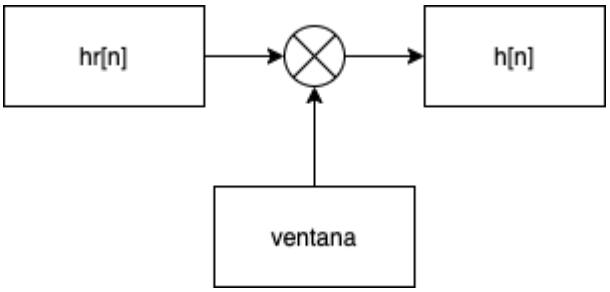
Implemente un sistema para reducir la frecuencia de muestreo del audio, de 44100 Hz a 5512.5 Hz. Muestre un diagrama en bloques y justifique su diseño. Diseñe el filtro pasabajos mediante el método de ventaneo explicando y justificando las decisiones, de forma tal que su retardo sea menor a 1 ms. Graficar un diagrama de polos y ceros, respuesta en frecuencia (módulo y fase), respuesta al impulso, y retardo de grupo.

Para reducir la frecuencia de muestreo debemos pasar la señal por un filtro pasa bajos y luego decimar.



Como debemos reducirlo a un tamaño 8 veces menor utilizaremos una frecuencia de corte equivalente a sr/8 y N = 8.

Pero el filtro ideal no tiene soporte acotado, por lo cual debemos recortarlo utilizando el metodo de ventaneo.



Si bien ahora tenemos un soporte acotado, debemos realizar un desplazamiento a la derecha de la respuesta al impulso para que el sistema sea causal, pero esto nos genera un desfasaje lineal y un retardo de grupo.

Para calcular el retardo de grupo debo analizar la cantidad de muestras por segundo que hay en las canciones, dado que la frecuencia de muestreo es de 44100Hz sabemos que hay 44100 muestras por segundo (44.1 muestras cada milisegundo). Al aplicar el metodo de ventaneo tenemos un sistema que no es causal, por lo cual debemos realizar un desplazamiento para convertirlo en causal y esto genera un delay equivalente a (N - 1) / 2, siendo N el orden del filtro. Dado que queremos un delay menor a 1 ms tenemos que cumplir con que (N - 1)/2 < 44.1 dando N < 89.2 (N natural). Se eligio N = 81

Mostramos el efecto de filtrar usando usando una ventana rectangular y luego submuestrear.

```
pinkMonoFiltRec =
  Float64[-1.11022e-16,  0.0020284,  0.00671888,  0.0131774,  0.0202393,  0.0268405,  0.031

pinkSubratedRec =
  Float64[-1.11022e-16,  0.0340695,  -0.0134714,  0.0524563,  -0.0729597,  0.457034,  0.798
```

0:00 / 0:10

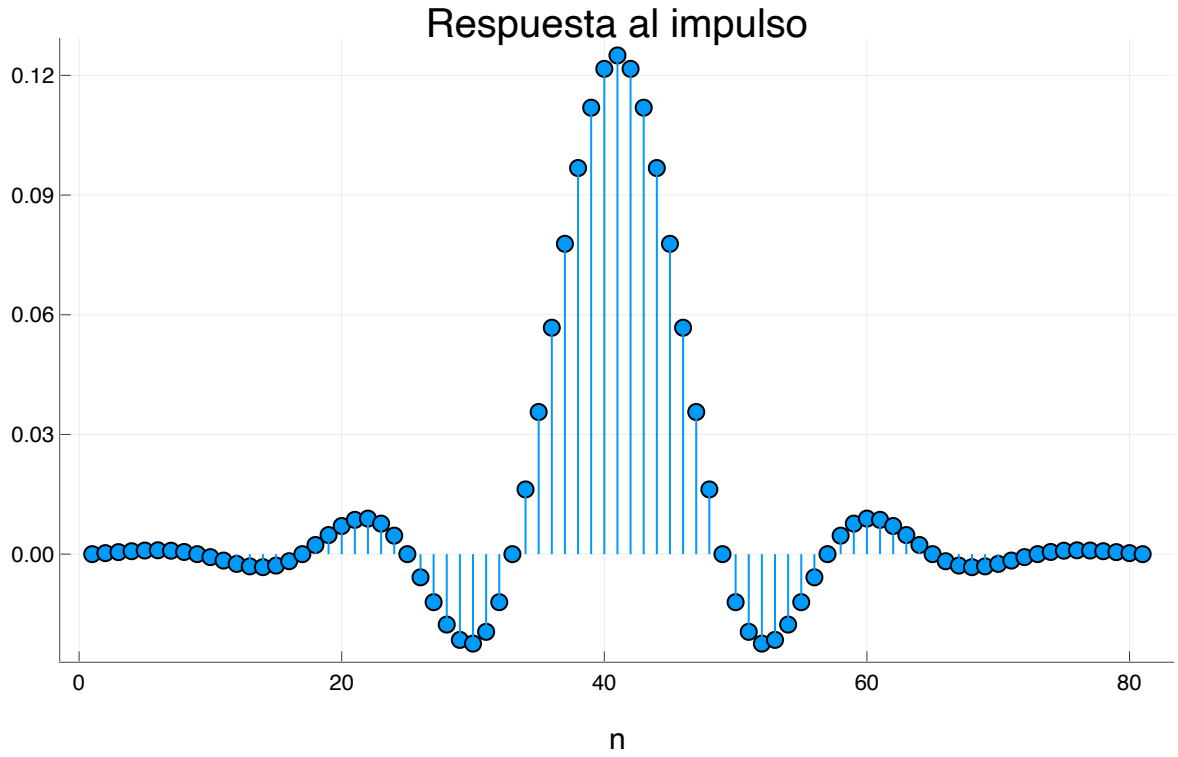
Mostramos el efecto de filtrar usando usando una ventana de hamming y luego submuestrear.

```
pinkMonoFiltHamming =
  Float64[5.55112e-17,  0.000165149,  0.000563368,  0.00115429,  0.0018722,  0.00263262,  0

pinkSubratedHamming =
  Float64[5.55112e-17,  0.00369376,  -0.00890007,  0.0262507,  -0.0645101,  0.443526,  0.79
```

0:00 / 0:10

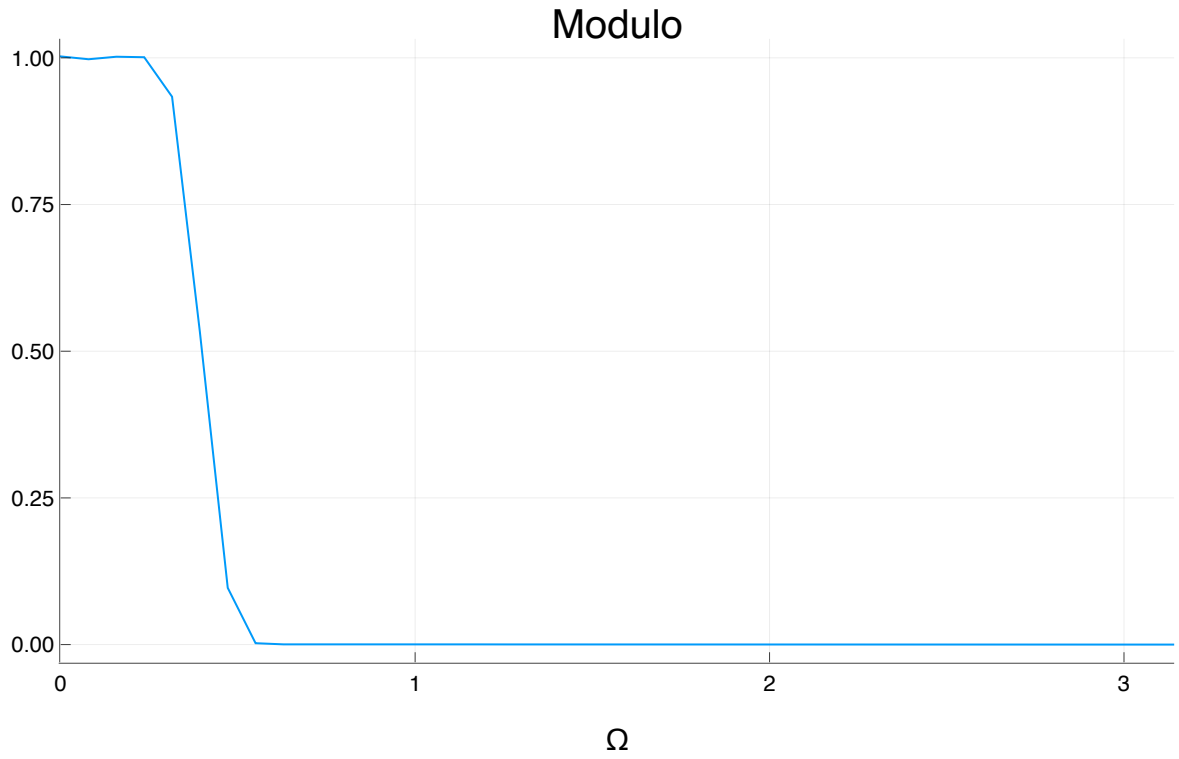
Graficos correspondientes al filtro usando ventana de hamming.



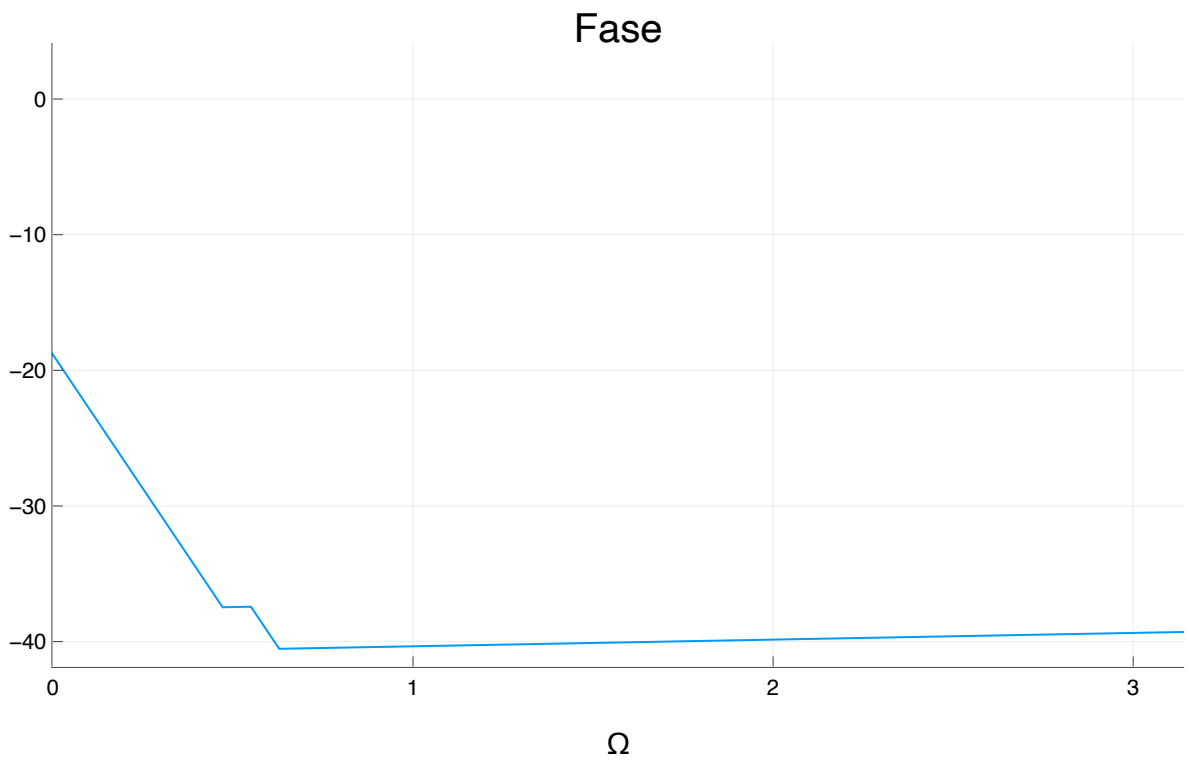

```
groupDelay (generic function with 1 method)
```

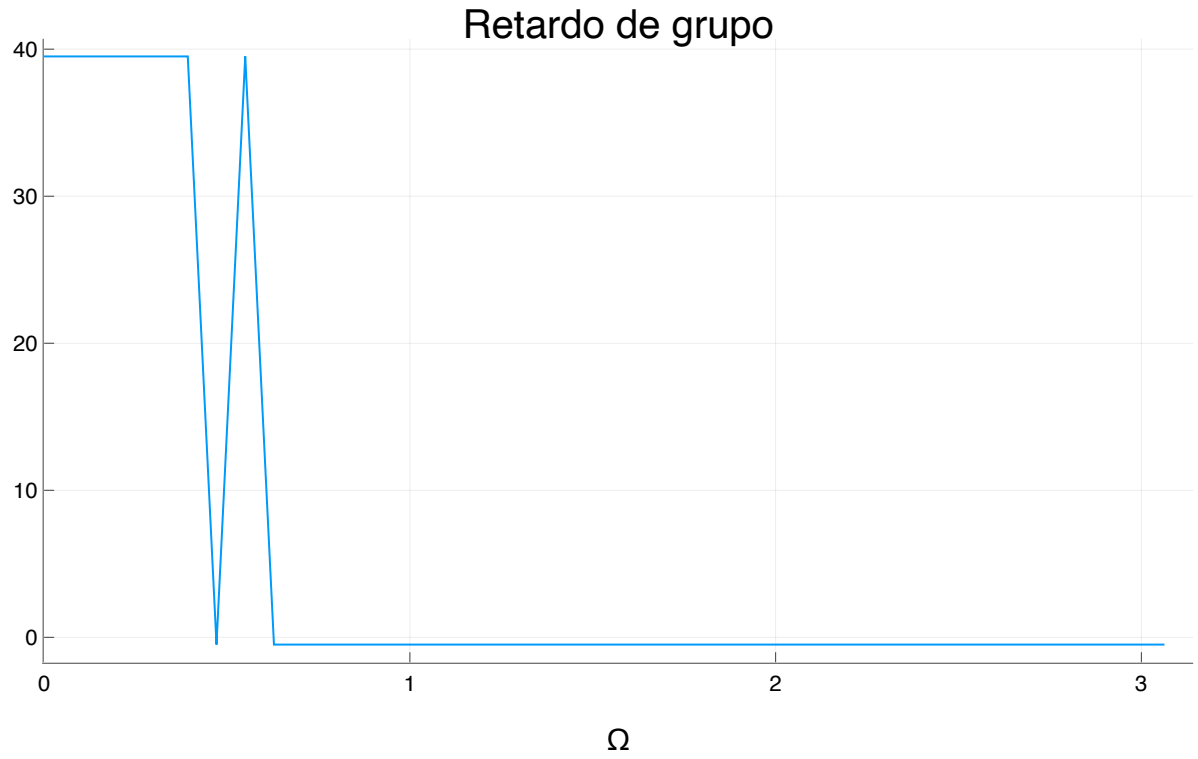
```
Float64[-0.493827, -0.493827, -0.493827, -0.493827, -0.493827, -0.493827, -0.493827
```

Aca observamos el modulo de la respuesta en frecuencia. Se nota que la forma es muy similar al filtro ideal.



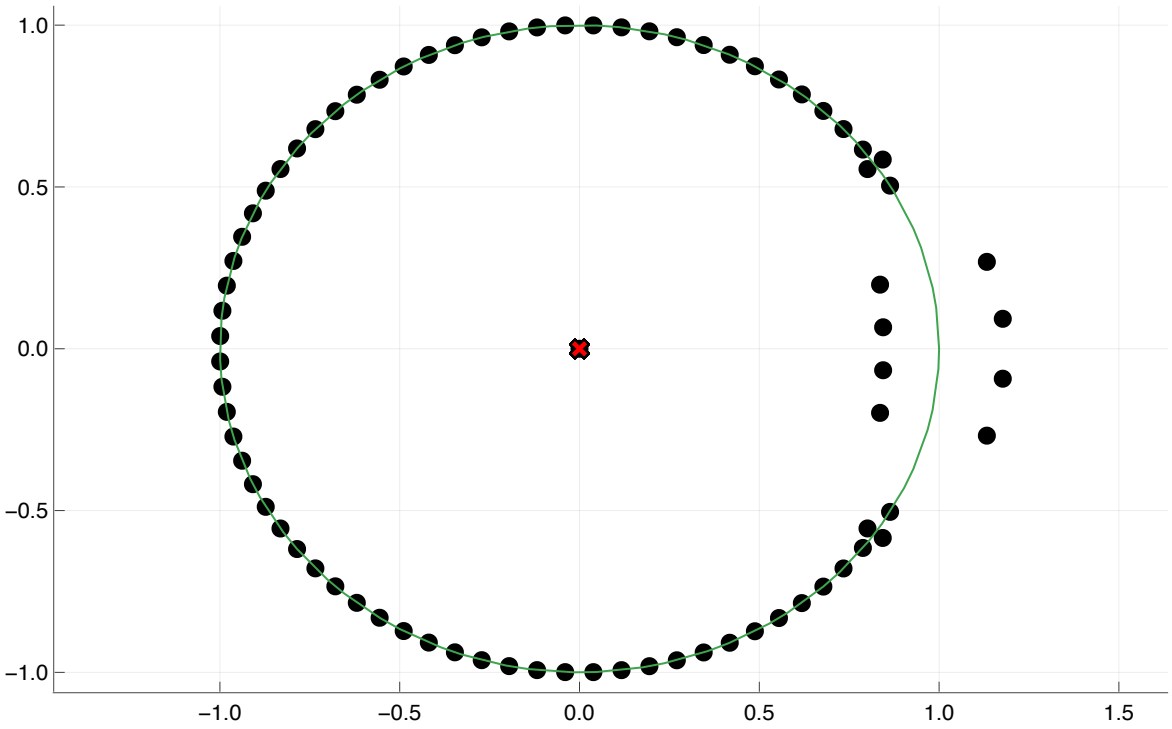
En el gráfico de la fase, en sector entre 0 y pi/8, vemos que esta es lineal, lo cual se corresponde con el desfasaje que se le aplico al filtro ideal. El desplazamiento fue de 40 puntos a la derecha, por lo cual se espera que haya una pendiente de 40 lo cual se observa en el gráfico.





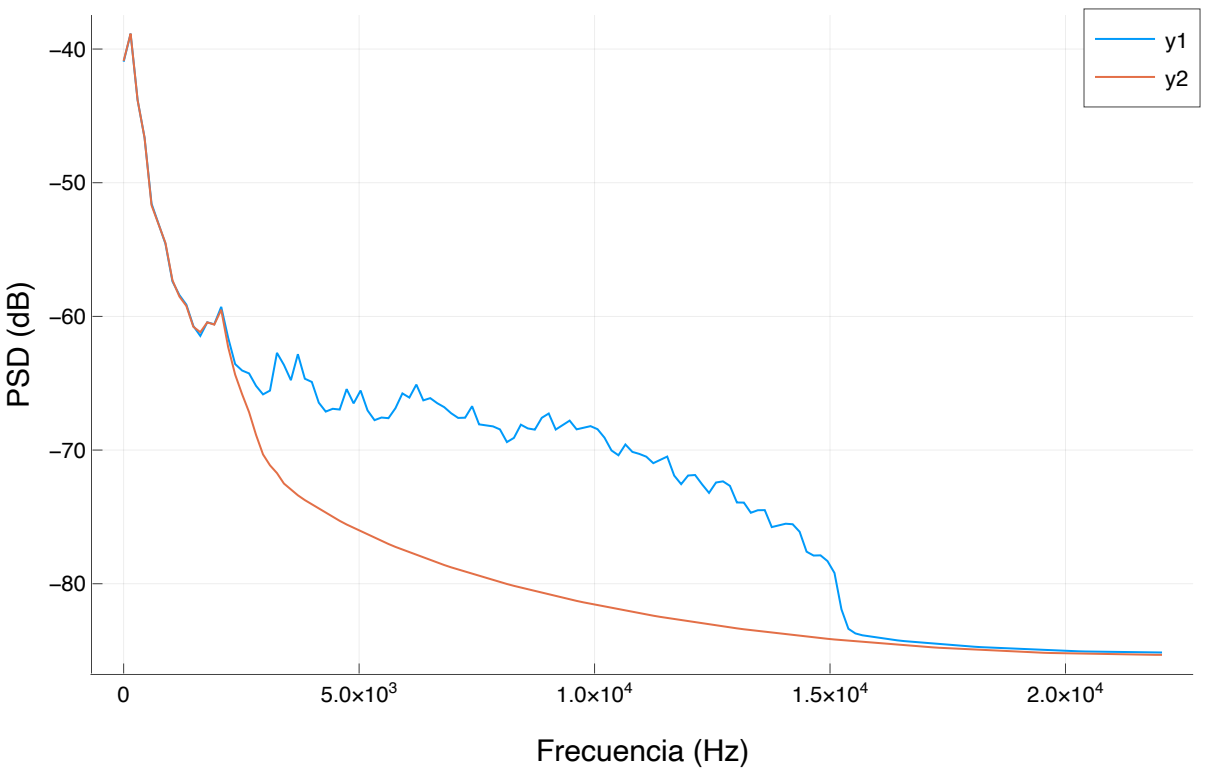
En el siguiente diagrama de Polos y Ceros observamos que tenemos polos en el origen y ceros sobre el origen y sobre el circulo unitario, a excepcion de la zona correspondiete a ω chicas. Esto tiene sentido ya que un filtro FIR si tiene polos los tiene en el origen y/o infinito y estamos realizando un filtro pasabajos por los cuales las bajas frecuencias deben pasar.

Por las características del filtro, se esperan tener 80 polos en cero y como la fase es lineal se esperan que los ceros sean 80, siendo pares conjugados. En este caso, tenemos 79 ya que el cero ubicado en el origen es doble.



Aca comparamos las PSD correspondientes a la señal original y la señal filtrada, como se observa, en las frecuencias bajas ambas señales tienen la misma potencia, pero llegado a un punto, la potencia de las frecuencias mas altas decae abruptamente en la señal filtrada con respecto a la original, lo cual tiene sentido ya que estabamos filtrando con un pasabajos.

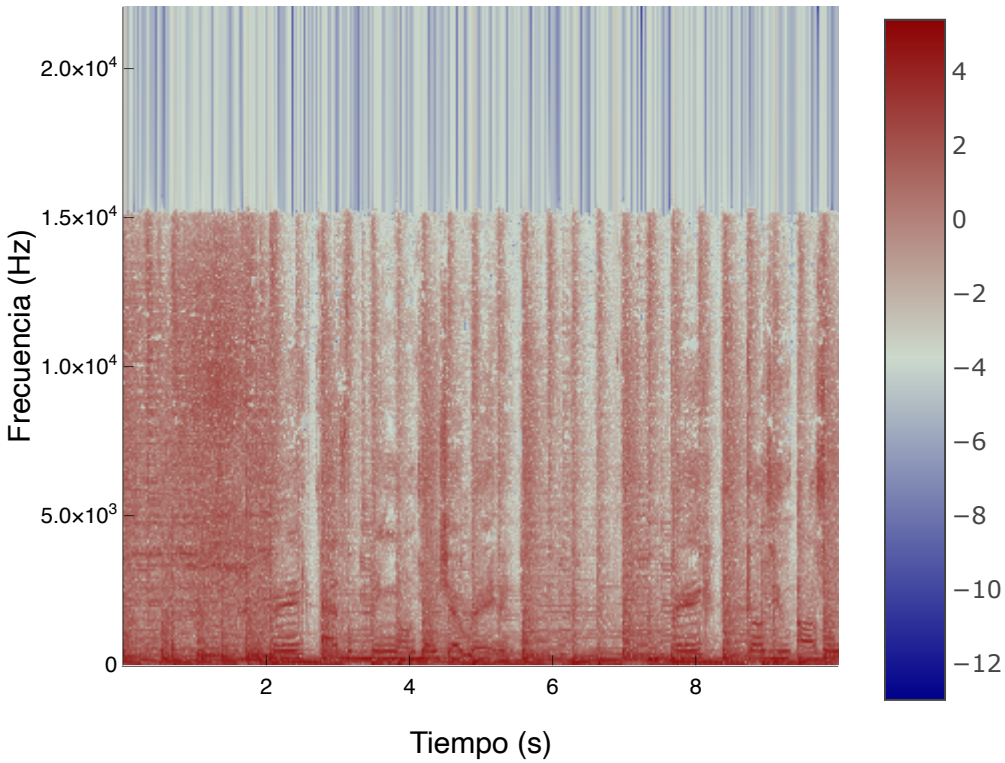
```
psdFiltered =
  Float64[-40.8863, -38.8486, -43.8146, -46.637, -51.6839, -53.0156, -54.5082, -57.29
```



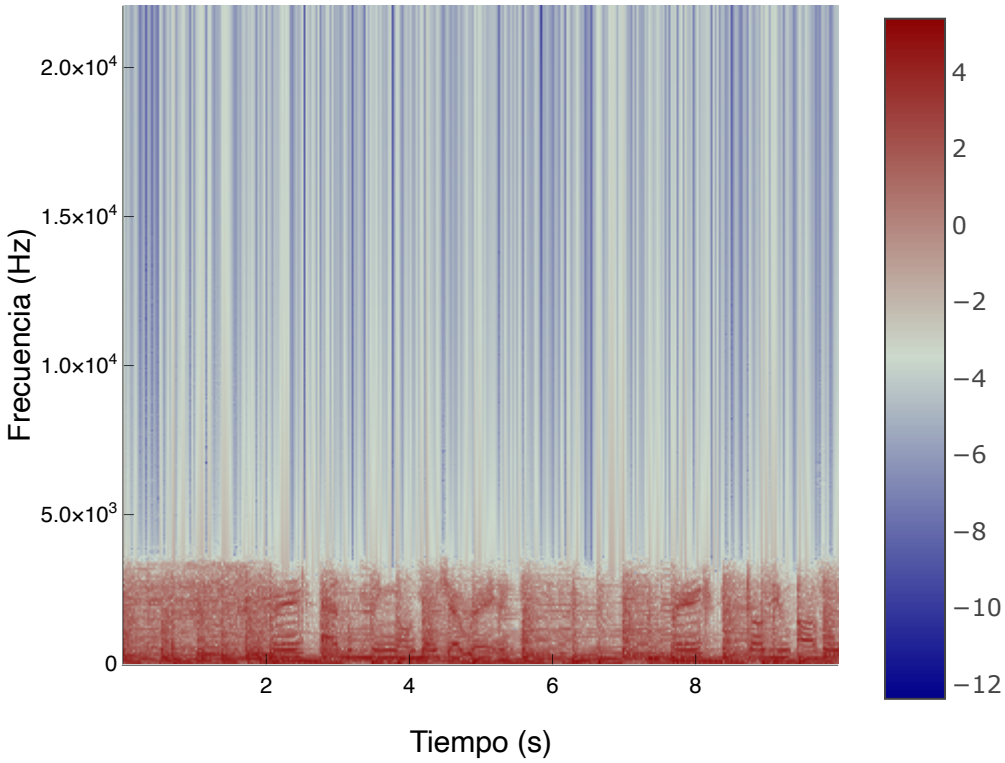
Ejercicio 5)

Realice un espectrograma de la señal original y la filtrada y verifique los efectos del filtrado. Indique el tipo y longitud de ventana utilizada.

Para ambos casos se utilizar una ventana de hamming de 1024 puntos con un solapamiento del 90%.



Como se observa en el espectograma, vemos que se detectan frecuencias de hasta 15 KHz pero se nota que las frecuencias bajas predominan.



En este caso, observamos que hay frecuencias de hasta 2700 Hz aproximadamente. Lo cual tiene sentido debido a que la filtramos por un pasabajos con un frecuencia de corte 8 veces mas baja que la frecuencia de muestreo.

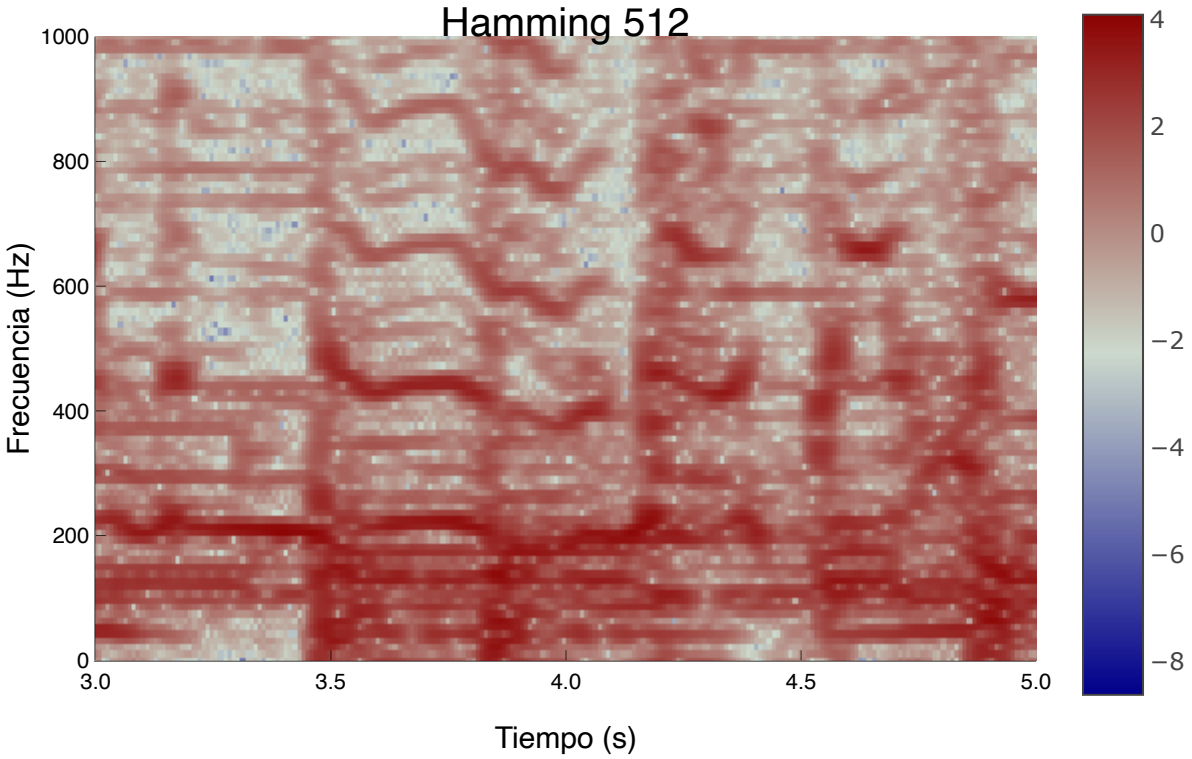
Espectrograma

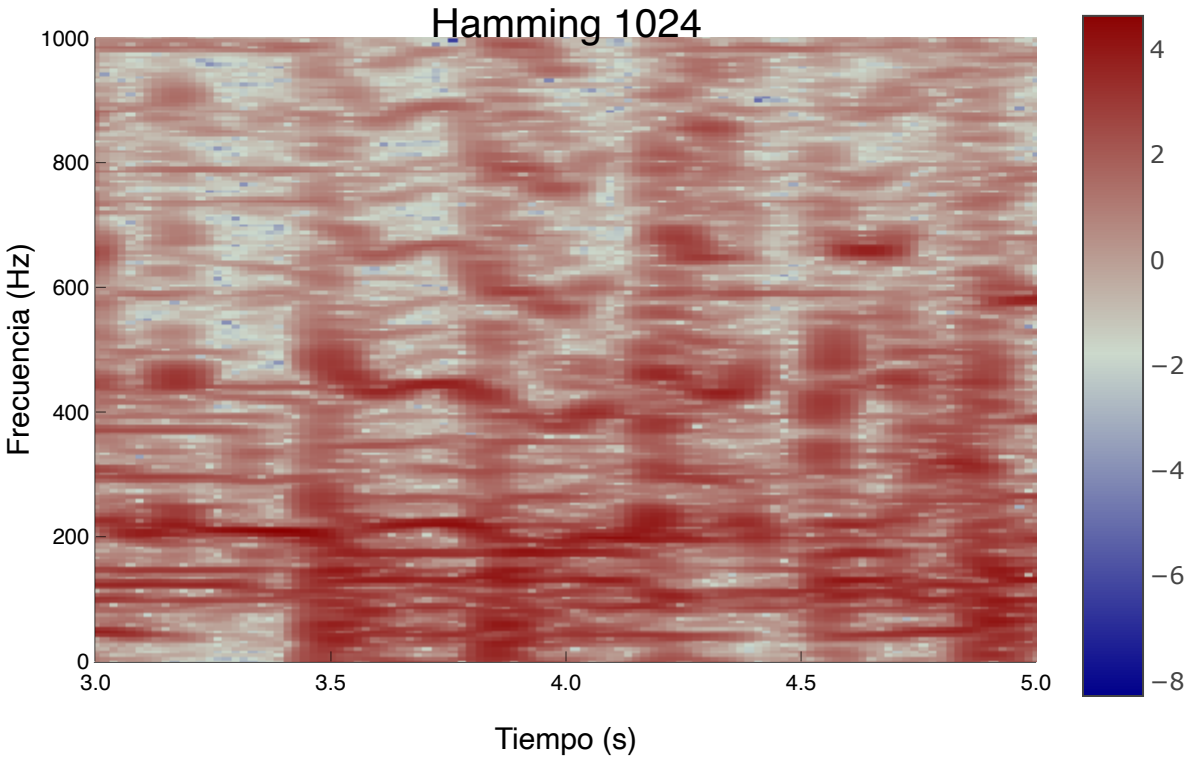
Los parámetros con los que se realice el espectrograma tienen una influencia directa sobre la extracción de las características. La elección de la ventana (tipo y longitud) es el parámetro principal, ya que debería lograr una resolución razonable tanto en tiempo como en frecuencia que permita distinguir las características espectrales y temporales con claridad.

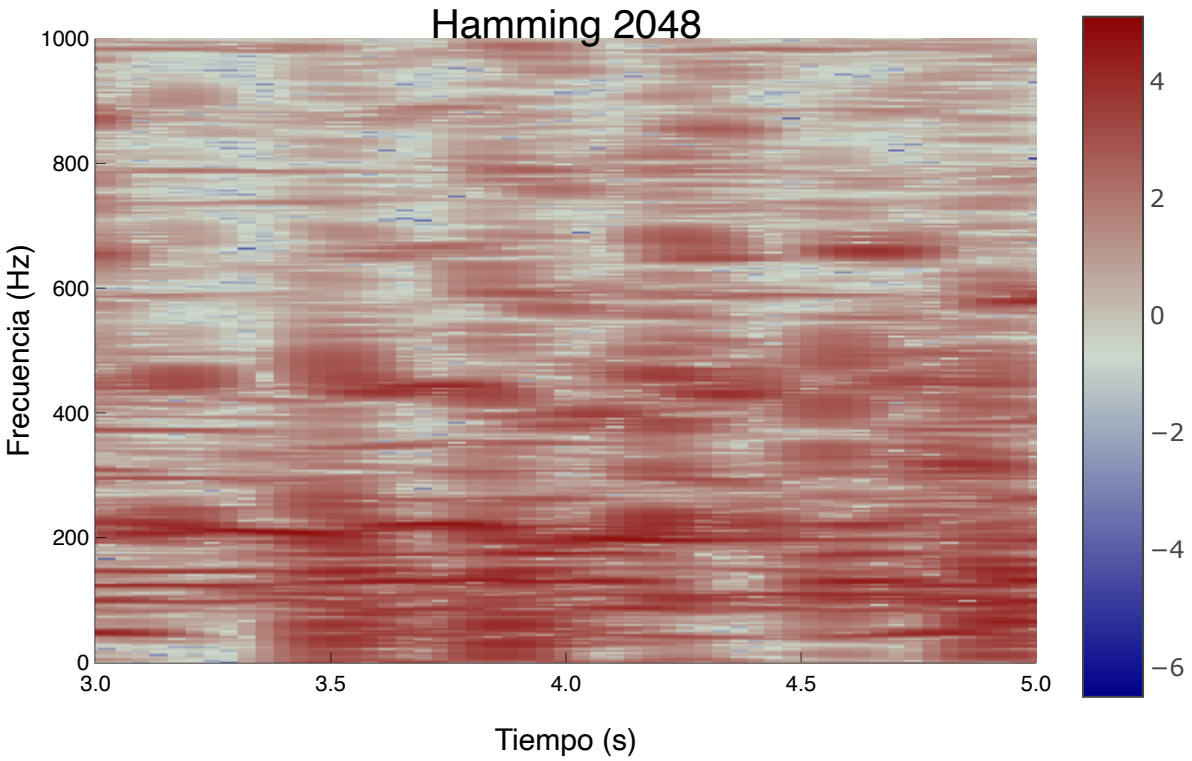
Ejercicio 6)

Realice un espectrograma de la señal sub-muestreada y muestre una imagen (con zoom) de alguna región donde se vean las características espectrales de la señal dada la ventana utilizada. Muestre y justifique cómo cambia la visualización de las características con 3 diferentes longitudes de ventana y comente qué longitud utilizará en el algoritmo. Realice las comparaciones con ventana rectangular y de Hamming.

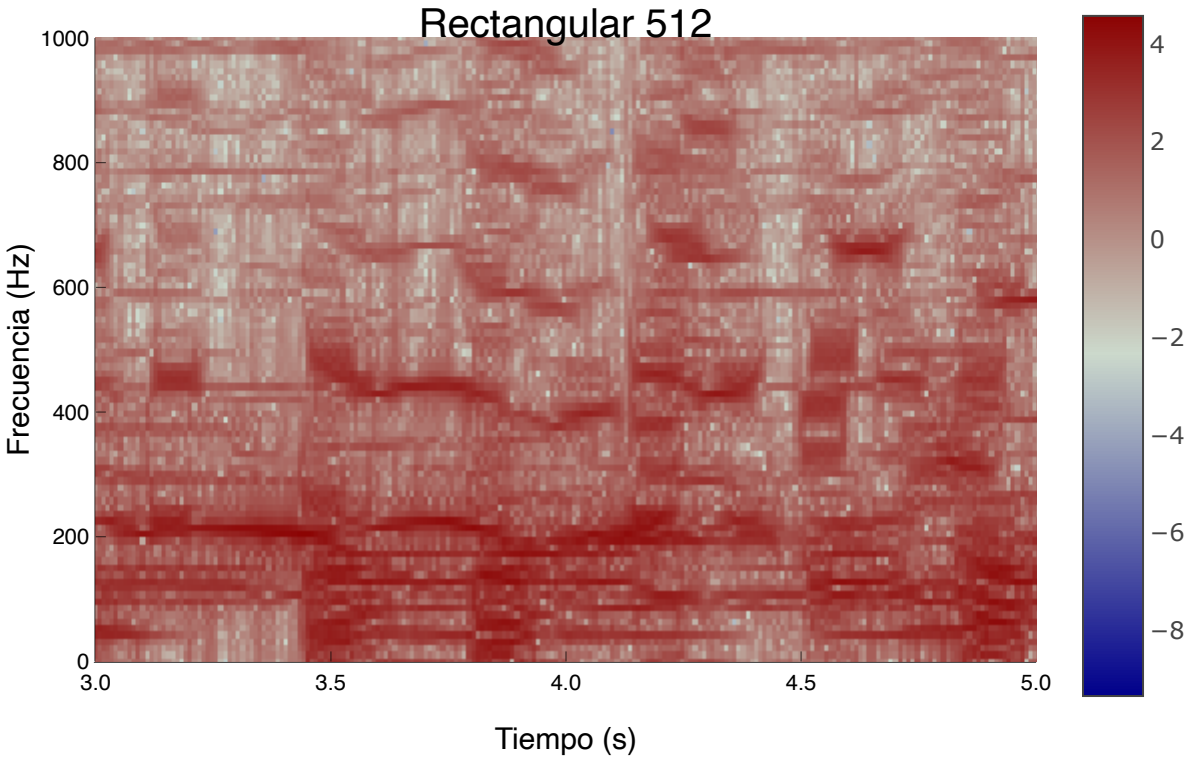
Espectogramas con ventana de hamming de 512, 1024 y 2048 puntos, con un solapamiento del 90%.

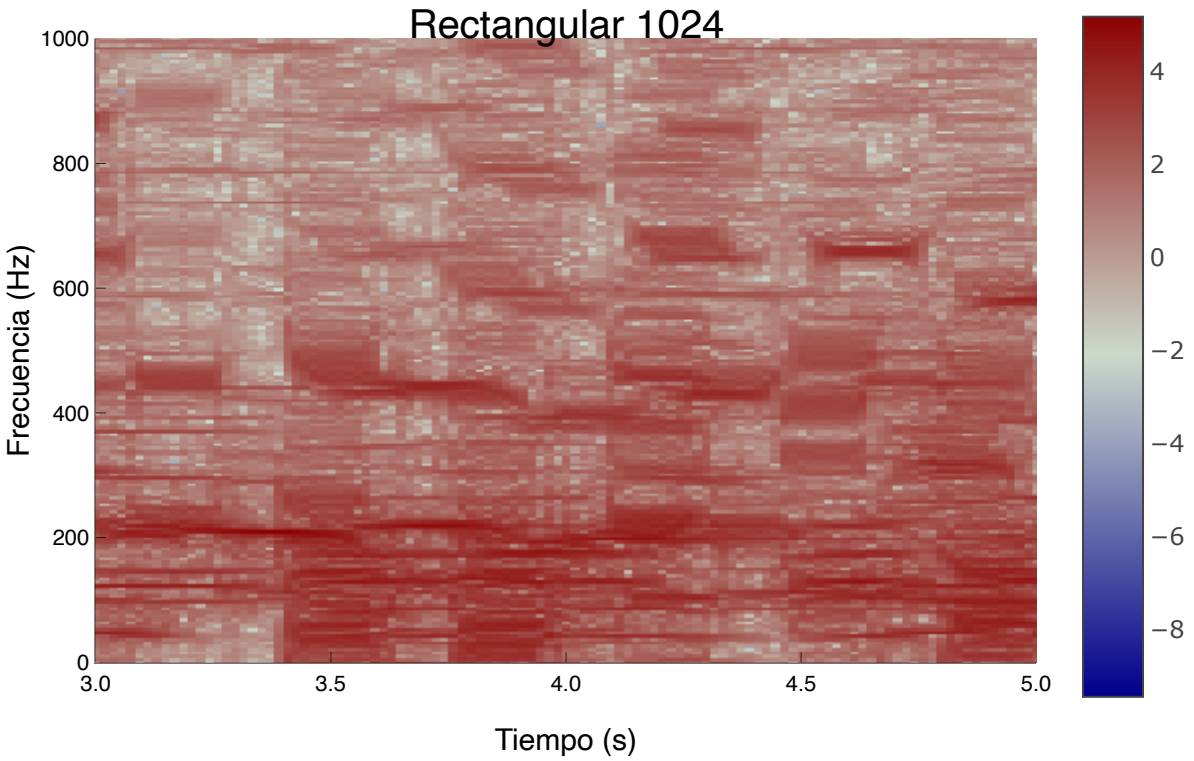

Hamming 512
Frecuencia (Hz)
1000
800
600
400
200
0
3.0
3.5
4.0
4.5
5.0
Tiempo (s)
4
2
0
−2
−4
−6
−8

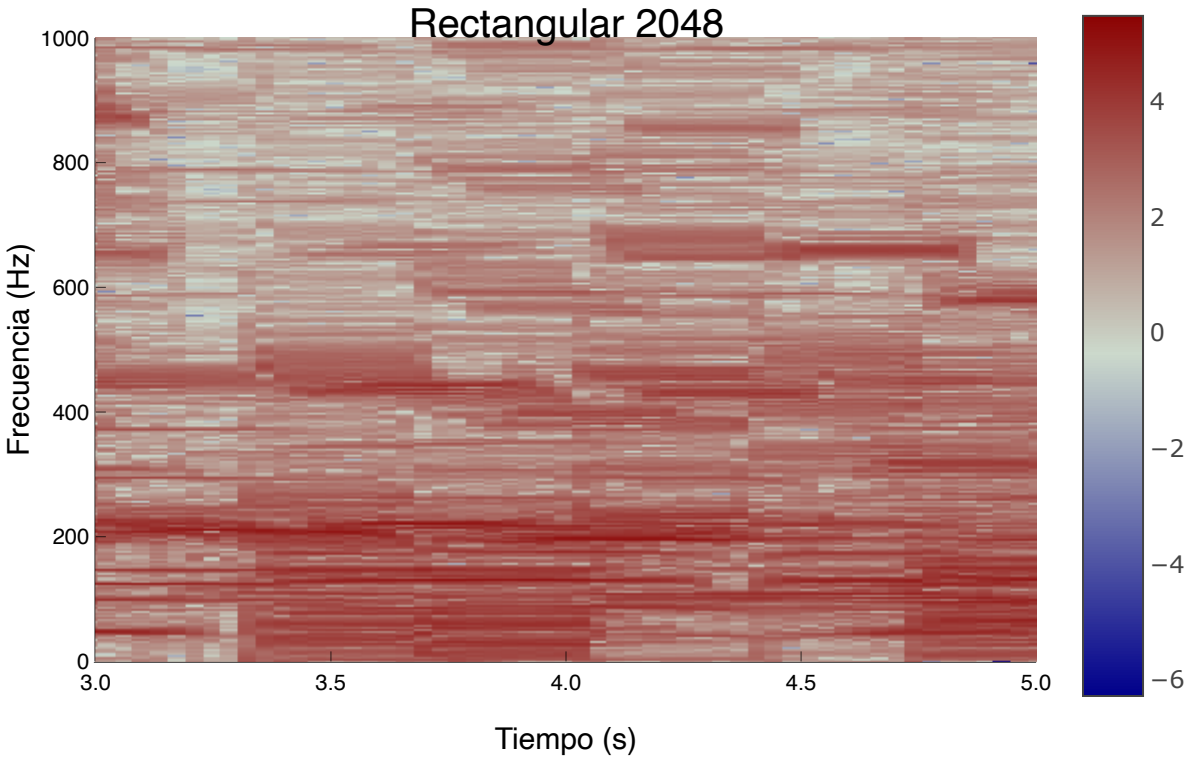

Hamming 1024
Frecuencia (Hz)
1000
800
600
400
200
0
3.0
3.5
4.0
4.5
5.0
Tiempo (s)
4
2
0
−2
−4
−6
−8



Espectogramas con ventana rectangular de 512, 1024 y 2048 puntos, con un solapamiento del 90%.



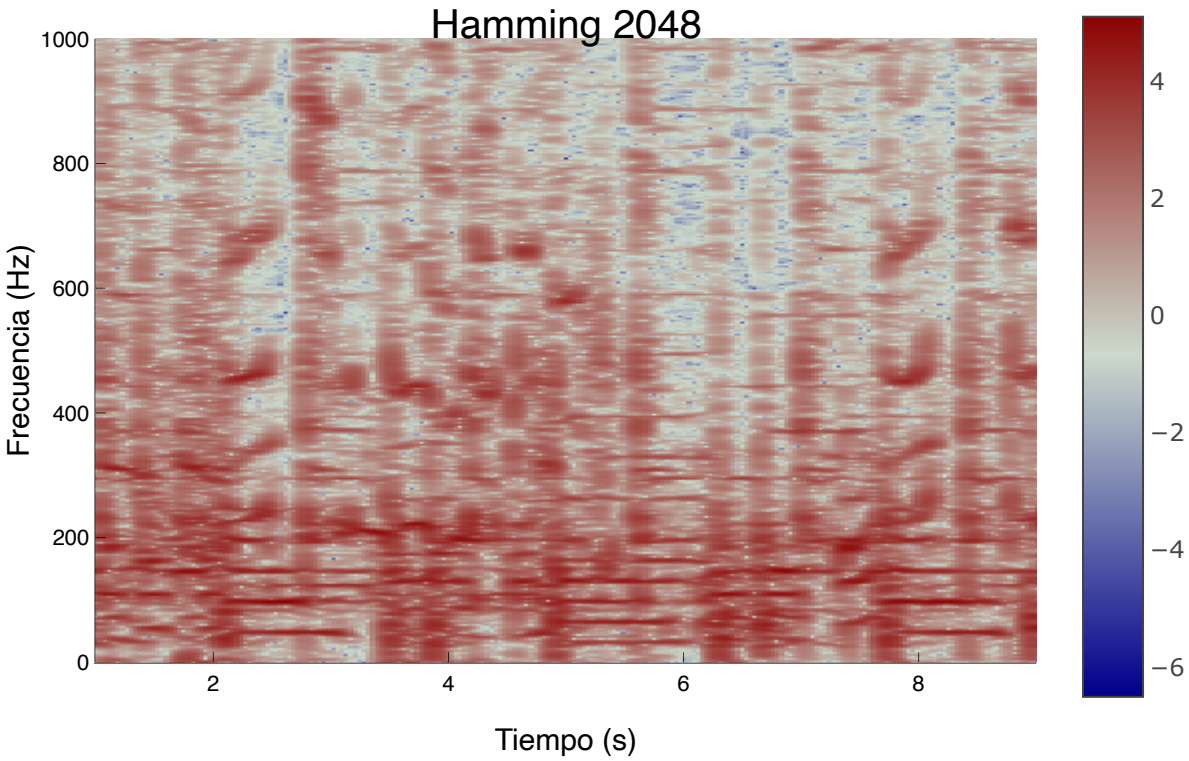




A simple vista, si comparamos los espectogramas correspondientes a la ventanas de 512 puntos podemos notar que en el espectograma que utiliza la ventana de hamming las frecuencias se encuentran bastante mas diferenciadas con respecto a la que utilizar la ventana rectangular.

Por otro lado, al utilizar una ventana mas grande se podrá visualizar de mejor manera el contenido en frecuencias perdiendo resolucion temporal, mientras que, cuanto mas chica es, mejora la resolucion temporal perdiendo resolucion espectral.

En el caso de la ventana de 2048 vemos que hay mucho solapamiento en las bandas de tiempo. Por lo cual nos conviene ver el espectograma con un rango mas grande en el tiempo.



Dado que a mayor ventana hay mayor resolución espectral y dado que en la comparación entre las ventanas de hamming y rectangular tenia mejor resolución la de hamming, se optó por utilizar una ventana de hamming de 2048 puntos para el algoritmo.

Extracción de características

La función provista `stft` permite obtener la matriz s con las sucesivas DFT de los segmentos de la señal ventaneados – lo que se grafica en un espectrograma.

Estas DFT nos devuelven muestras del espectro en frecuencias equiespaciadas en escala lineal – junto con los vectores de tiempos y frecuencias correspondientes a cada columna y fila.

A partir de esto, necesitaremos la energía en bandas de frecuencia equiespaciadas en escala logarítmica (esto es similar a la escala psicoacústica Bark) debido a que así funciona el oído humano, que es un buen reconocedor de canciones – la relación entre frecuencias y posición de resonancia de la membrana basilar es aproximadamente logarítmica.

Debemos obtener una matriz de energías `E`, cuyas columnas, al igual que las de la matriz del espectrograma `s`, representan características espectrales del n-ésimo segmento de señal ventaneado (frame `n`). Para todo `n`, el elemento `E[m, n]` de nuestra nueva matriz debe contener la energía total de todos los coeficientes de `s[k, n]` asociados a frecuencias que caen dentro de la k-ésima banda de frecuencia.

Ejercicio 7)

Divida al espectrograma en 21 bandas de frecuencias, equiespaciadas en escala logarítmica (es decir, el cociente entre frecuencias de bandas consecutivas debe ser constante). La frecuencia inferior de la primera banda debe ser 300 Hz y la superior de la última, 2 kHz.

Calculo de S

```
function calcS(x; window = hamming(1000), ov = 0.5)
    nfft = length(window)
    overlap = Int(round(nfft*ov))
    S = stft(x; overlap=overlap, window=window, nfft=nfft)
    S = abs.(S).^2
    return S
end;
```

```
Float64[0.0, 2.69428, 5.38856, 8.08284, 10.7771, 13.4714, 16.1657, 18.86, 21.5543,
```

```
2048×259 Array{Float64,2}:
  0.594393    3.39212   12.1862    18.3866    …    9.7039    40.4878    54.751    41.775
 25.815      18.4032    13.5863     8.77765        32.0615    21.487     33.8788   53.4367
  4.24479     2.17061    2.21579    1.5654         17.2583    21.0999    39.7401   93.1303
  5.48369     3.32974    2.18302    2.07357         2.01428   23.2964    83.1282  169.392
  9.3481      4.88029    2.1457     0.346644       21.2458    68.3451   162.883   240.953
  0.834305    3.59704    4.0836     3.34722    …   19.4867    80.1745   208.232   335.376
  3.45155     1.21223    2.05653    2.95767        40.0255   117.722    241.544   342.968
  ⋮                                              ⋱    ⋮
  3.45155     1.21223    2.05653    2.95767        40.0255   117.722    241.544   342.968
  0.834305    3.59704    4.0836     3.34722        19.4867    80.1745   208.232   335.376
  9.3481      4.88029    2.1457     0.346644       21.2458    68.3451   162.883   240.953
  5.48369     3.32974    2.18302    2.07357    …    2.01428   23.2964    83.1282  169.392
  4.24479     2.17061    2.21579    1.5654         17.2583    21.0999    39.7401   93.1303
 25.815      18.4032    13.5863     8.77765        32.0615    21.487     33.8788   53.4367
```

Calculo de E

```
function calcE(S, bands, freq)
    cantFilas, cantCol = size(S)
    E = [zeros(cantCol) for x in 1:21]
    j = 1
    while freq[j] < bands[1]
        j += 1
    end;
    for i in 1:length(bands)-1
        cant = 0
        lowFreq = bands[i]
        highFreq = bands[i+1]
        while freq[j] < lowFreq
            j += 1
        end;
        while j < length(freq) && freq[j] >= lowFreq && freq[j+1] < highFreq
            for k in 1:cantCol
                E[i][k] += S[cantFilas * (k - 1) + j]
            end;
            cant += 1
            j += 1
        end;
        E[i] ./ cant
    end;
    return E
end;
```

```
E =
```

```
Array{Float64,1}[Float64[1013.16,  951.855,  779.782,  540.391,  327.379,  193.808,  158.
```

Finalmente, las características se obtienen mediante una función que opera sobre E(m, n) según:

$$H[m,n] = \begin{cases} 1 & \text{si E[m+1, n] - E[m, n] > E[m+1, n-1] - E[m, n-1]} \\ 0 & \text{en otro caso} \end{cases}$$

En palabras, $H[m,n]$ es 1 si la diferencia de energía entre las bandas m y $m+1$ para el *frame* actual n es mayor a la del *frame* anterior $n-1$. Experimentalmente, se verficó que estas características son robustas ante varios tipos de procesamientos y distorsiones del audio[5].

Ejercicio 8)

Implemente el algoritmo de extracción de características, calculando la huella. Debido al efecto borde, $H[m,n]$ debería resultar una matriz de 20 filas. Ejecute el algoritmo sobre un segmento de audio y muestre una imagen de la huella digital acústica obtenida.

Calculo de H

```
function calcH(E)
    f = length(E)-1
    c = length(E[1])
    H = Int.(zeros(f,c))
    for m in 1:length(E)-1
        for n in 2:length(E[m])
            dif1 = E[m+1][n] - E[m][n]
            dif2 = E[m+1][n-1] - E[m][n-1]
            H[f*(n-1)+m] = dif1 > dif2 ? 1 : 0
        end
    end
    return H
end;
```

```
H = 20×259 Array{Int64,2}:
 0  1  1  1  1  1  0  0  0  0  0  0  0  1  …  0  0  0  0  1  1  1  1  1  0  0  0  0
 0  0  1  1  1  1  1  1  1  1  1  0  0  0     0  0  0  1  1  1  1  1  0  1  1  1  1
 0  1  1  1  1  0  0  0  0  0  0  1  1  1     0  0  0  1  0  0  0  1  1  0  0  0  0
 0  1  1  1  1  0  1  1  1  1  1  0  0  0     0  0  0  0  0  0  0  0  0  1  1  1  1
 0  0  0  0  1  1  1  0  0  0  0  1  1  1     1  1  1  1  1  1  1  0  1  1  1  0  0
 0  1  1  1  1  1  1  0  0  0  0  1  1  1  …  1  1  1  1  0  0  1  1  0  0  0  0  0
 0  0  0  0  0  0  0  1  1  1  1  0  0  0     0  0  1  1  1  1  1  1  0  0  0  0  0
 ⋮                 ⋮                 ⋮     ⋱           ⋮                 ⋮
 0  1  1  1  0  0  0  0  0  0  0  0  1  1     1  1  1  1  1  1  0  0  0  0  0  0  0
 0  0  0  1  1  1  0  0  0  1  1  1  1  1  …  1  1  1  0  0  0  1  1  1  0  0  0  0
 0  1  1  1  1  1  1  1  1  1  0  0  0  0     0  0  0  0  1  0  0  0  1  1  1  1  1
 0  0  0  0  0  0  0  0  0  0  0  1  1  1     0  0  0  0  1  1  1  1  1  0  0  0  0
 0  1  1  1  1  0  0  0  0  1  1  0  0  1     0  0  0  0  0  0  0  1  1  1  1  1  0
 0  0  0  0  1  1  1  1  0  0  0  0  0  0     0  0  1  1  1  1  0  0  0  0  0  0  0
```

Confección de la base de datos

En principio, la base de datos simplemente debe

- Guardar, para cada *frame*
  - las características obtenidas
  - un identificador de la canción de la cual provino
  - el número de *frame* dentro de la canción.
- Permitir buscar el identificador de canción y el número de *frame* a partir de una característica, o informar si la característica buscada no se encuentra en la base de datos.

Así, cuando se desea identificar una música desconocida, se calculan las características de cada *frame*, se obtienen los identificadores de la base, y se opera con ellos de algún modo razonable para devolver la (o las) canciones más probables.

Ahora bien, más allá de este trabajo, es importante que este tipo de algoritmos escalen a bases de datos grandes y funcione rápido; para eso, es crucial que la la base de datos aproveche bien el espacio en memoria y permita una búsqueda eficiente. Por esto, se le proveen funciones que implementan una versión sencilla de una base de datos que cumple con estas características, que deberán ser entendidos.

Para confeccionar la base de datos, se utilizará la función *generar_DB*. Esta función requiere que el alumno haya definido otra función que, dado un archivo de audio como entrada, lo pre-procese, analice y extraiga su huella digital acústica en forma automática.

Ejercicio 9)

Encapsule su algoritmo de extracción de huellas acústicas en una función llamada `generar_DB` que reciba como entrada el path del archivo de audio y devuelva su huella digital acústica.

```
"""
    generar_huella(fname)

Devuelve la huella del audio en el archivo `fname`.

Tiene que estar muestreado a 44100 Hz.
"""
function generar_huella(fname::String)
    #...

    return huella
end
```

```
addNoiseToSong (generic function with 1 method)

trimSong (generic function with 1 method)
```

```
function generar_huella(fname::String; trim = false, duration = 0, addNoise = false,
noise = 0)
    song = loadaudio(fname)
    songMono = meanAudio(song)
    if trim
        songMono = trimSong(songMono, duration)
    end;
    if addNoise
        songMono = addNoiseToSong(songMono, noise)
    end;
    songMonoFiltHamming = conv(songMono,h_hamming)
    songSubratedHamming = subRate(songMonoFiltHamming)
    window = hamming(2048)
    S = calcS(songSubratedHamming; window = window, ov=0.9)
    bands = exp.(range(log(300); stop=log(2000), length=22))
    freq = [i for i in range(0; stop = 2756.25, length = Int(2048/2))]
    E = calcE(S, bands, freq)
    H = calcH(E)
    return H
end;
```

Ejercicio 10)

Observe que la cantidad de elementos a guardar en la base de datos se incrementa conforme la longitud de las ventanas del espectrograma inicial disminuye, o el solapamiento entre ventanas se incrementa. Determine el solapamiento entre ventanas del espectrograma para obtener una densidad de aproximadamente 25 elementos por segundo y utilice este valor para el ejercicio siguiente.

Para el calculo del overlap, nos basamos en la documentacion de la funcion specgram de matlab.

> If `x` is a signal of length N_x, then `s` has k columns, where
>
> - $k = \lfloor (N_x - \text{noverlap})/(\text{window} - \text{noverlap}) \rfloor$ if `window` is a scalar.
> - $k = \lfloor (N_x - \text{noverlap})/(\text{length}(\text{window}) - \text{noverlap}) \rfloor$ if `window` is a vector.
>
> If `x` is real and `nfft` is even, then `s` has `(nfft/2 + 1)` rows.
>
> If `x` is real and `nfft` is odd, then `s` has `(nfft + 1)/2` rows.
>
> If `x` is complex, then `s` has `nfft` rows.

```
calcOverlap (generic function with 1 method)
```

```
function calcOverlap(x, fps, window)
    ns = length(x)
    duration = 8*ns/sr
    nw = length(window)
    f = duration * fps
    no = (nw * f - ns)/(f-1)
    return no/nw # Para calcular el porcentaje
end
```

```
0.8959170024040213
```

Se redondeará y se tomará como overlap 0.9

Ejercicio 11)

Ejecute la función `generar_DB` para confeccionar la base de datos completa de su lista de canciones. Utilice al menos 40 canciones para llenar la base de datos. Puede usar la lista de canciones provista, y/o usar una lista de canciones propia. (Recuerde verificar que la frecuencia de muestreo de sus canciones sea de 44100 Hz).

```
String["'Filthy Rich' Since I.ogg", "4 Non Blondes - What's Up.ogg", "Aerosmith - Craz
```

'Filthy Rich' Since I.ogg

0:00 / 4:05

```
db = generar_DB(songs; dir=songsdir);
```

Test del algoritmo

En esta etapa final, pondremos a prueba la eficacia del algoritmo completo de reconocimiento. El procedimiento consistirá en obtener el porcentaje de aciertos del método al reconocer segmentos de audio, los cuales serán sometidos a distintas distorsiones.

Para esto se suministra la función *query_DB*, la cual recibe como entradas la base de datos y la huella acústica del segmento de audio a reconocer, y devuelve el ID del la canción que mejor coincide con la huella. Para entender cómo opera esta función, lea el Apéndice y el código.

Ejercicio 12)

Evalúe la tasa de aciertos del algoritmo identificando segmentos de duración T con tiempo inicial elegido al azar de canciones elegidas al azar (vea la función `rand`). Las canciones deberán ser las mismas que utilizó para confeccionar la base de datos. Realice la evaluación para 50 segmentos con duración T entre 5, 10 y 20 segundos cada vez (150 evaluaciones en total) obteniendo la tasa de aciertos en cada caso.

Funcion para testear con distintas duraciones, con posibilidad de cambiar la cantidad y agregar ruido.

```
function test(durations; cant = 50, noises = [nothing])
    matchs = zeros(length(durations), length(noises))
    addNoise = noises != [nothing]
    for nIndex in 1:length(noises)
        for dIndex in 1:length(durations)
            pos = 0
            for i in 1:cant
                songid = rand(1:length(songs))
                hSong = generar_huella(joinpath(songsdir, songs[songid]); trim =
true, duration = durations[dIndex], addNoise = addNoise, noise = noises[nIndex])
```

```
                queryid = Int(query_DB(db, hSong))
                if songid == queryid
                    pos += 1
                end;
            end;
            matchs[dIndex, nIndex] = pos / cant;
        end;
    end;
    return matchs
end;
```

Resultados de testear 50 muestras de 5s, 10s y 20s sin ruido.

```
results = 3×1 Array{Float64,2}:
          1.0
          1.0
          1.0
```

```
results = test([5,10,20])
```

Como se observa, se obtuvo una tasa de 100% de acierto.

Ejercicio 13)

Repita el ejercicio 12 sumando ruido a los segmentos de audio. Utilice la función `randn` para generar las muestras de ruido. Evalúe tasa de aciertos para $SNR = 0dB$, $10dB$ y $20dB$, mostrando sus resultados en una tabla para 9 combinaciones de longitud temporal y ruido. Nota: $SNR = 10log_{10}(P_X/P_N)$ donde P_X es la potencia media de la señal sin ruido, y P_N es la potencia media del ruido sumado a la señal. Para el cálculo de la potencia media puede utilizar la función `var` , que estima la varianza de una señal, ya que las señales de audio no deberían componente continua o valor medio.

Resultados de testear 50 muestras de 5s, 10s y 20s con 0dB, 10dB y 20dB de ruido.

Las filas corresponden a la duración y las columnas al ruido.

```
resultsWithNoise = 3×3 Array{Float64,2}:
                    0.9   0.92  1.0
                    0.92  1.0   1.0
                    0.96  1.0   1.0
```

```
resultsWithNoise = test([5,10,20]; noises = [0,10,20])
```

Como se observa, se obtuvo una alta tasa de acierto incluso al haberle agregado ruido.

Ejercicio 14) *(OPTATIVO)*

Reproduzca y grabe un segmento de alguna de las canciones e intente reconocerla con su algoritmo. Puede utilizar dispositivos como el micrófono de su PC, su teléfono celular, una radio, etc. Comente los resultados obtenidos así como las condiciones de ruido de fondo y los dispositivos utilizados. (Recuerde que su función debe recibir audios a 44100 Hz.)

Ejercicio 15) *(OPTATIVO, SOLO PARA LOS MÁS ENTUSIASTAS)*

Repita el ejercicio 12 para T=10 solamente, afectando los segmentos con otro tipos de distorsiones elegidas entre las siguientes:

- **Saturación**
- **Cuantización**
- **Cambio de velocidad**
- **Ecualización**

Si elige saturación y/o cuantización, muestre el efecto que ocasiona sobre el audio mediante un espectrograma. Justifique si estas distorsiones pueden considerarse o no sistemas LTI.

Ejercicio 16) *(OPTATIVO; SÓLO PARA LOS MÁS OBSESIVOS)*

Verifique cómo cambia la tasa de aciertos cuando:

- **cambia el solapamiento**
- **incrementa el solapamiento sólo al momento de identificar pero no al armar la base de datos**
- **cambia la longitud de la ventana manteniendo la tasa de frames por segundo**
- **cambia el algoritmo de extracción de características por algún otro que se le ocurra**

Apéndice

La estructura de la base de datos

La base de datos es un vector de 2^{20} elementos, cada uno de los cuales es un vector de tamaño dinámico. Para cada frame de la huella acústica, se genera un valor a guardar en estos vectores de tamaño dinámico. El índice en el que se guarda cada valor se obtiene pasando a decimal el número binario conformado por las características del frame en cuestión. Note que la cantidad de elementos del vector de la base, 2^{20} se debe a que cada frame de la huella acústica posee 20 elementos binarios.

Cada valor a guardar es un objeto de tipo `FrameID`, que consiste de un entero de 32 bits sin signo, que determina el número de frame, y un entero de 16 bits sin signo que guarda el índice de la canción. Observe que el máximo número de canciones distintas que se podrán almacenar en esta tabla es 2^{16}. Obserbe también que múltiples FrameIDs de distintos frames pueden requerir ser guardados en la misma posición por tener la misma huella; por esto, cada elemento es un vector que puede crecer según se requiera.

Esta implementación puede ser optimizada pero es un balance razonable entre eficiencia y simpleza, suficiente para este trabajo.

La búsqueda de la canción más probable

La búsqueda en la tabla *hash* se realiza partiendo de una huella acústica de entrada y devuelve el ID de la canción más probable a la cual corresponde como salida.

Para cada *frame* de la huella se obtiene el índice del vector de la base de datos que debe consultarse, pasando la característica de cada *frame* de binario a decimal, del modo inverso que cuando se almacenan elementos en la base, y se extraen todos los elementos almacenados en esa posición. Cada uno de estos elementos posee el ID de la canción a la cual corresponde y el número del frame que le correspondía originalmente.

Una forma sencilla de decidir a qué canción corresponde la huella podría ser elegir, de todos los elementos que se extrajeron, el ID que aparezca el mayor número de veces.

Un refinamiento al criterio anterior es, dado que para cada ID se dispone del número de *frame* del cual se extrajo, quedarse con la mayor cantidad de ID dentro de un intervalo de *frames* igual a la longitud de la huella que se está consultando. Esto es lo que está implementado en la función *query_DB*.

Por ejemplo, supongamos que se realiza un query a la base de datos con una huella que posee una longitud de 5 frames y se obtienen los siguientes 7 elementos:

Elemento	ID	Frame
1	3	5
2	3	7
3	3	8
4	7	10
5	7	19
6	7	25
7	7	28

Bajo el primer criterio se declararía ganador al ID 7 dado que aparece mayor cantidad de veces, pero bajo el segundo criterio se decide por el ID 3, debido a que en el intervalo de 5 frames que tiene la huella de entrada el ID 7 aparece como máximo 2 veces y el ID 3 aparece 3 veces.

Bibliografía

[1] Eric Allamanche, Jürgen Herre, Oliver Hellmuth, Bernhard Fröba, Thrsten Kastner, y Markus Cremer. Content-based identification of audio material using mpeg-7 low level description. In *ISMIR*. 2001.

[2] Shumeet Baluja y Michele Covell. Audio fingerprinting: combining computer vision & data stream processing. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volumen 2, página 0. IEEE, 2007.

[3] Eloi Batlle, Jaume Masip, y Enric Guaus. Automatic song identification in noisy broadcast audio. In *IASTED International Conference on Signal and Image Processing*, volumen 2, página 2. 2002.

[4] Thomas L Blum, Douglas F Keislar, James A Wheaton, y Erling H Wold. Method and article of manufacture for content-based analysis, storage, retrieval, and segmentation of audio information. 29 1999. US Patent 5,918,223.

[5] Jaap Haitsma y Ton Kalker. A highly robust audio fingerprinting system with an efficient search strategy. *Journal of New Music Research*, 32(2):211–221, 2003.

[6] Akisato Kimura, Kunio Kashino, Takayuki Kurozumi, y Hiroshi Murase. Very quick audio searching: introducing global pruning to the time-series active search. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*, volumen 3, pages 1429–1432. IEEE, 2001.

[7] Yu Liu, Kiho Cho, Hwan Sik Yun, Jong Won Shin, y Nam Soo Kim. Dct based multiple hashing technique for robust audio fingerprinting. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 61–64. IEEE, 2009.

[8] Avery Wang. The shazam music recognition service. *Communications of the ACM*, 49(8):44–48, 2006.

[9] Avery Wang et al. An industrial strength audio search algorithm. In *Ismir*, volumen 2003, pages 7–13. Washington, DC, 2003.