SolucionP2Habla_BlazquezRamirez

November 26, 2023

1 Herramientas Básica de Procesamiento Digital de Audio

1.1 Ejercicio 1: Creación y Manipulación Básicas de Señales

1.1.1 Actividades

1. Genere una onda sinusoidal de duración 1s., amplitud A = 1 y frecuencia de muestreo fs = 1000Hz. Varíe la frecuencia fundamental del tono f0 = [100, 200, 300]Hz y reproduzca los comandos usando una primitiva adecuada en Julia.

Para generar una función seno, tenemos que saber que parámetros describen una función seno. Esos parámetros son la frecuencia fundamental y la fase del seno.

Para la fase, trabajaremos con fase igual a 0, en todas las funciones seno de esta actividad. En la frecuencia, probaremos con varias, 100, 200 y 300 Hz.

Declararemos una función, dependiente de t, que será el tiempo, y dependiente de la frecuencia. Como queremos una señal en digital, tenemos que muestrearla. En el tiempo, le pasaremos todos los valores del tiempo. Estos valores irán desde t = 1 hasta 2*fs. Y para muestrearla, haremos que la función coja valores solo en las frecuencias de muestreo. Esto será, generar la señal desde 0 hasta dos veces la fs.

[ ]:
```
f(f0,fs, A) = A * sin.((2*pi*(0:2*fs)*f0)/fs)
```

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

Ahora que ya tenemos nuestra función, vamos a escuhar todas nuestras frecuencias centrales.

[ ]:
```
using Plots
using Sound
```

[ ]:
```
A = 1;
fs = 1000;
f0 = [100,200,300];
for i in f0
    f_sin = f(i,fs,A)
    sound(f_sin,fs);
end
```

```
ALSA lib pcm.c:8559:(snd_pcm_recover) underrun occurred
ALSA lib pcm.c:8559:(snd_pcm_recover) underrun occurred
 Warning: libportaudio: Output underflowed
```

```
 @ PortAudio
/home/javiblzqz/.julia/packages/PortAudio/HNBv4/src/PortAudio.jl:99
ALSA lib pcm.c:8559:(snd_pcm_recover) underrun occurred
 Warning: libportaudio: Output underflowed
 @ PortAudio
/home/javiblzqz/.julia/packages/PortAudio/HNBv4/src/PortAudio.jl:99
 Warning: libportaudio: Output underflowed
 @ PortAudio
/home/javiblzqz/.julia/packages/PortAudio/HNBv4/src/PortAudio.jl:99
ALSA lib pcm.c:8559:(snd_pcm_recover) underrun occurred
 Warning: libportaudio: Output underflowed
 @ PortAudio
/home/javiblzqz/.julia/packages/PortAudio/HNBv4/src/PortAudio.jl:99
ALSA lib pcm.c:8559:(snd_pcm_recover) underrun occurred
 Warning: libportaudio: Output underflowed
 @ PortAudio
/home/javiblzqz/.julia/packages/PortAudio/HNBv4/src/PortAudio.jl:99
ALSA lib pcm.c:8559:(snd_pcm_recover) underrun occurred
ALSA lib pcm.c:8559:(snd_pcm_recover) underrun occurred
 Warning: libportaudio: Output underflowed
 @ PortAudio
/home/javiblzqz/.julia/packages/PortAudio/HNBv4/src/PortAudio.jl:99
```

Vemos como la frecuencia de las señales aumenta, tanto gráficamente como acústicamente. La función seno con una frecuencia de 300Hz, genera un tono a 300Hz, que, es más aguda que el tono de 100Hz.

2. Represente la onda sinusoidal del punto 1 para f0 = [5, 10] Hz, compruebe gráficamente el valor del periodo (T0). Represente también el módulo de la FFT y compruebe el valor de f0

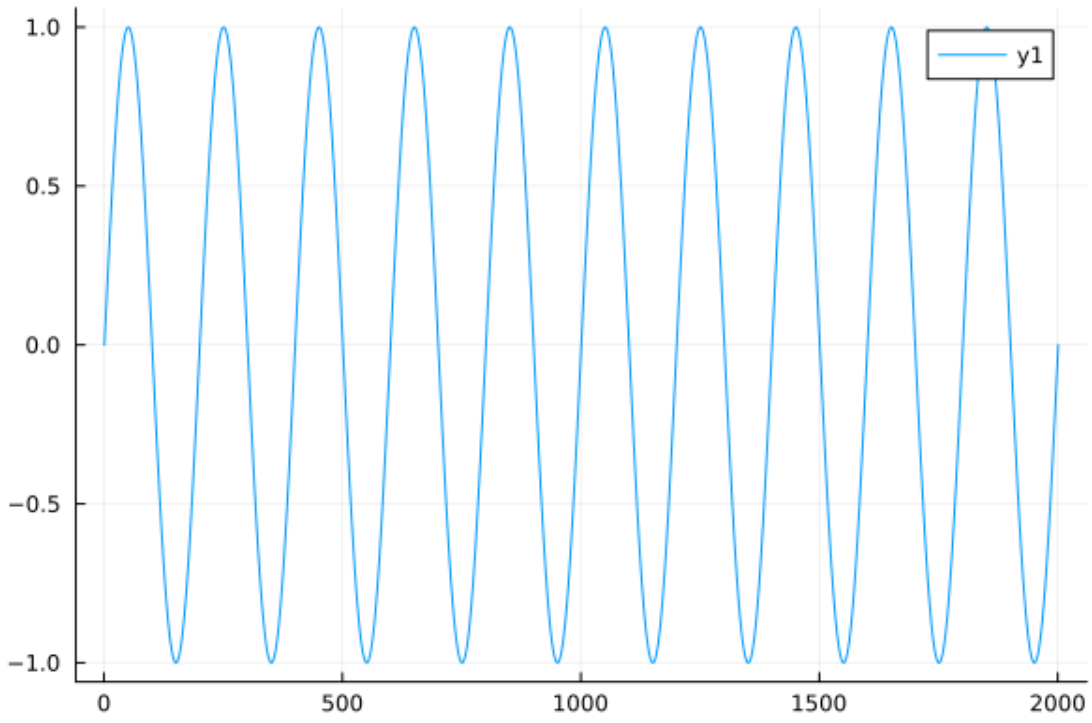
Para representar la señal con otras frecuencias, solo tenemos que llamar a la función que hemos creado en el punto anterior.

Primero la creamos con f0 = 5Hz.

[ ]:
```
sin5 = f(5,fs,A);
```

Ya tenemos creada la señal, ahora grafiquémosla.

[ ]:
```
plot(sin5)
```



Antes de ver cual es el periodo en la gráfica. Calculemos este periodo de forma teórica, para ver si coincide.

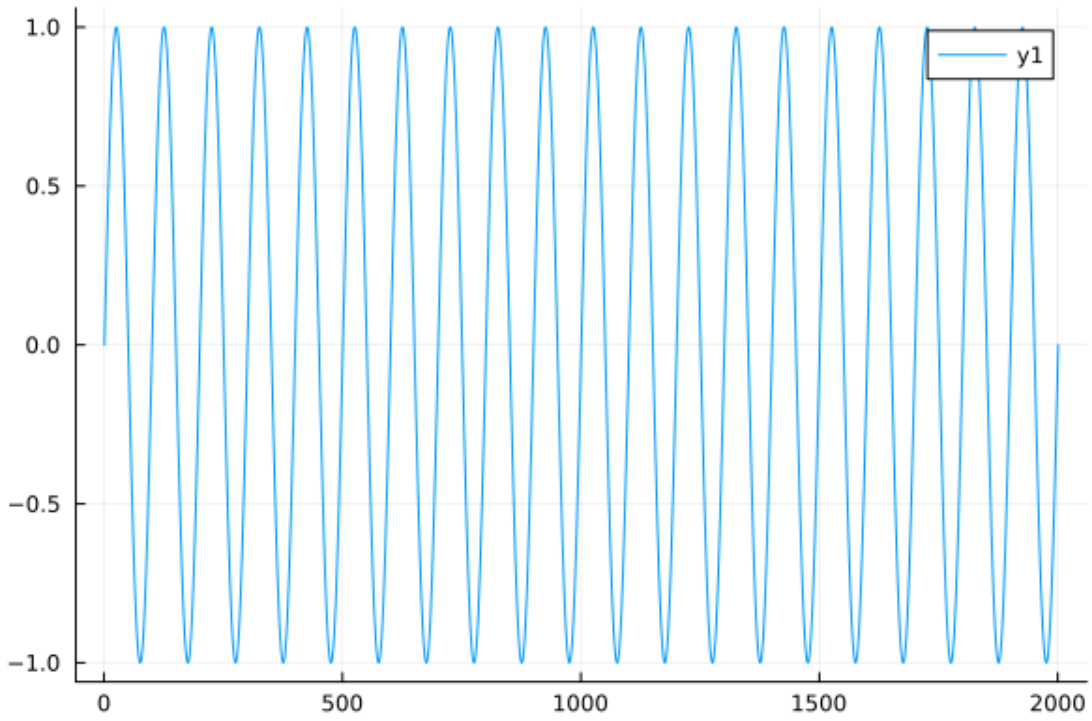
Sabemos que la frecuencia, es cuantos ciclos completa la señal en una unidad de tiempo, y que el periodo es, cuanto tarda un ciclo en completarse.

Si tenemos una frecuncia de 5Hz, quiere decir que, en un segundo completa 5 ciclos. El periodo es la inversa de la frecuencia. Tenemos que $1/5 = 0.2$ segundos en completar un ciclo.

Comparemos con lo obtenido, y efectivamente, ocurre así. El periodo de esta señal es 200ms, es decir, 0.2 segundos.

Ahora con la función de 10Hz. Seguiremos el mismo camino que con la frecuencia anterior.

[ ]:
```
sin10 = f(10,fs,A);
plot(sin10)
```



Tenemos que nuestro periodo ahora es 100ms, 0.1 segundos. El cual también coincide, si lo calculamos de forma teórica. $1/f0 = 1/10 = 0.1$

Calculemos la FFT de estas dos señales, y lo que nos marque las gráficas, serán las frecuencias centrales.

[ ]:
```
using FFTW
```

[ ]:
```
t0 = 0;
tmax = 2;
t = t0:1/fs:tmax;

F5 = fftshift(fft(sin5))
freqsF5 = fftshift(fftfreq(length(t), fs))
```

```
-499.7501249375312:0.49975012493753124:499.7501249375312
```

Antes de dibujar la señal y ver su módulo, trabajaremos con ella. Primero, de la Transformada de Fourier (TF) de la señal, nos quedaremos solo con la parte positiva. Podemos hacer esto, porque sabemos que la TF de una señal seno es simétrica par, es decir, respecto a y=0.

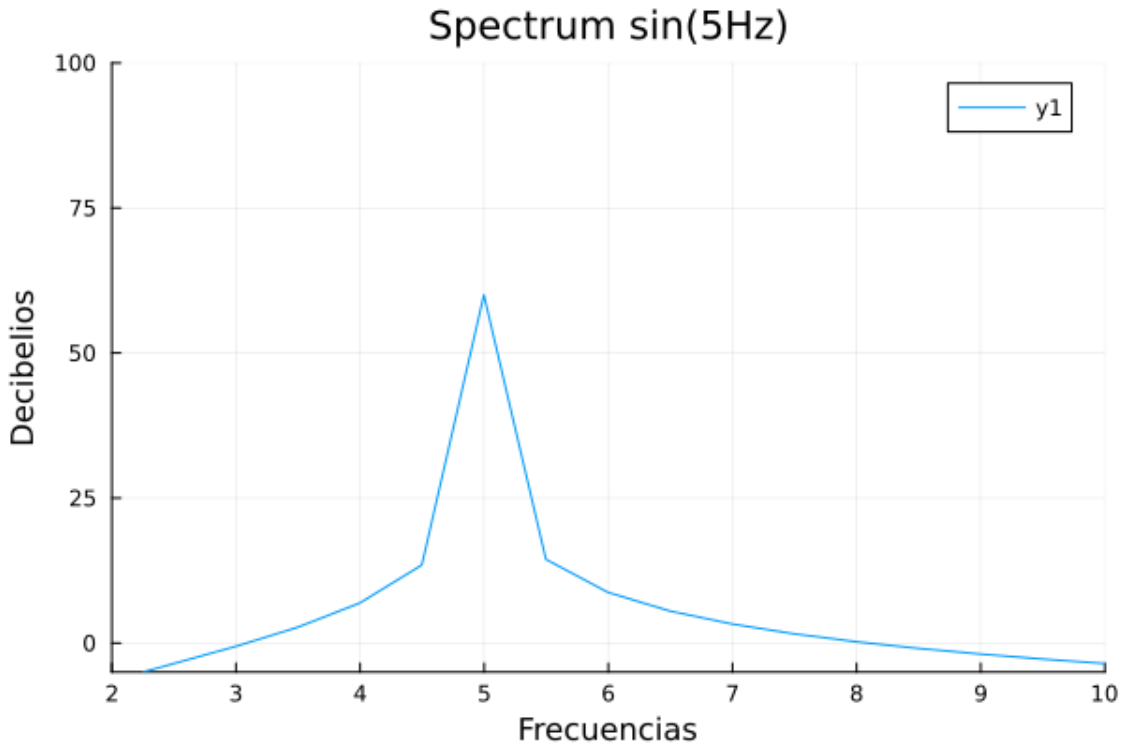
Podemos hacerlo de dos maneras, o eliminando la primera mitad de la señal, o al dibujar, permitir que solo dibuje a partir del 0.

Lo haremos de la segunda manera, esto sería una ventana rectangular ideal.

Si lo hicieramos de otra manera, para poder aplicar una ventana diferente a la rectangular ideal,

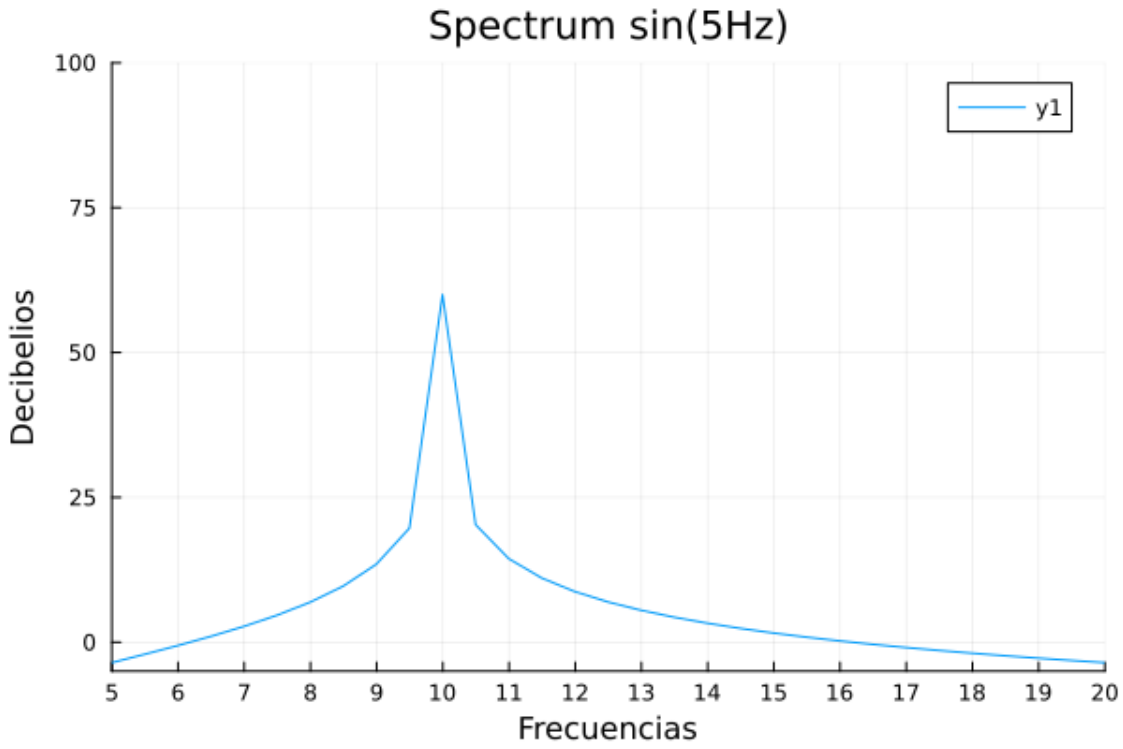
podemos ayudarnos de otras librerías, o también, programarlas nosotros mismos siguiendo las fórmulas de esas ventanas.

[ ]:
```
plot(freqsF5, 20*log10.(abs.(F5)), title = "Spectrum sin(5Hz)", xlim=(2,␣
↪10),ylim=(-5,100), xticks=0:1:20, xlabel="Frecuencias",ylabel="Decibelios")
```



Esta Transformada Rápida de Fourier (Fast Fourier Transform, FFT) la aplicaremos también a nuestra otra señal, la función seno con frecuencia 10Hz.

[ ]:
```
F10 = fftshift(fft(sin10))
freqsF10 = fftshift(fftfreq(length(t), fs))

plot(freqsF10, 20*log10.(abs.(F10)), title = "Spectrum sin(5Hz)", xlim=(5,␣
↪20),ylim=(-5,100), xticks=0:1:20, xlabel="Frecuencias",ylabel="Decibelios")
```



Estas frecuencias que obtenemos en la gráfica son las reales, son las frecuencias reales de nuestras señales, sino fueran calculadas así, en el eje X no tendríamos las frecuencias, sino el orden de las muestras.

Sabemos, teóricamente, que, la TF de una función seno, en el eje positivo de las frecuencias, es una delta centrada en la frecuencia central del seno, 5Hz y 10Hz, respectivamente. El valor de f0 según estas gráficas, son las que hemos comentado, 5Hz y 10Hz.

3.Represente el módulo de la FFT de la señal ejemploEj3T4.mat que fue muestreada a fs = 11000 Hz, identifique f0 y y las frecuencias formantes.

Lo primero que debemos hacer es poder leer la señal del archivo ejemploEj3T4.mat. Para ello, nos ayudaremos de una librería MAT.jl. Con la ayuda de esta librería, podemos hacer uso de varias funciones, abrir el archivo, escribir en él, leer todas las variables, etc.

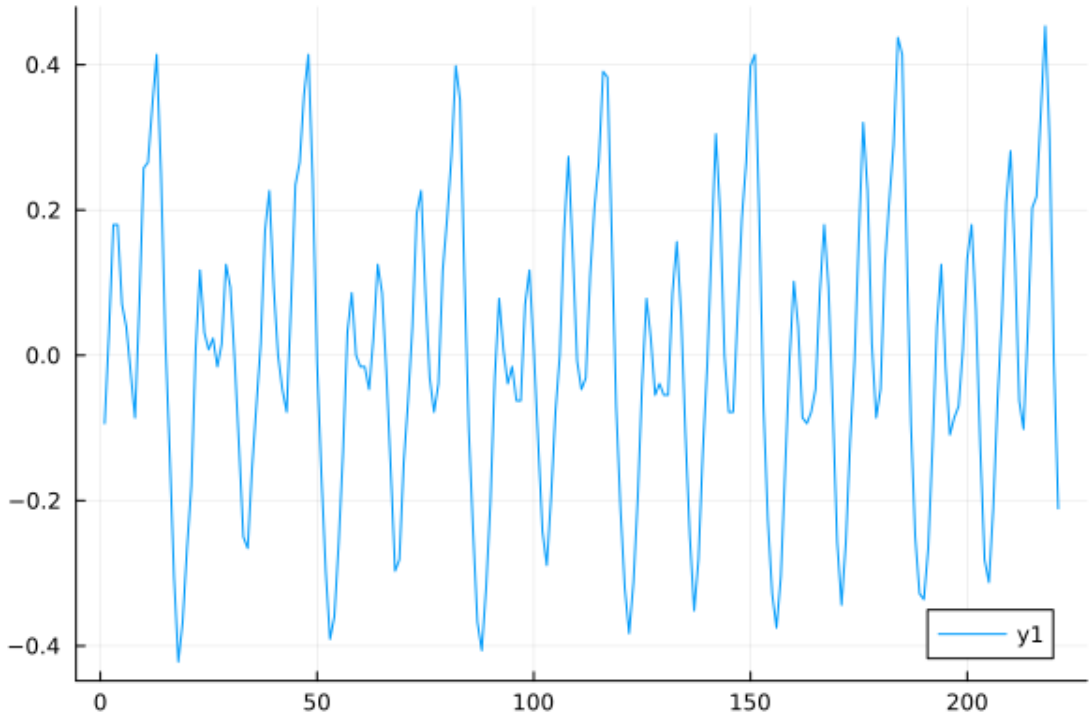
Haremos uso de la función "matread()", que nos permitirá leer el archivo, nos devuelve un diccionario, con las claves siendo los nombres de las variables, y su valor.

[ ]:
```
using MAT
```

[ ]:
```
fs = 11000;
vars = matread("ejemploEj3T4.mat")
```

```
Dict{String, Any} with 1 entry:
  "trama3" => [-0.09375; 0.03125; … ; -0.0078125; -0.210938;;]
```

[ ]:
```
y = vec(vars["trama3"])
plot(y)
```



Tenemos guardado la señal en la en la variable "y" Para representar el módulo de la FFT, haremos lo de antes, pero en vez de con una función seno, con otra diferente. Sabemos que esta señal fue muestreada con una frecuencia de 11kHz.

[ ]:
```
function fftSignal(y, fs)
    """
    Función que calcula, a partir de la señal de entrada, y, y la frecuencia de␣
 ↪muestreo, fs,
    nos devuelve la FFT de la señal, y las frecuencias con respecto a las␣
 ↪muestras.
    """
    N = length(y) - 1;
    Ts = 1 / (1.1 * N);
    t0 = 0;
    tmax = t0 + N * Ts;
    t = t0:Ts:tmax;
    F = fft(y) |> fftshift
    freqs = fftfreq(length(t), fs) |> fftshift
    return F,freqs
end
```

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

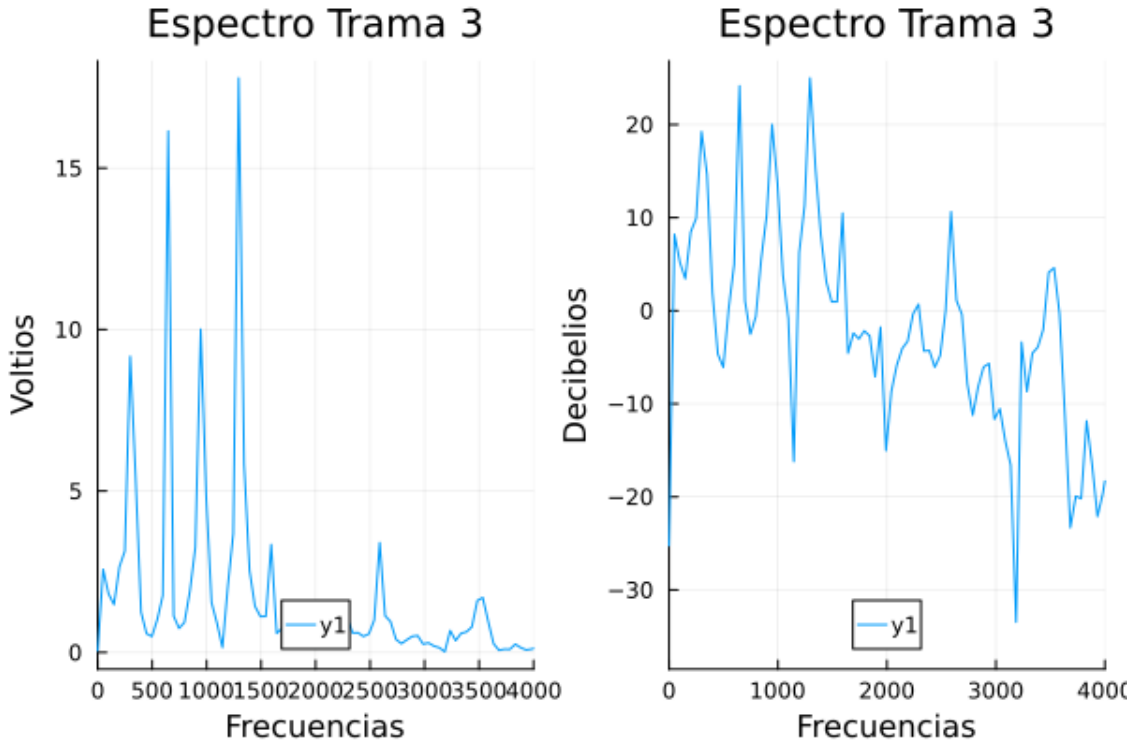
Creamos una función para no repetir este código durante todo la práctica.

El cálculo de la Transformada de Fourier con sus frecuencias correspondientes.

[ ]:
```
Y, freqsF = fftSignal(y,fs);
```

Ahora, dibujemos el espectro de esta señal.

[ ]:
```
voltiosPlot = plot(freqsF, abs.(Y), title = "Espectro Trama␣
 ↪3",xlim=(0,4000),xticks=(0:500:4000),xlabel="Frecuencias",ylabel="Voltios")
dbPlot = plot(freqsF, 20*log10.(abs.(Y)), title = "Espectro Trama␣
 ↪3",xlim=(0,4000),xticks=(0:1000:
 ↪4000),xlabel="Frecuencias",ylabel="Decibelios")
plot(voltiosPlot,dbPlot,legend=:bottom)
```



Ahora tenemos dos espectros diferentes, uno basado en que la amplitud está medida en Voltios, y la otra, en escala de decibelios. Nos interesa saber ambas, la primera es porque, queremos ver que frecuencias tienen mayor amplitud en cuanto a voltaje se refiere, para saber cuáles son las principales. La amplitud en decibelios es importante, porque, el cuerpo humano, especialmente el oído, siguen escalas logarítmicas, lo que nos ayuda a entender mejor como se comporta el sonido.

Vemos que ambas envolventes de los espectros, tienen los mismos máximos locales en las mismas frecuencias, esto nos ayuda, porque podemos usar ambas para analizar espectros.

Al igual que antes, estas frecuencias están calculadas, y son las reales. Ahora vemos que no hay tonos puros, sino que en el espectro de frecuencias está repartida por todo el espectro.

La forma del espectro nos dice cual es la f0, y sus armónicos.

Los armónicos, en la teoría o en casos experimentales ideales, se producen en múltiplos de la f0. En este caso, la f0 vemos que es igual a 300Hz. Si esta señal fuera ideal, tenemos que los armónicos estarían en torno a $2f0 = 600$, 3f0=900, etc.

Viendo el espectro, vemos que, nuestras frecuencias armónicas estarían cerca de las ideales, la primera sería 600Hz, la segunda estaría en torno a 950Hz y la tercera y última en esta señal, la frecuencia armónica se sitúa en 1300Hz.

La frecuencia formante, en este caso es de 1300Hz.

4. Represente en el dominio temporal y en el dominio espectral los fragmentos de una señal de voz contenidos en ejemploEj4AT4.mat y ejemploEj4BT4.mat. Identifique qué fragmento es sonoro y qué fragmento es sordo (fs = 11000 Hz).

Este ejercicio tiene varios apartados, iremos despacio para completarlo.

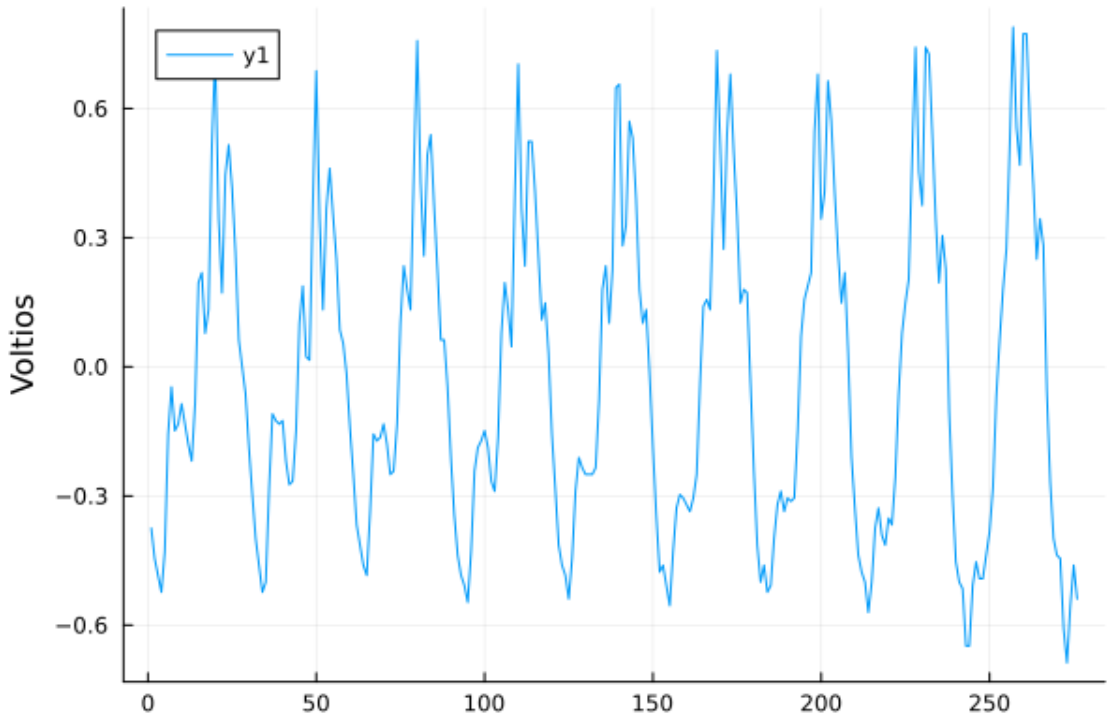
Empezaremos leyendo ambas señales, y graficándolas, para ver que forma tienen estas señales. Tras esto, podemos ver el espectro, para ver sus frecuencias, y su forma y saber si los sonidos son sordos, o sonoros.

Los sonidos sonoros tienen un espectro con frecuencias armónicas, mientras que los sordos, su espectro está lleno de ruido. Los sonidos sonoros son provocados por las cuerdas vocales, lo que hace que el espectro tenga esas frecuencias armónicas, que serán diferentes para cada persona. Eso ocurre por la característica de los sonidos, **timbre**, que está formado por la estructura armónica de las frecuencias de ese sonido. Cada persona genera un timbre diferente.

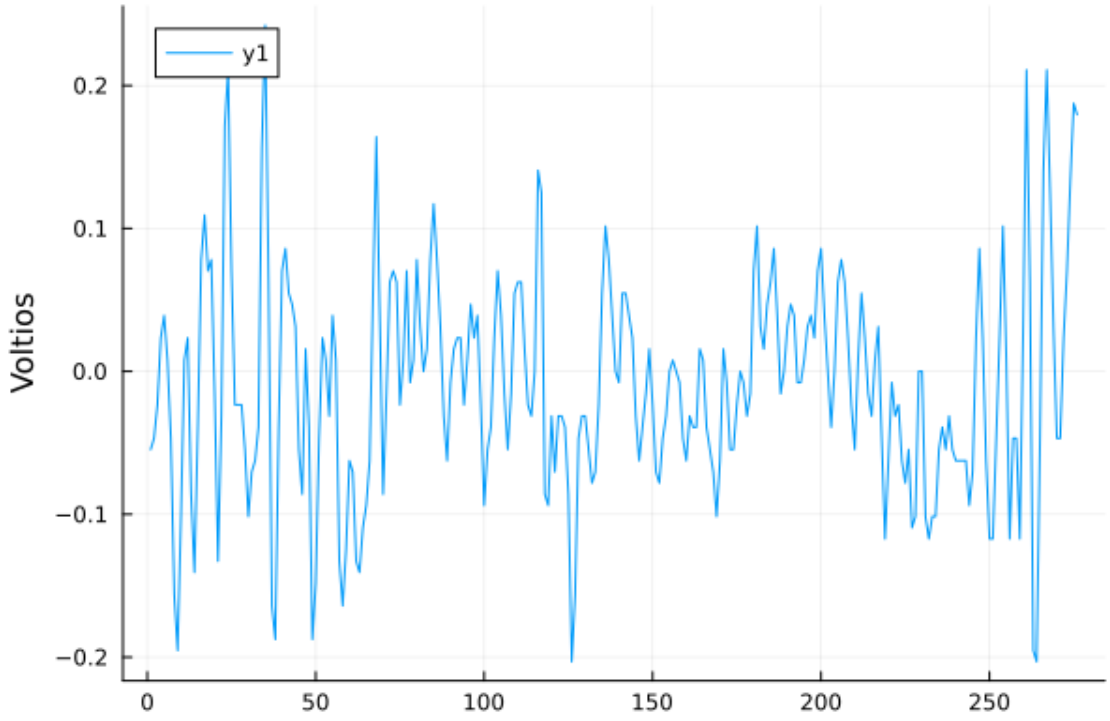
[ ]:
```
fs = 11000;

varsEj4AT4 = matread("ejemploEj4AT4.mat")
varsEj4BT4 = matread("ejemploEj4BT4.mat")

sigEJ4AT4 = vec(varsEj4AT4["trama1"]);
sigEJ4BT4 = vec(varsEj4BT4["trama2"]);
```

Ya tenemos los datos de la señal. Veámos como es su forma.

[ ]:
```
timeDomainA = plot(sigEJ4AT4,ylabel="Voltios")
```



[ ]:
```
timeDomainB = plot(sigEJ4BT4,ylabel="Voltios")
```



Vemos como estas señales son diferentes, como la primera se asemeja a una señal periódica, sin serlo. Y la segunda, no tiene una forma especial.

Escuchemos ambas señales, pero al ser ambas de duración muy corta, como es 0.276 segundos, el sonido es muy corto, y no es apreciable.

[ ]:
```
sound(sigEJ4AT4, fs)
sound(sigEJ4BT4, fs)
```

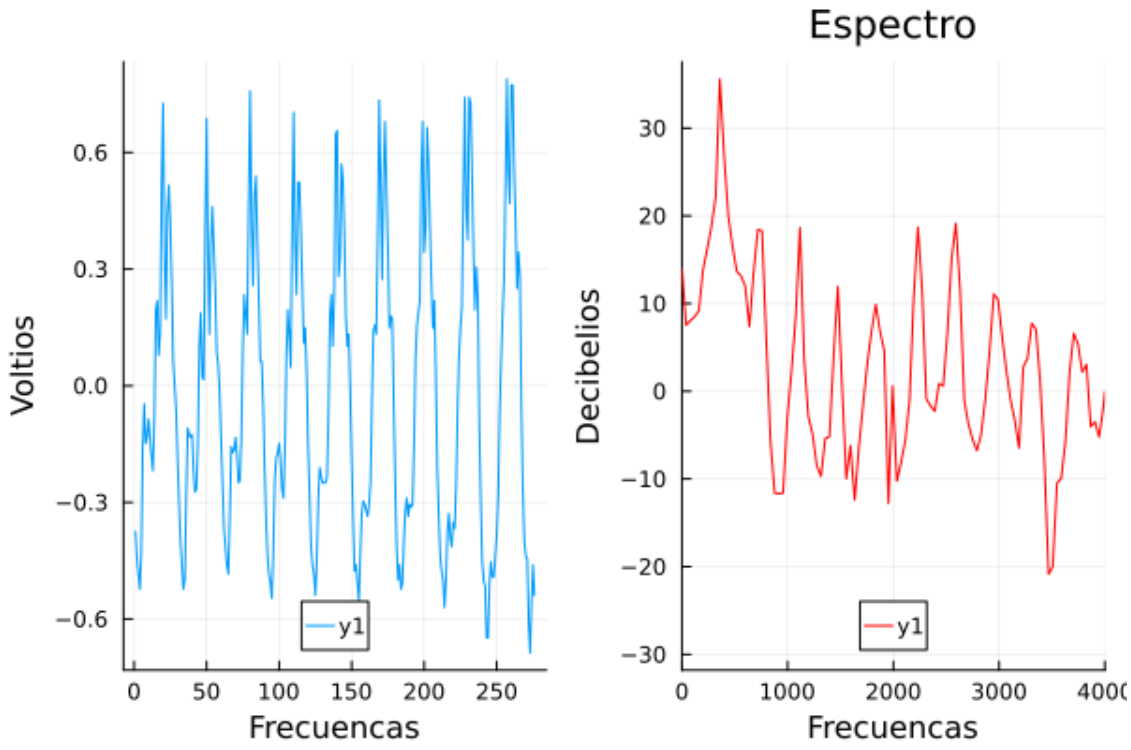
276

Representemos ahora los espectros de ambas señales. Primero, Transformada Fourier y cálculo de frecuencias de la primera señal.

[ ]:
```
FEJ4AT4, freqs4AT4 = fftSignal(sigEJ4AT4,fs);
```

Y la dibujamos para ver su espectro.

[ ]:
```
freqDomainDbA = plot(freqs4AT4, 20*log10.(abs.(FEJ4AT4)), title =␣
↪"Espectro",xlim=(0,4000),xticks=(0:1000:4000),ylabel="Decibelios",color=:Red)
plot(timeDomainA, freqDomainDbA, legend=:bottom, xlabel="Frecuencas")
```



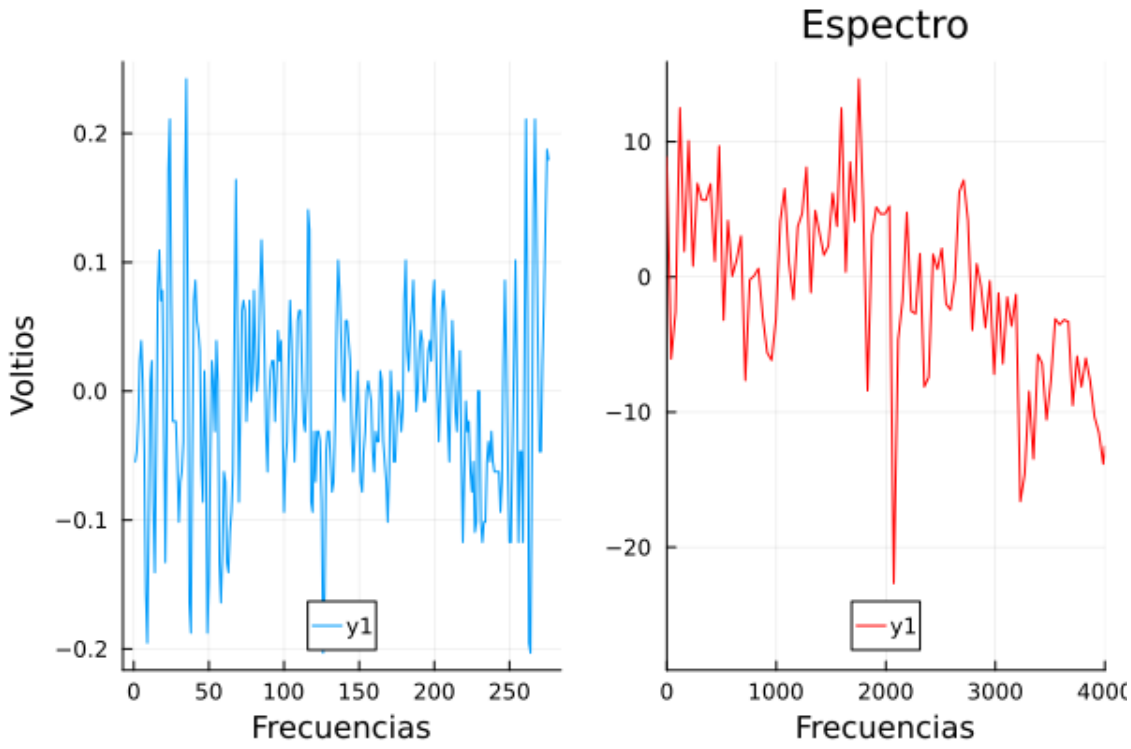
Tenemos dibujadas las señales en ambos dominios.

Y ahora la segunda señal, Transformada Fourier y cálculo de frecuencias.

[ ]:
```
FEJ4BT4, freqs4BT4 = fftSignal(sigEJ4BT4,fs);
```

Dibujemos nuestro espectro de esta señal.

[ ]:
```
freqDomainBDb = plot(freqs4BT4, 20*log10.(abs.(FEJ4BT4)), title =␣
↪"Espectro",xlim=(0,4000),xticks=(0:1000:4000),color=:red)
plot(timeDomainB,freqDomainBDb,legend=:bottom,xlabel="Frecuencias")
```



Una vez con ambas gráficas dibujadas, podemos identificar que sonido es cada uno. Tenemos que en la primera señal, el ejemploEj4AT4.mat, vemos que su espectro tiene una forma con varias frecuencias armónicas, por tanto, sabemos que pertenece a un sonido sonoro, es decir, que las cuerdas vocales han vibrado.

Para la segunda señal, ejemploEj4BT4.mat, su espectro no tiene una forma concreta, de hecho, se ve el espectro, con muchas frecuencias, es decir, la señal tiene ruido y, por lo tanto, es un sonido sordo.

1.2 Ejercicio 2: Extracción de Características de la Señal de Voz

1.2.1 Actividades

1. Visualización

a. Lea el fichero de audio confront.wav y reprodúzcalo. ¿Cuál es la frecuencia de muestreo

original?

b. Haga lo mismo al doble y a la mitad de la frecuencia de muestreo.

c. Visualice la señal de voz.

Para la resolución de esta acti es necesario que podamos leer un archivo *.WAV*. Para ello, nos ayudaremos del paquete **WAV**, el cual nos otorga diversas funciones que podemos usar para la lectura y el tratamiento de estos archivos.

[ ]:
```
using WAV
```

Una vez cargado el paquete en nuestro entorno Julia, procedemos a leer el archivo, el cual está en el mismo directorio que este *notebook*.

Nos apoyamos en la función ***wavread()***, a la que pasando el archivo a leer, nos devolverá, los datos del audio, y la frecuencia de muestreo con la que ha sido grabada.

[ ]:
```
confront,fs = wavread("confront.wav")
```

```
([0.0039215686274509665; 0.0039215686274509665; … ; 0.0039215686274509665; 0.
↪0039215686274509665;;], 11025.0f0, 0x0008, WAVChunk[WAVChunk(Symbol("fmt "),␣
↪UInt8[0x10, 0x00, 0x00, 0x00, 0x01, 0x00, 0x01, 0x00, 0x11, 0x2b, 0x00, 0x00,␣
↪0x11, 0x2b, 0x00, 0x00, 0x01, 0x00, 0x08, 0x00])])
```

Si vemos el valor de la variable *fs* obtendremos su frecuencia de muestreo.

[ ]:
```
println("El valor de la frecuencia de muestreo es: ",fs,"Hz")
```

```
El valor de la frecuencia de muestreo es: 11025.0Hz
```

Vamos a reproducir primero este audio, a su misma frecuencia de muestreo, para saber como suena y poder comparar si cambiamos esa frecuencia de muestreo.

[ ]:
```
sound(confront,fs)
```

```
19778
```

Ahora modifiquemos la frecuencia de muestreo, primero al doble de la obtenida anteriormente:

[ ]:
```
new_fs = 2*fs
sound(confront,new_fs)
```

```
19778
```

Como escuchamos, el audio ahora se ejecuta más rápido, siendo ininteligible. Esto ocurre, porque, donde antes se ejecutaba 1 muestra, ahora se ejecutan dos. Es lo que queríamos, que la frecuencia a la que se ejecutan esas muestras, sea el doble de lo que era antes.

Siguiendo esta lógica, podemos intentar predecir, lo que ocurrirá si modificamos la frecuencia de muestreo a la mitad de lo que obtuvimos en primer lugar. Lo que ocurrirá, que la señal se dilatará en el tiempo, al inverso que ocurre con la frecuencia, que se contrae.

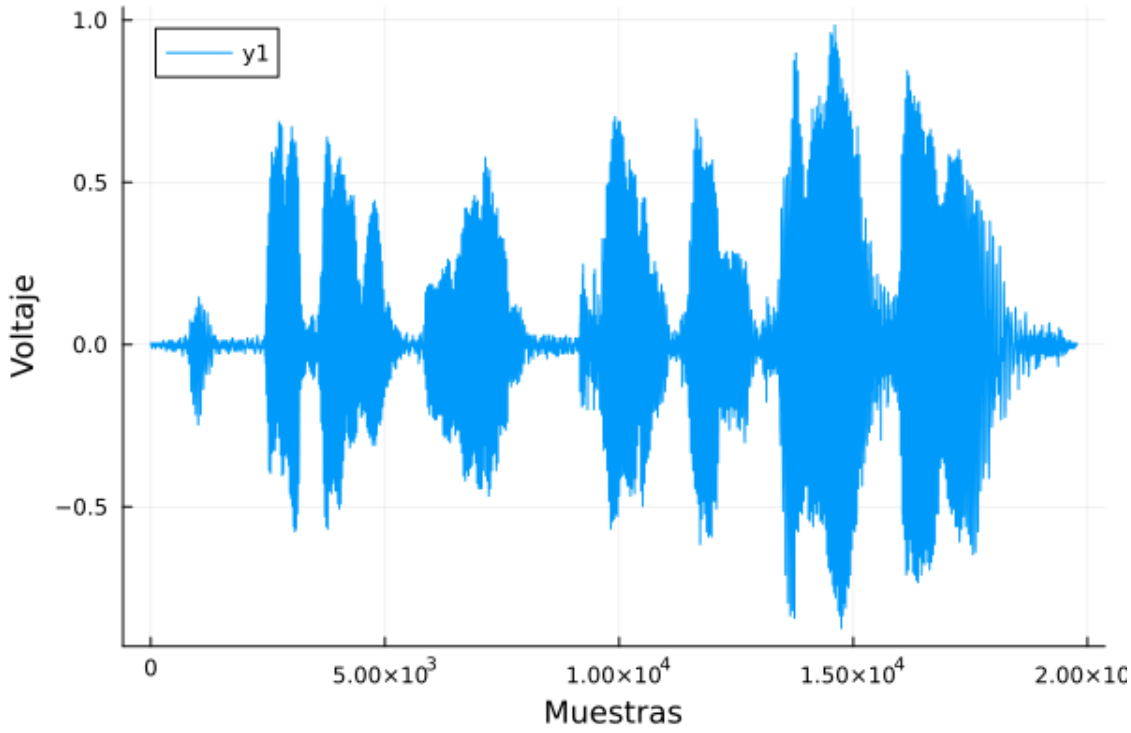
[ ]:
```
new_fs = fs/2;
sound(confront,new_fs)
```

```
19778
```

En ambos experimentos, obtenemos que la señal de voz tiene las mismas muestras. Eso está en consonancia con lo que hemos comentado antes, la cantidad de muestras no cambia, varía la velocidad a la que ejecutamos esas muestras.

Veamos la señal de voz, que forma tiene.

[ ]:
```
plot(confront,xlabel="Muestras",ylabel="Voltaje")
```



[ ]:
```
confront
```

```
19778×1 Matrix{Float64}:
 0.0039215686274509665
 0.0039215686274509665
 0.0039215686274509665
 0.0039215686274509665
 0.0039215686274509665
 0.0039215686274509665
 0.0039215686274509665
 0.0039215686274509665
 0.0039215686274509665
```

```
0.0039215686274509665

0.0039215686274509665
0.0039215686274509665
0.0039215686274509665
0.0039215686274509665
0.0039215686274509665
0.0039215686274509665
0.0039215686274509665
0.0039215686274509665
0.0039215686274509665
```

Esta forma de onda, es típica en las señales de voz. Únicamente viendo esta gráfica, podemos averiguar cuando hay silencio o cuando se está captando una señal de voz.

2.Enventanado: localización del análisis

`a. Obtenga y represente en el tiempo las ventanas rectangular y de Hamming de una duración de 20ms.`

`b. Observe sus perfiles. ¿Qué ventana introduce menos distorsión en el dominio temporal?`

`c. Calcule el espectro de frecuencia de dichas ventanas, usando la FFT y compare los módulos de los espectros de ambas ventanas representándolos lado a lado.`

`d. Varíe la longitud de las ventanas (p.ej. a 10ms y 30 ms) y observe el efecto de los espectros de las ventanas (anchura del lóbulo principal y nivel de los lóbulos secundarios).`

Todas estas ventanas, se obtienen en el dominio temporal. Estas ventanas no son más "filtros" en los que cogemos una parte de la señal, de ganancia 1. Las ventanas están parametrizadas por su función, es decir, como cogen la señal y si la modifican, y por la duración de la señal que cogen, que no será más que cuantas muestras son capaces de atravesar la señal. Estas ventanas, se centran en un tramo de la señal. Si usamos estas ventanas, durante todas los tramos disponibles, podemos permitirnos considerar que las caracteristicas de la señal son constantes.

Para obtener y representar ventanas, podemos hacer uso de un paquete ya pre-programado, **DSP.jl**. Pero podemos también a programarlos nosotros mismos. Si empezamos con la ventana rectangular, podemos programarla fácilmente. Esta ventana rectangular, es un filtro ideal de ganancia 1, coge todas las muestras de la señal en el tramo asignado, y no modifica su amplitud.

[ ]:
```
function rectWindow(signal, duration, firstSample, fs)
    """
    Le pasamos como argumentos de entrada, la señal, la duración de la ventana,␣
 ↪la primera muestra que queremos
    que esa ventana coja, y la frecuencia de muestreo, y nos devuelve esa señal␣
 ↪enventanada, a partir de la primera
    muestra deseada.
    """
```

```
    # Numero de muestras en la duración deseada, en segundos.
    numSamples = fs * duration;
    sigWindowed = Vector{Float64}(undef, 0)
    for i in firstSample:(firstSample+(fs * duration) - 1)
        j = Int(i)
        push!(sigWindowed,signal[j])
    end
    return sigWindowed
end
```

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

Esta función coge como parámetros la duración de la ventana,*duration*, en segundos, cual es el índice de la primera muestra,*firstSample*. También coge como parámetro la propia señal a **enventanar**,*signal*, en nuestro caso, señales de voz, y, por último, la frecuencia de muestreo de la señal,*fs*.

Cogeremos la primera muestra de la señal, y sumaremos tantas muestras como caben en nuestra duración de ventana. Las demás serán descartadas.

Podemos recurrir a otras primitivas de Julia, o de otros paquetes, pero de esta manera, con Julia más básico, podemos entender de mejor manera como funcionan las ventanas.

Veremos esta ventana con la señal con la que estábamos trabajando.

La función de los coeficientes de las ventanas de Hamming, es la siguiente:

$$f(x) = 0.54 - 0.46 * cos(\frac{2\pi * x}{N-1})$$

Siendo N la longitud de la ventana.

Siguiendo esta función obtenemos la función a la que debemos multiplicar el trozo de la señal para obtener la señal enventanada.

Debemos multiplicar esta señal a los coeficientes obtenidos, pero en la señal rectangular, esos coeficentes son 1, y en Hamming la amplitud se ve afectada por la ventana.

En los extremos la función se ve reducida y en el centro no se modifica.

Podemos parar a analizar los perfiles de estas ventanas para ver cuál introduce más distorsión temporal. Nosotros sabemos la forma de la función de la ventana de Hamming, su forma hace que se atenúe los valores de la señal en los extremos, y en el centro no se modifica los valores. Sin embargo, la ventana rectangular es más abrupta, es decir, coge los valores anteriores a la primera muestra como 0, y a partir de la primera muestra, su valor normal, sin importar si hay un gran salto. Estos picos provocan **MAYOR** distorsión temporal que la ventana Hamming, la cual será la que debemos elegir entre ambas si queremos menor distorsión.

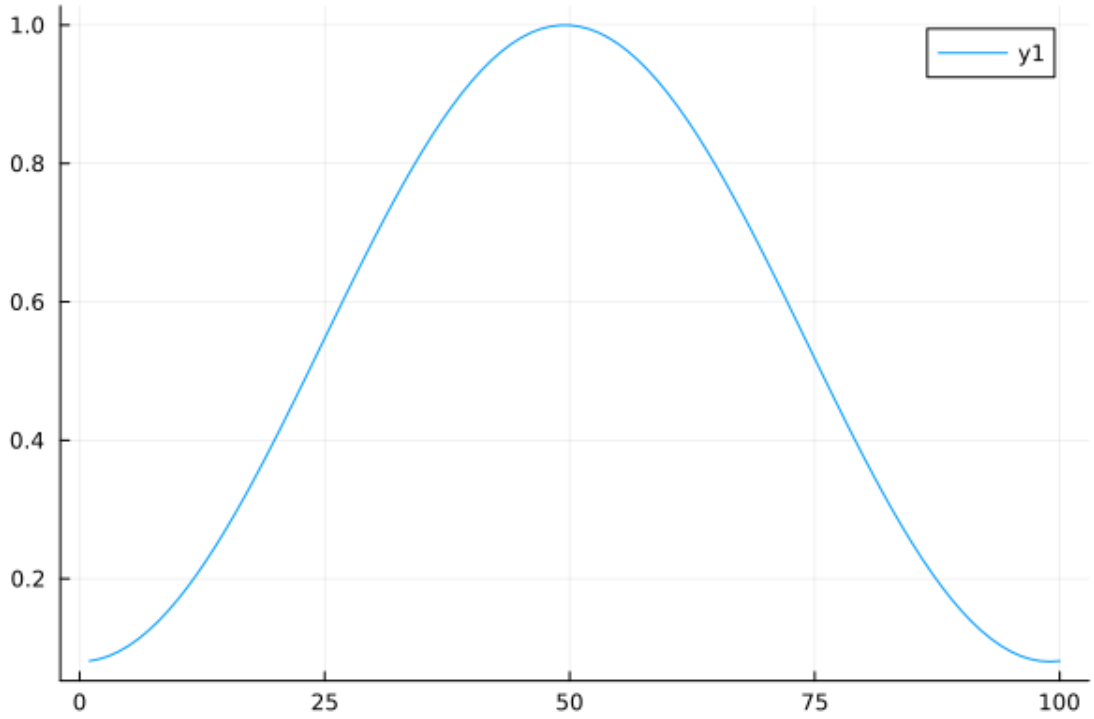
Ahora haremos la transformada de Fourier de ambas señales enventanadas, podremos ver como se comportan en frecuencia en comparación con la señal sin enventanar. Primero con la señal enventanada con la ventana rectangular.

Usaremos 220 muestras, que con esta frecuencia de muestreo, son las muestras que nos caben en 20ms.

[ ]:
```
function hamming(size)
    x = 1:1:size;
    return  0.54 .- (0.46 * cos.((2*pi*x) / (size - 1)));
end
```

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

[ ]:
```
hammingWin = hamming(100)
plot(hammingWin)
```



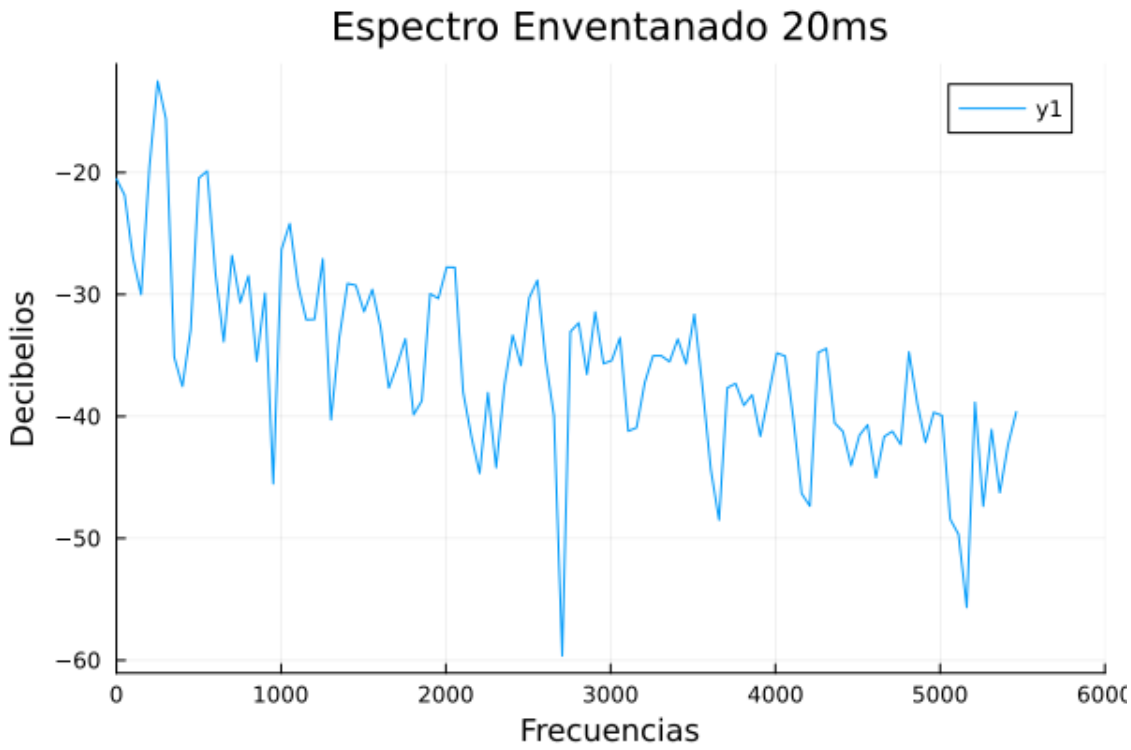
Esta es la función que viene definida por la ventana de Hamming, veamos el espectro de la función de la ventana de Hamming. Primero calcularemos su FFT, y después la representaremos.

[ ]:
```
fftHamm, hammfreqs = fftSignal(hammWin,fs)
plot(hammfreqs, 20*log10.(abs.(fftHamm)), title = "Espectro Ventana␣
↪Hamming",xlabel="Frecuencias",ylabel="Decibelios")
```

```
UndefVarError: `hammWin` not defined



Stacktrace:

 [1] top-level scope
```

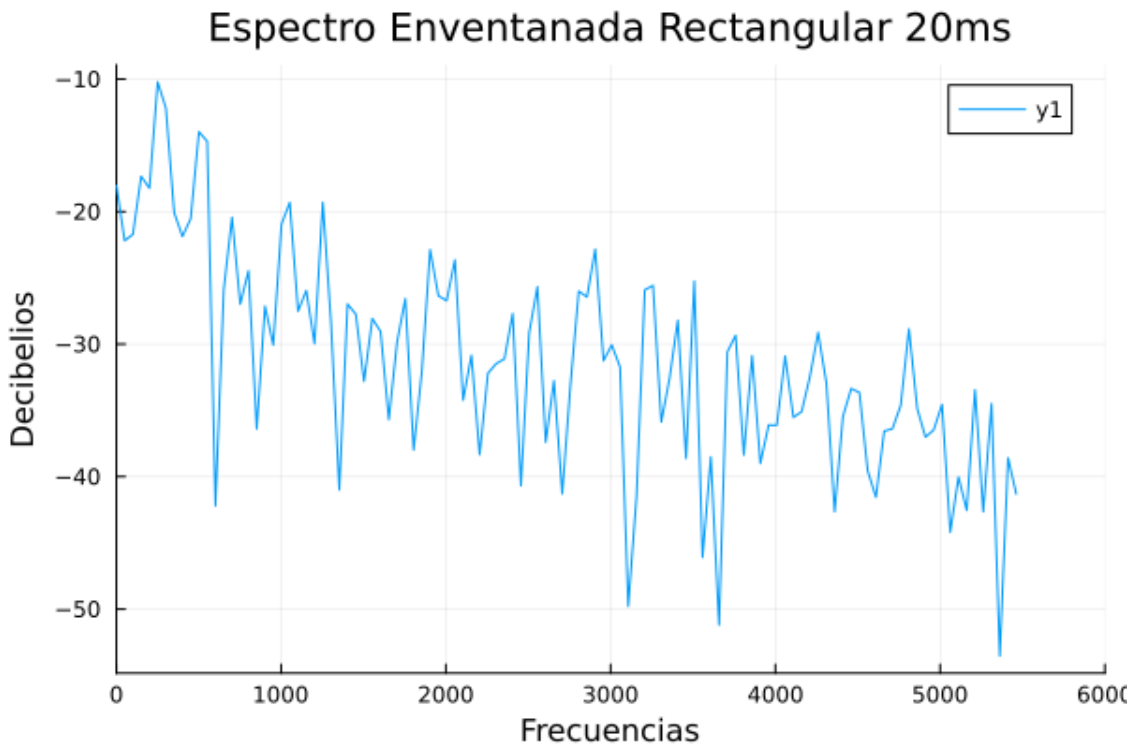
```
@ ~/Documentos/PDI/Sesion2y3/SolucionP2Habla_BlazquezRamirez.ipynb:1
```

[ ]:
```
sizeWindow = 220
hammWin = hamming(sizeWindow)
sigWindowed = confront[1:220] .* hammWin
hammSig,hammfreqs = fftSignal(sigWindowed,fs)
plot20ms = plot(hammfreqs, 20*log10.(abs.(hammSig)), title = "Espectro␣
↪Enventanado 20ms",xlim=(0,6000),xlabel="Frecuencias",ylabel="Decibelios")
```



Si calculamos el espectro de la señal si la enventanamos rectangularmente.

[ ]:
```
windowRect = rectWindow(confront,0.02,1,fs);
recSig,recfreqs = fftSignal(windowRect,fs)
plot(recfreqs, 20*log10.(abs.(recSig)), title = "Espectro Enventanada␣
↪Rectangular 20ms",xlim=(0,6000),xlabel="Frecuencias",ylabel="Decibelios")
```

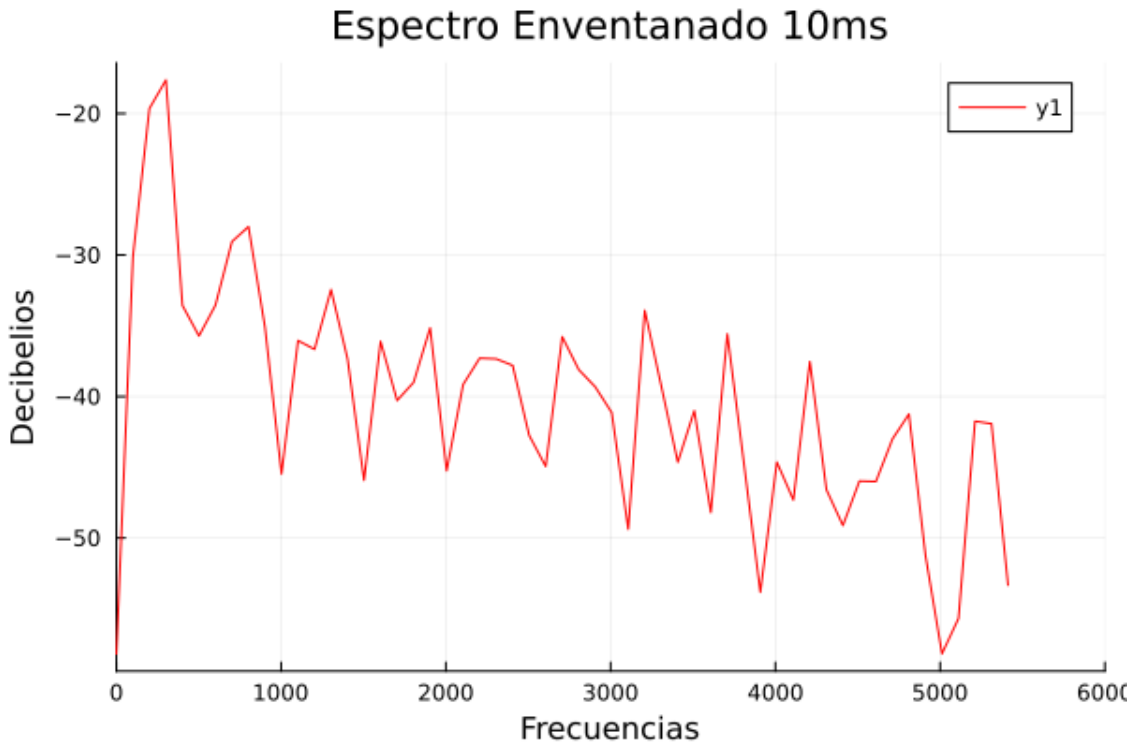


Probemos ahora con otras duraciones de ventanas y veamos:

Primero ventana de 0.01 segundos, 10ms.

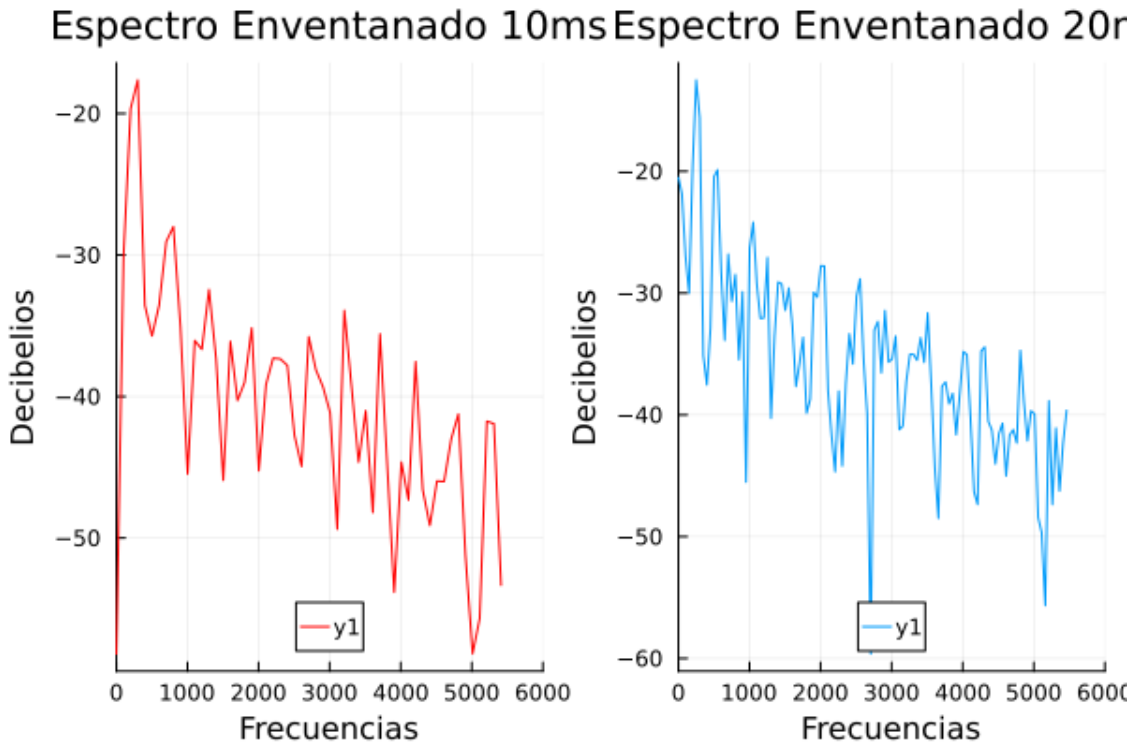
[ ]:
```
durationWindow = 0.01
numSamples = Int(round(fs * durationWindow));
hammWind = hamming(numSamples);
```

[ ]:
```
sigWindowed = confront[1:numSamples] .* hammWind
hammSig,hammfreqs = fftSignal(sigWindowed,fs)
plot10ms = plot(hammfreqs, 20*log10.(abs.(hammSig)), title = "Espectro␣
 ↪Enventanado␣
 ↪10ms",xlim=(0,6000),xlabel="Frecuencias",ylabel="Decibelios",color=:Red)
```



Si reducimos el tiempo de la ventana, vemos que los lóbulos, principales y secundarios, se ven ampliados. Esto ocurre porque al comprimir en tiempo, expandimos en el dominio de la frecuencia.

[ ]:
```
plot(plot10ms,plot20ms,legend=:bottom)
```



Energía localizada y tasa de cruces por cero

a. Seleccione el segmento de la señal entre las muestras 15500-19500. Obtenga la evolución de l

b. Observe el efecto del tamaño de la ventana. ¿Qué ocurre si el tamaño de la ventana es muy pe

c. La función zcr.m estima la tasa de cruces por cero. Tradúzcala a Julia como zcr(s,L) en dond

d. Use las funciones energía y tasa de cruces por cero para determinar qué partes de la señal s

Empecemos por el primer apartado. Seleccionemos las muestras deseadas de la señal de voz.

[ ]:
```
confrontSelect = confront[15501:19500];
```

Para calcular la energía en cada instante, elevaremos al cuadrado la señal, y la convolucionaremos con la función de la ventana de Hamming.

Como una ventana de 4000 muestras es demasiado grande, iremos iterando para calcular la energía de tramos de muestras más pequeños.

[ ]:
```
using DSP
```

[ ]:
```
function energia(s,h)
    """
```

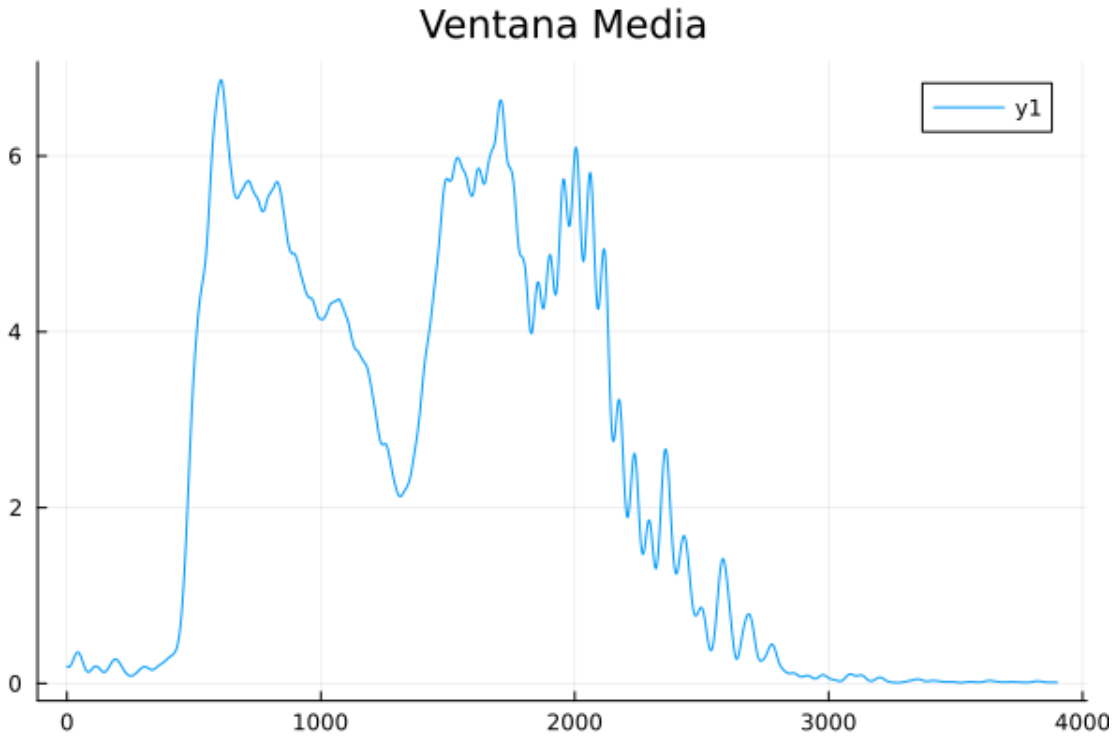
```
    Calcula la energía de la señal dada, s, convolucionando la señal al␣
 ↪cuadrado con la ventana.
    """
    energy = Vector{Any}(undef,0)
    Lt = length(s) - length(h)
    for n in 1:Lt
        j = Int(n)
        push!(energy, sum((s[1+n:length(h)+n].*h).^2))
    end
    return energy
end
```

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

Ahora tenemos la función que nos devuelve la energía, calculada como la convolución de la señal al cuadrado con la ventana de Hamming, como respuesta al impulso.

Iteremos sobre todo el trozo de señal, y representemos:

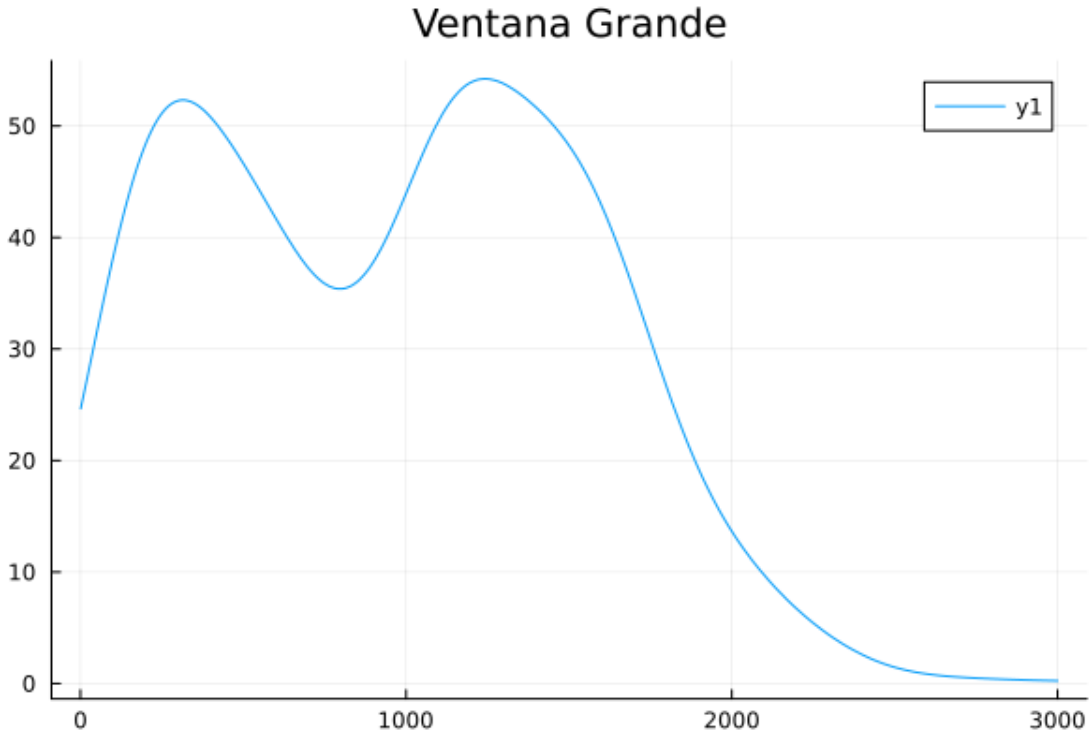
[ ]:
```
sizeWindow=100
hammWin = hamming(sizeWindow)
#Crearemos una ventana Hamming de tamaño 100 muestras.
energyTot = energia(confrontSelect, hammWin)
plotEnergyMediumWindow = plot(energyTot,title="Ventana Media")
```



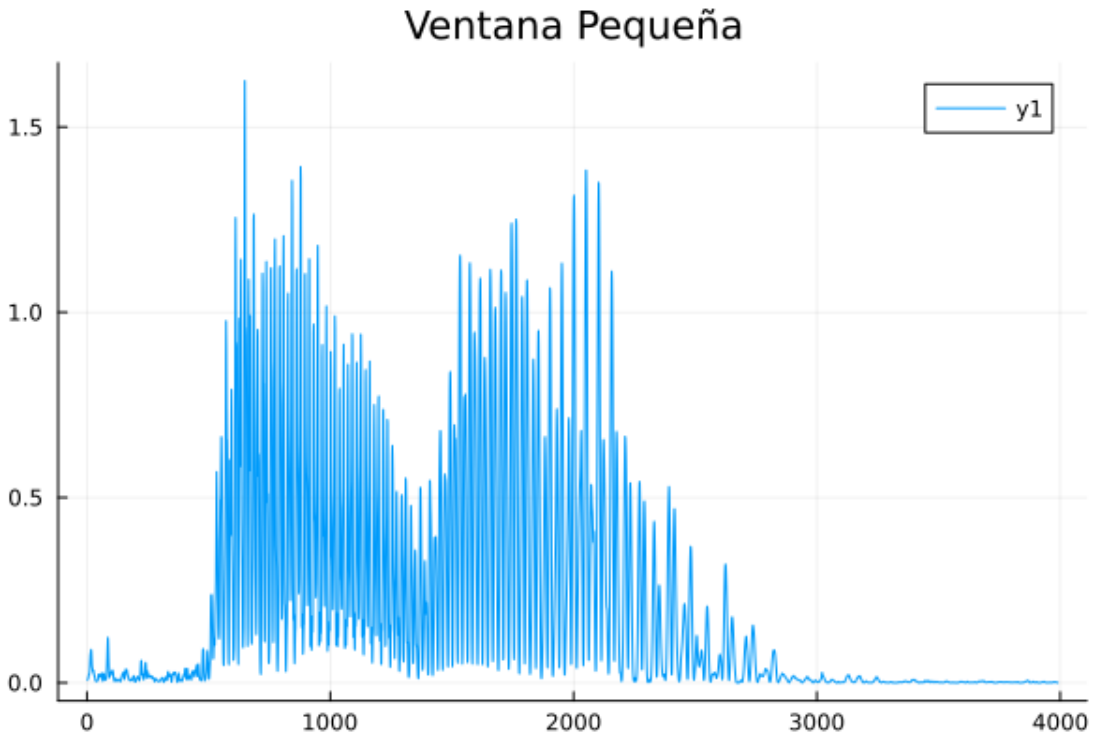
Vemos como la energía varía a lo largo del tiempo, siendo al principio de la señal cercana a 0, debido a que, al principio, la señal empieza con silencio, baja energía, y después se va modificando según avanza.

Probemos ahora modificando el valor de la ventana. Probemos primero con un número mucho más grande que el anterior, y otro mucho más pequeño.

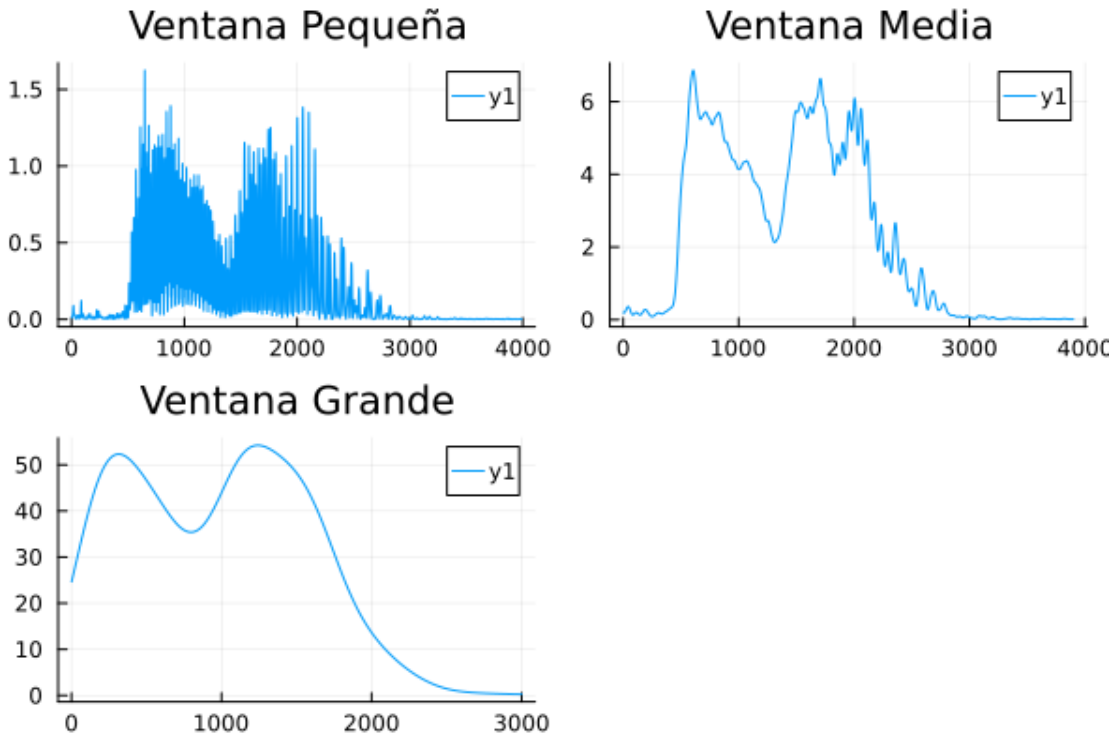
[ ]:
```
sizeWindow=1000
#Crearemos una ventana Hamming de tamaño 1000 muestras.
hammWin = hamming(sizeWindow)
energyTot = energia(confrontSelect, hammWin)
plotEnergyBigWindow = plot(energyTot,title="Ventana Grande")
```



[ ]:
```
sizeWindow=10
#Crearemos una ventana Hamming de tamaño 10 muestras.
hammWin = hamming(sizeWindow)
energyTot = energia(confrontSelect, hammWin)
plotEnergySmallWindow = plot(energyTot,title="Ventana Pequeña")
```



[ ]:
```
plot(plotEnergySmallWindow,plotEnergyMediumWindow,plotEnergyBigWindow)
```



Si vemos, si hacemos el valor de la ventana más grande, no tendremos tanta exactitud en la medida de la energía, será menos exacta que si la ventana es más pequeña, en la que obtendremos más detalle de la energía.

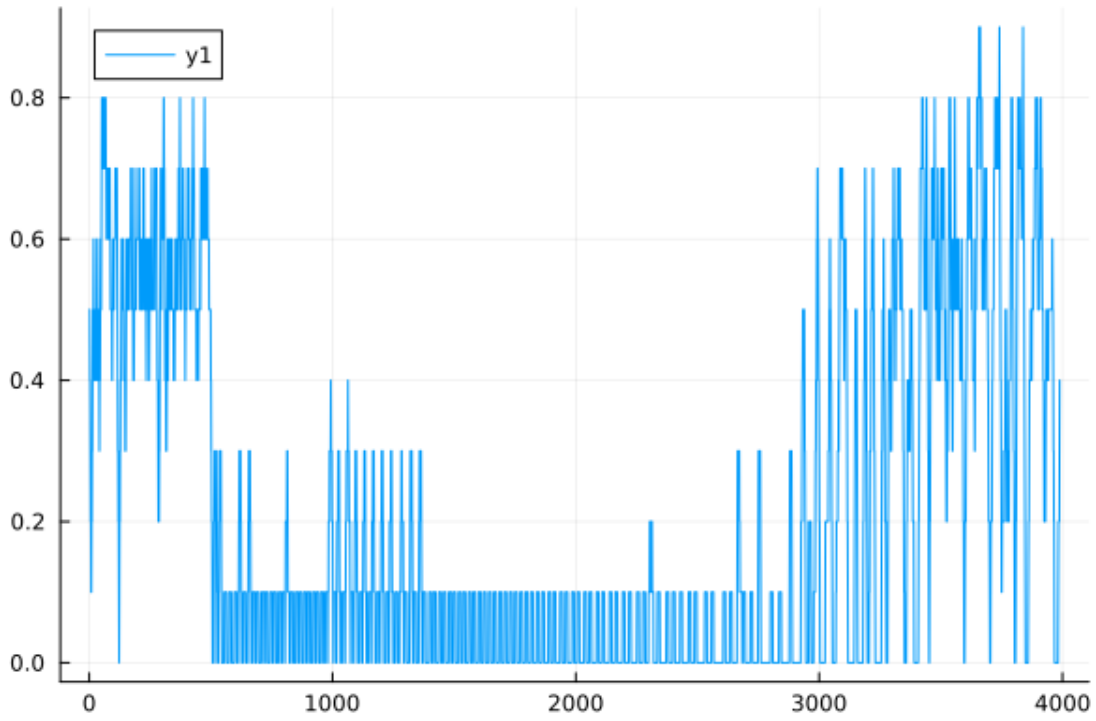
El archivo **zcr.m** no he sido capaz de encontrarlo, pero podemos programar una primitiva que nos devuelva la tasa de cruce por ceros.

Esta función nos devuelve la tasa que una gráfica cruza por cero, buscando que dos muestras consecutivas tienen diferente signos, dividido entre las muestras totales de esa gráfica.

[ ]:
```
function zcr(s,L)
    """
    Nos calcula la tasa de cruces por cero, de una señal , en la longitud de␣
 ↪ventana L. Utilizando la función para
    calcular la tasa.
    args:
    s: señal
    L: tamaño ventana
    returns:
    ZCR: vector con tasa de cruces por tramo
    """
    ZCR = Vector{Any}(undef,0)
    for n in 2:length(s) - L
        push!(ZCR,sum(0.5/L*abs.(sign.(s[1+n:L+n]) - sign.(s[n:L+n-1]))))
    end
    ZCR[1] = ZCR[2]
  return ZCR
end
```

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

[ ]:
```
Zcr = zcr(confrontSelect,length(hammWin));
plot(Zcr)
```



[ ]:
```
function vozSS(s, h)
    """
    Con la señal s, y la ventana h, calcula para cada ventana de la señal, su␣
 ↪energía y su Tasa de Cruces por Cero.
    Nos devuelve dos vectores, uno con el valor de su energía y su tasa de␣
 ↪cruces por cero.
    """
    E = energia(s,h);
    Zcr = zcr(s,length(h));
    return E, Zcr
end
```

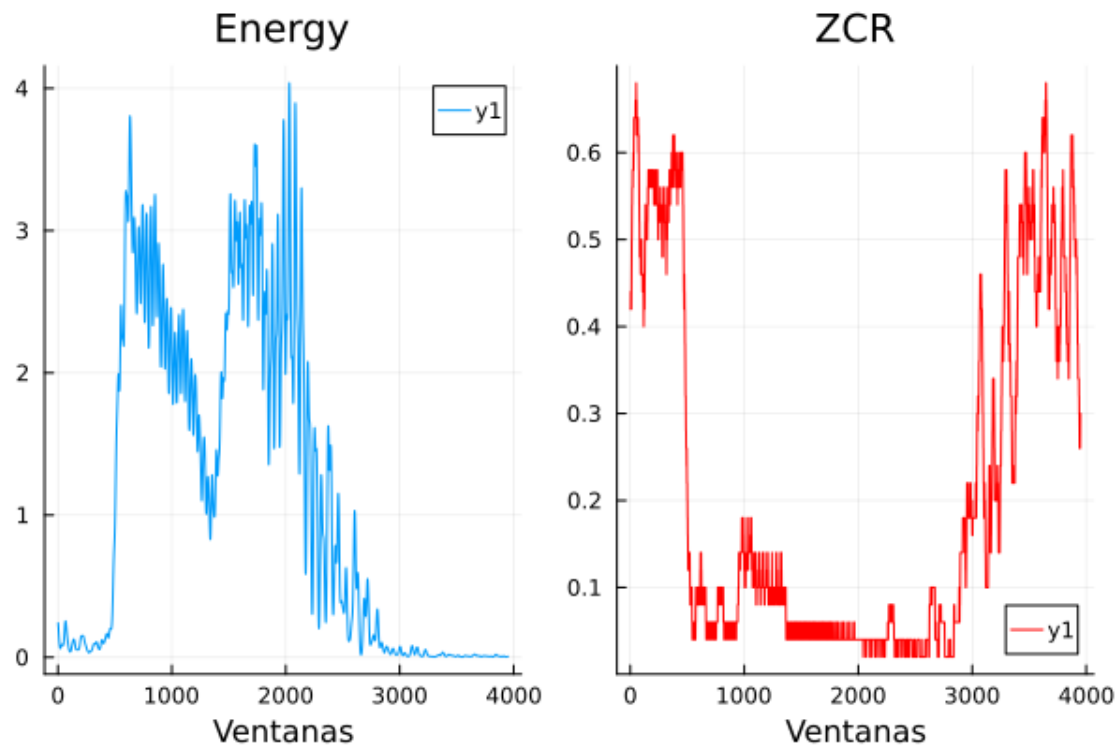
```
vozSS (generic function with 1 method)
```

Ahora dibujamos las gráficas de ambos resultados, como hicimos antes.

[ ]:
```
sizeWindow=50
#Crearemos una ventana Hamming de tamaño 100 muestras.
x = 1:1:sizeWindow;
hammWin = 0.54 .- (0.46 * cos.((2*pi*x) / (sizeWindow - 1)))
ener, zcrt = vozSS(confrontSelect,hammWin);
```

[ ]:
```
plotEnergy = plot(ener,title="Energy")
plotZCR = plot(zcrt,title="ZCR",color=:red)
```

```
plot(plotEnergy,plotZCR,xlabel="Ventanas")
```



El valor de la energía es el mismo que hemos obtenido antes, pero para el valor de *Zero Crossing Rate*, zcr, obtenemos como para cada trozo de ventana, el valor cambia. Cuanto más grande es el valor de ZCR, significa que el sonido puede ser sordo, y en los que el valor es más pequeño, tenemos que esos tramos pueden ser sonoros.

4.Estimación del "pitch": autocorrelación y espectro

```
a. Obtenga dos tramas de la senál original (sig), trama 1: muestras 14200-14475 y trama 2:muest
b. Encuentre o cree una función para hallar la autocorrelación entre dos señales
[c,k] = xcorr(trama)
En c se almacenarán los valores de la autocorrelación y en k los valores de los
desplazamientos. Determine si se trata de tramas sonoras o sordas. En el caso de que
alguna de las tramas sea sonora,estime el pitch o frecuencia fundamental.
c. Obtenga el espectro de frecuencia de ambas tramas con una función spectrum(trama, fs) en dor
tramas sonoras o sordas. En el caso en que alguna de las tramas sea sonora, estime su
"pitch" o frecuencia fundamental.
d. Compare y comente las estimaciones obtenidas por ambos métodos.
```

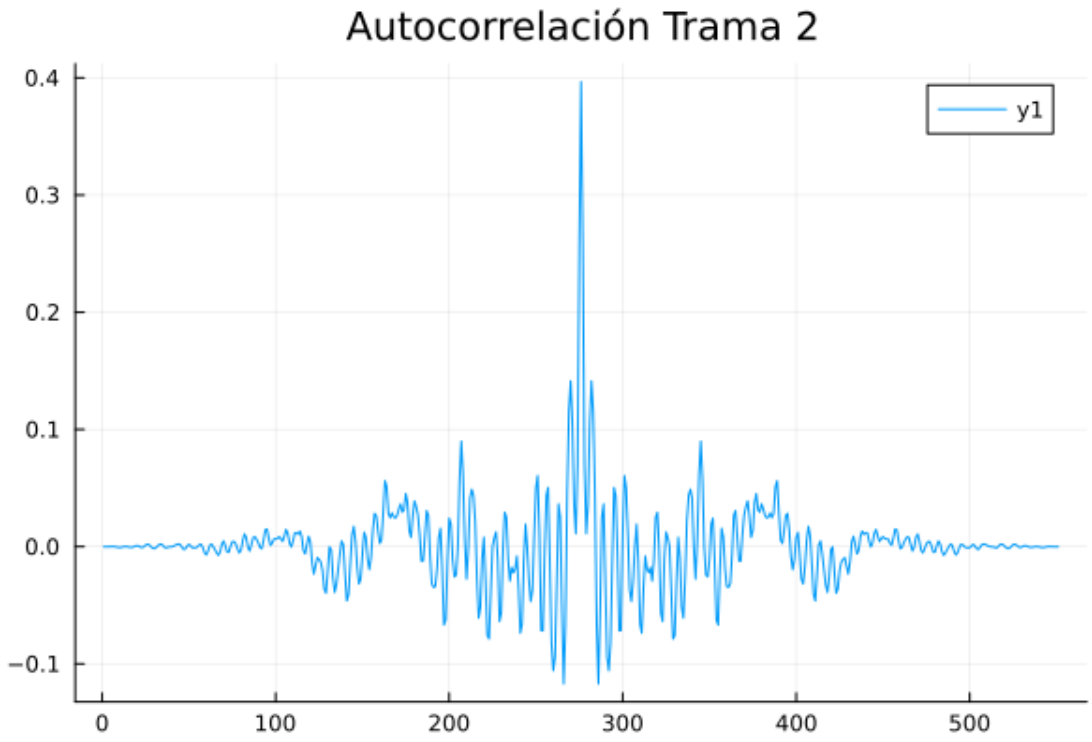
[ ]:
```
using StatsBase
```

[ ]:
```
trozo1 = confront[14200:14475];
trozo2 = confront[9200:9475];
```

[ ]:
```
trama1 = trozo1 .* hamming(276);
trama2 = trozo2 .* hamming(276);
```

Obtenidas los dos tramos de la señal, calculemos la autocorrelación de ambas señales.

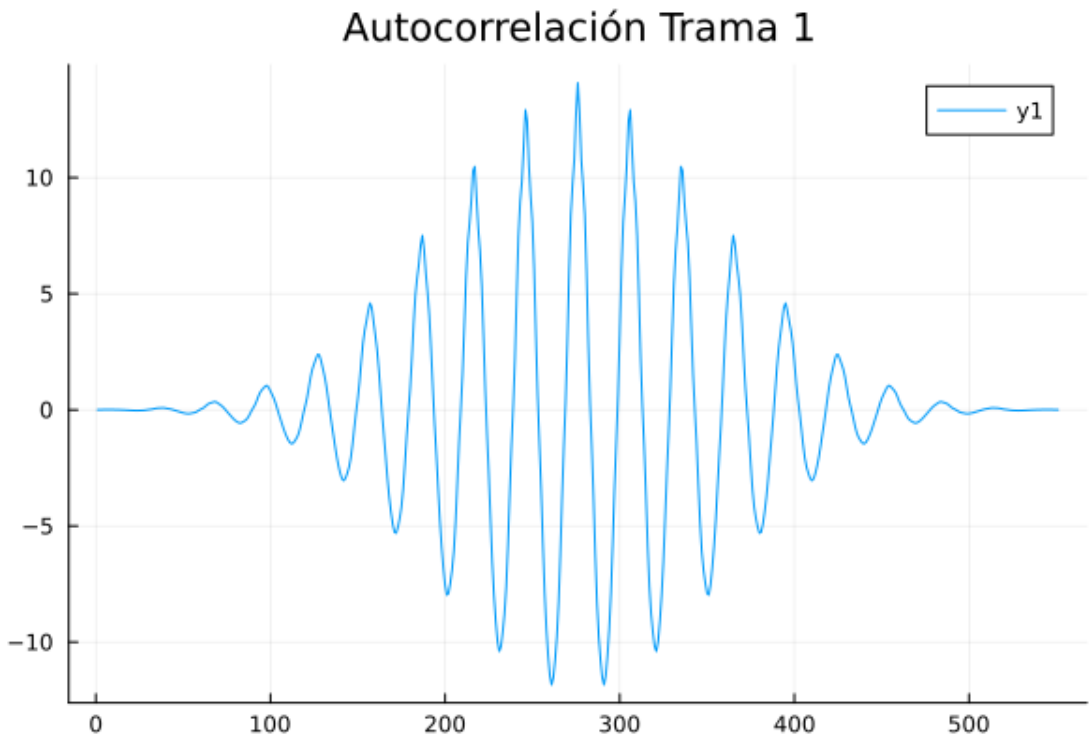
[ ]:
```
c = xcorr(trama2,trama2)
plot(c,title="Autocorrelación Trama 2")
```



Tenemos que esta función nos devuelve los valores de la correlación en un vector, y los valores del desplazamiento k es:

$$k = 2 * length(trama) - 1$$

[ ]:
```
c = xcorr(trama1,trama1)
plot(c, title="Autocorrelación Trama 1")
```



El primer sonido que tenemos, es de la trama 2, y vemos como su autocorrelación indicándonos que se trata de un sonido sordo, debido al ruido que tenemos durante toda la autocorrelación. Sin embargo, la trama 2, nos muestra que es una función que nos dice que se trata de un sonido sonoro, debido a su periodicidad.

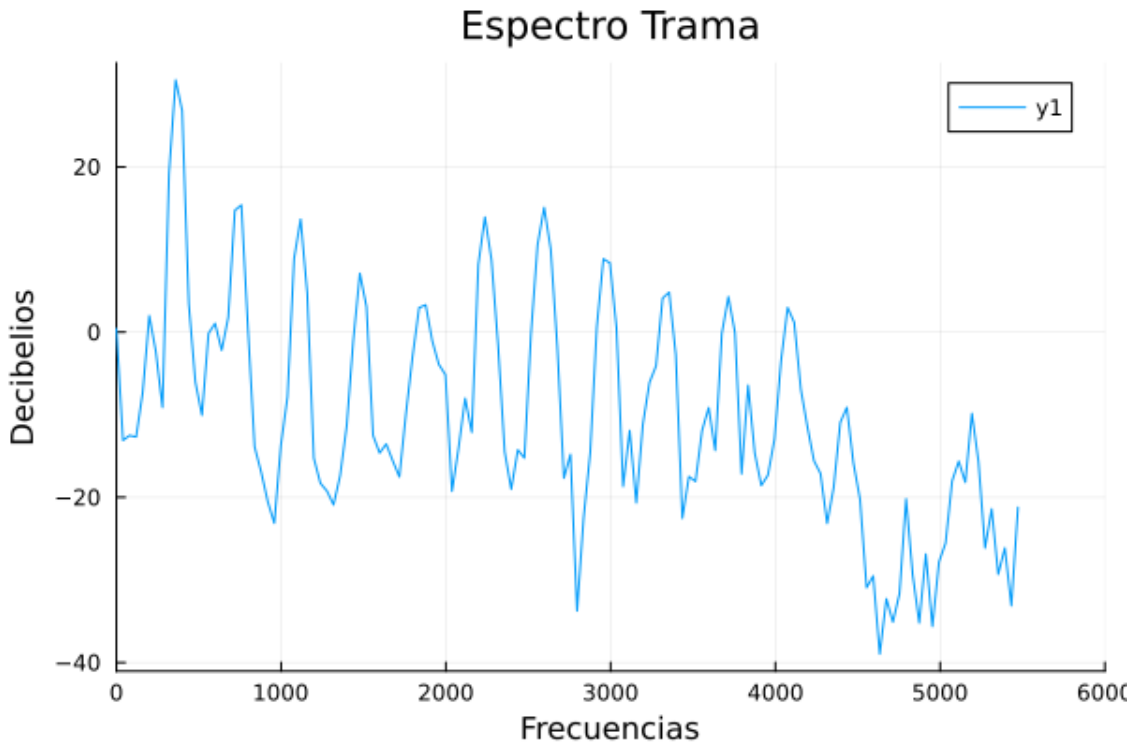
Para calcular los espectros, crearemos la función *spectrum*, la cual nos dibujará los espectros y podremos saber si son sonoros o sordos.

[ ]:
```
function spectrum(trama,fs)
    """
    A partir de una trama dada, y una frecuencia de muestreo, calcula su FFT y␣
 ↪dibuja su espectro.
    """
    sigFFT, freqsTrama = fftSignal(trama,fs) # Función que calcula las FFT.
    plot(freqsTrama, 20*log10.(abs.(sigFFT)), title = "Espectro␣
 ↪Trama",xlim=[0,6000],xlabel="Frecuencias",ylabel="Decibelios")
end
```

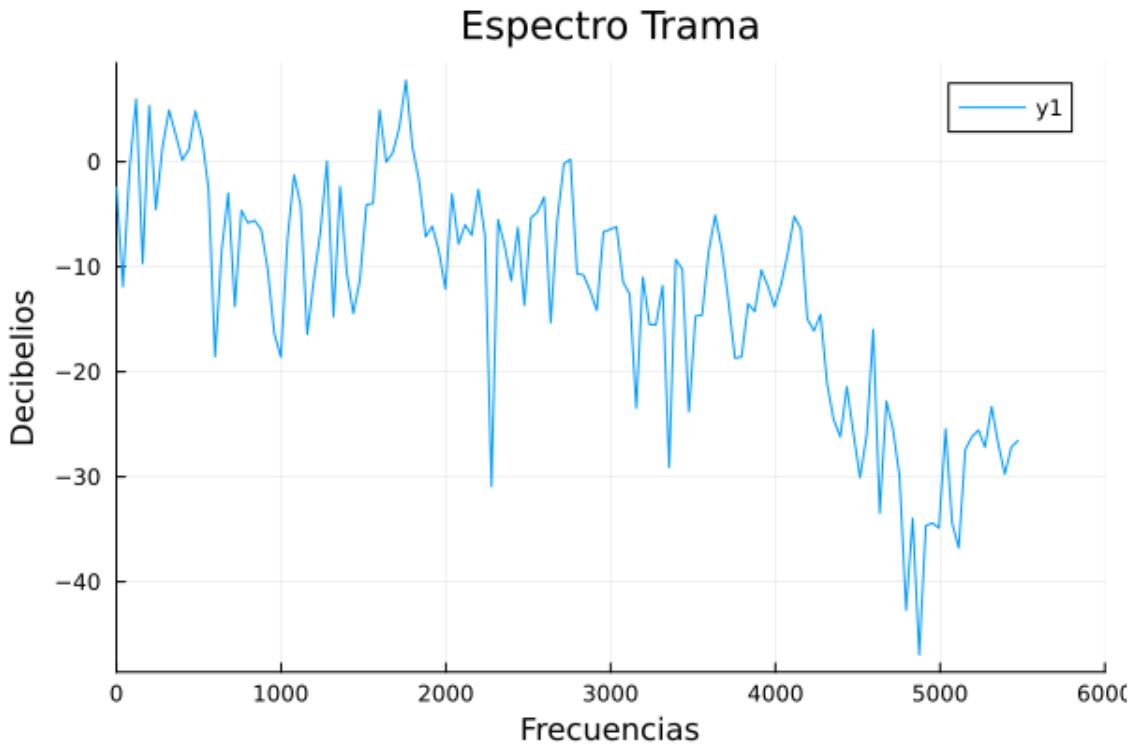
```
spectrum (generic function with 1 method)
```

Ploteamos los espectros de ambas tramas.

[ ]:
```
spectrum(trama1,fs)
```



[ ]:
```
spectrum(trama2,fs)
```



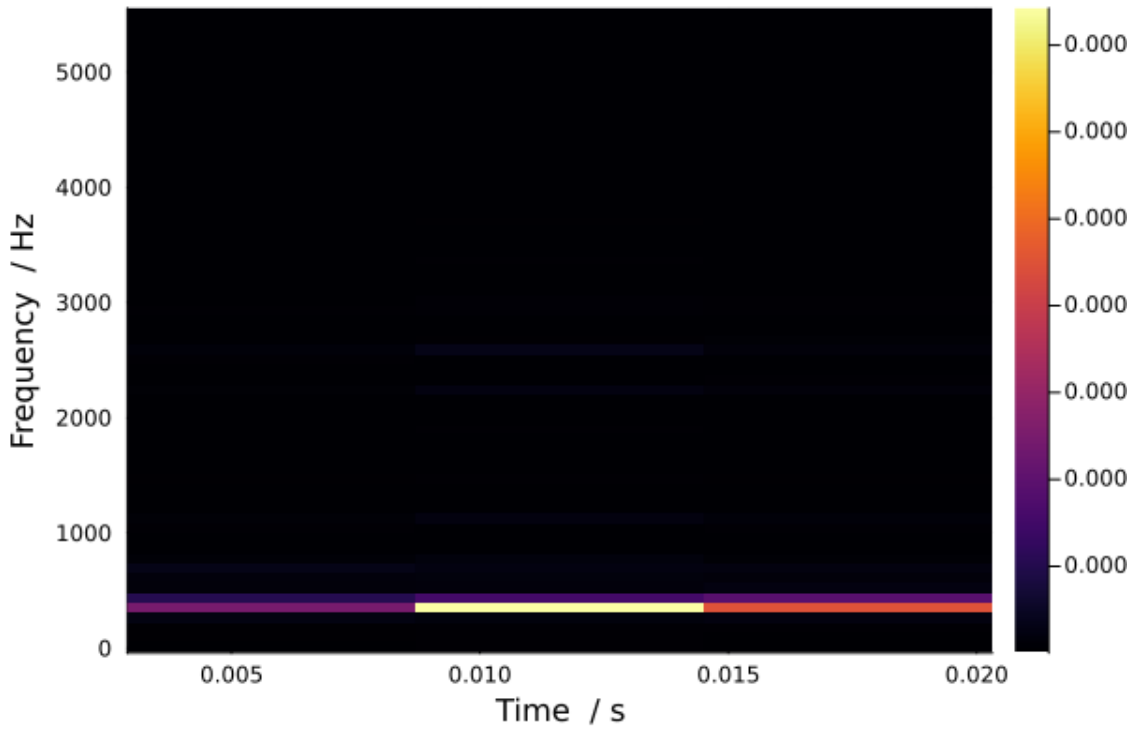
Estos espectros confirman nuestras sospechas, y la trama 1 es un sonido sonoro con una frecuencia fundamental en torno a los 360Hz.

Tenemos que las estimaciones que hemos hecho han sido acertadas, aunque en el dominio frecuencial, se hace de una manera más directa y visual.

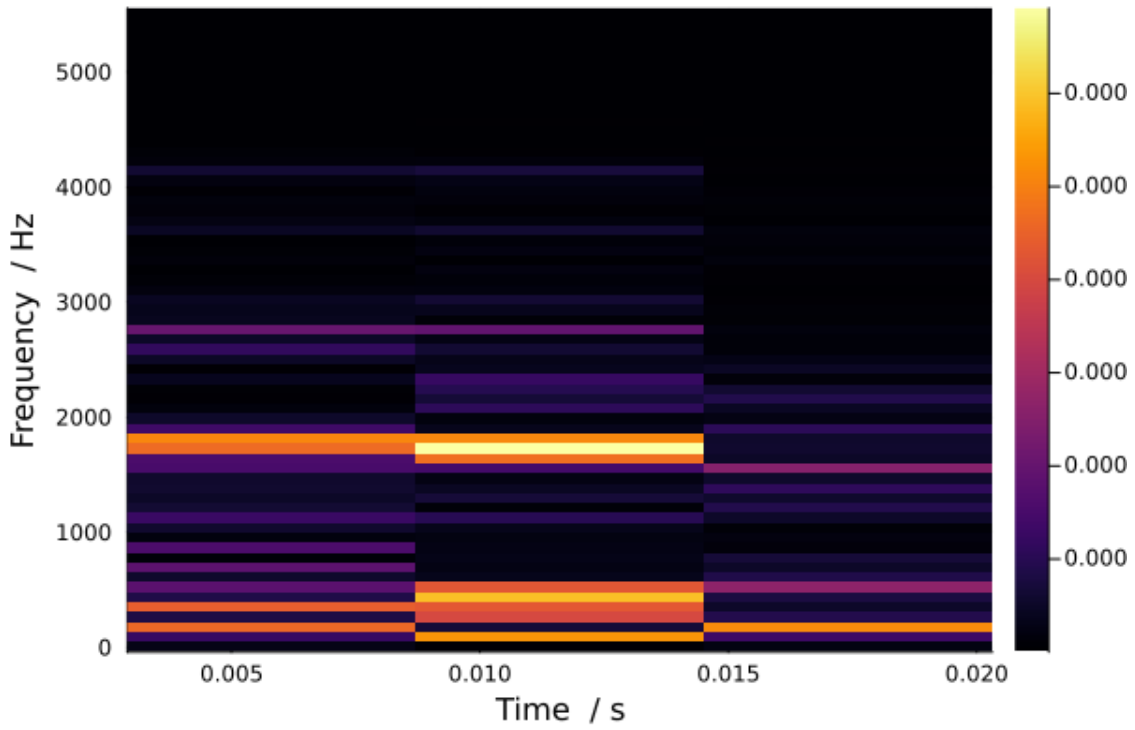
[ ]:
```
function spectrograms(s,N,fs)
    """
    Calcula el espectrograma de una señal y la dibuja.
    """
    spec = DSP.spectrogram(s, N;fs=fs)
    heatmap(spec.time, spec.freq, spec.power, xguide="Time  / s",␣
 ↪yguide="Frequency  / Hz")
end
```

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

[ ]:
```
banda_estrecha = 128
banda_ancha = 1024
spectrograms(trama1,banda_estrecha,fs)
```



[ ]:
```
spectrograms(trama2,banda_estrecha,fs)
```



Estos son los espectrogramas de banda estrecha, son los únicos que podemos dibujar. Si quisieramos dibujar los de banda ancha, deberíamos coger una señal con más muestras, para que pudiese ser efectivo el espectrograma de banda ancha. Igualmente, podemos saber cual de los dos tiene más resolución. El de banda estrecha tendrá mayor resolución frecuencial, cogerá la señal de trozos en trozos frecuenciales más pequeños, y sabremos como se comporta en diferentes frecuencias. Mientras tanto, al tener bandas frecuenciales más grandes, la resolución temporal será más clara y efectiva.

1.2.2 Ejercicio 3: Reflexión

Es deseable que cualquier actividad que curse conlleve un aprendizaje. Este aprendizaje será más profundo y satisfactorio si usted es capaz de darse cuenta de él y enunciarlo.

Objetivo

Aprender a descubrir y ser consciente de los resultados de aprendizaje propios.

Actividad

Escriba un párrafo listando qué conocimientos y destrezas ha adquirido en esta práctica.

Durante el desarrollo de esta práctica he ido analizando y dándome cuenta de como funciona el procesado digital del sonido. Al trabajar desde sonidos con tonos puros, como son las funciones sinusoidales, hasta con señales de voz reales, te das cuenta de lo diferente que son, y lo diferente que se convierte el trabajar con ellas.

Para algunas señales, necesitarás unos parámetros, mientras que para otras señales, se buscarán

otras, para que el trabajo con ellas sean lo más completo posible, siguiendo siempre el objetivo que se tiene.

Además, se han trabajado los temas de las ventanas, tema desconocido anteriormente para mí, pero que convierte el cálculo en algo más preciso en cada instante de tiempo. Pudiendo ser más precisos en el cálculo de la energía, tasa de cruces por cero o autocorrelación, ayudándonos en la detección de sonidos, y en el tratamiento de ellos.

Ahora somos capaces de calcular espectrogramas, que es una herramienta que nos ayuda a ver la evolución de la frecuencia en el tiempo. Esta herramienta nos proporciona conocimiento para el tratamiento de la señal, y que, podamos ser más precisos.

Con todo esto, y con el conocimiento añadido de analizar espectros, somos más capaces de trabajar con señales de audio.