SolucionLab3VocoderLPC_BlazquezRamirez_BernabeuFernandez

December 14, 2023

1 Práctica 3: Construcción de un Vocoder

1.0.1 Blázquez Ramírez, Javier y Bernabéu Fernández, Alejandro

1.0.2 Ejercicio 1: Análisis Exploratorio de la Señal

Objetivo

Determinar si la señal proporcionada puede ser sujeta a codificación LPC efectiva.

Contenido

```
1.Lea y reproduzca la señal de voz suministrada con este enunciado.
2.Represente el sonido en el dominio del tiempo.
3.Represente la energía localizada y la tasa de cruces por cero como función del tiempo,
entramando la señal con diferentes tamaños de ventana y usando las primitivas desarrolladas en
la práctica 2.
```

Lo primero que debemos hacer, es ser capaces de leer la señal de voz, *confront.wav*. Para ello, usaremos el paquete *WAV*.

[ ]:
```
using WAV
```

[ ]:
```
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])])
```

Ahora tenemos que reproducir la señal, escuchemos, a la frecuencia de muestreo que fue grabada. Nos ayudaremos del paquete *Sound*.

[ ]:
```
using Sound
```

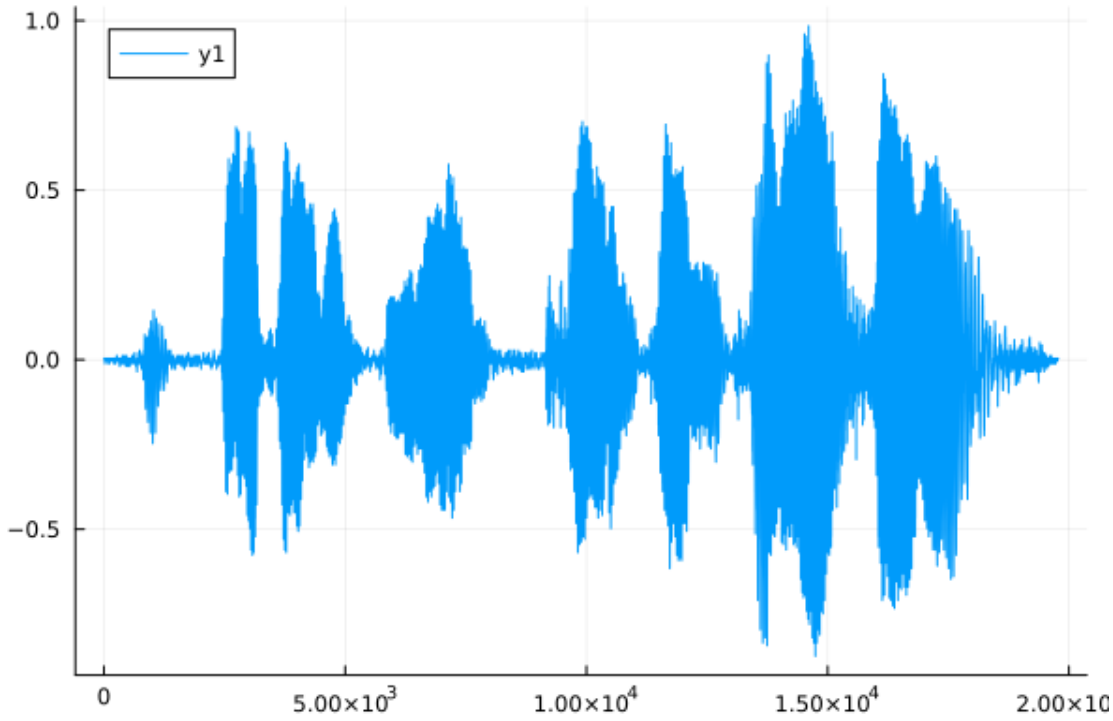
[ ]:
```
sound(confront,fs)
```

```
19778
```

Veamos la señal representada en el tiempo.

[ ]:
```
using Plots
```

[ ]:
```
plot(confront)
```



Ahora, haremos uso de las primitivas que creamos en la practica anterior, para representar la energía localizada y la tasa de cruces por cero

[ ]:
```
function hammingWindow(n)
    """
    Calcula la ventana de Hamming para un determinado número de muestras
    args:
    n: numero de muestras
    returns:
    hammWin: coeficientes de la ventana de Hamming
    """
    x = 1:1:n;
    hammWin = 0.54 .- (0.46 * cos.((2*pi*x) / (n - 1)))
    return hammWin
end
```

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

[ ]:
```
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)
```

[ ]:
```
function energia(s,h)
    """
    Calcula la energía de las tramas de la señal, que caben en la ventana␣
 ↪proporcionada. La ventana solo avanza
    una muestra por iteración.
    args:
    s: señal
    h: ventana de Hamming
    returns: valor de la energía para cada trama de la señal.
    """
    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)
```

Con esas funciones, tendríamos calculadas la tasa de cruces por cero, y la energía para una trama, pero queremos para toda las tramas de la señal, haremos uso de la siguiente función.

[ ]:
```
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.
    args:
```

```
    s: señal
    h: ventana de Hamming
    returns:
    E: energía de las tramas de la señal
    Zcr: Tasa de cruces por cero de las tramas señal
    """
    E = energia(s,h);
    Zcr = zcr(s,length(h));
    return E, Zcr
end
```

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

Le pasaremos a esa función, nuestra ventana Hamming y nuestra señal.

Querremos primeramente una ventana de 20ms, a una fs dada, eso corresponde a:

$$N = f_s * duration$$

Siendo medida frecuencia de muestreo en Hz, y la *duration* en segundos.

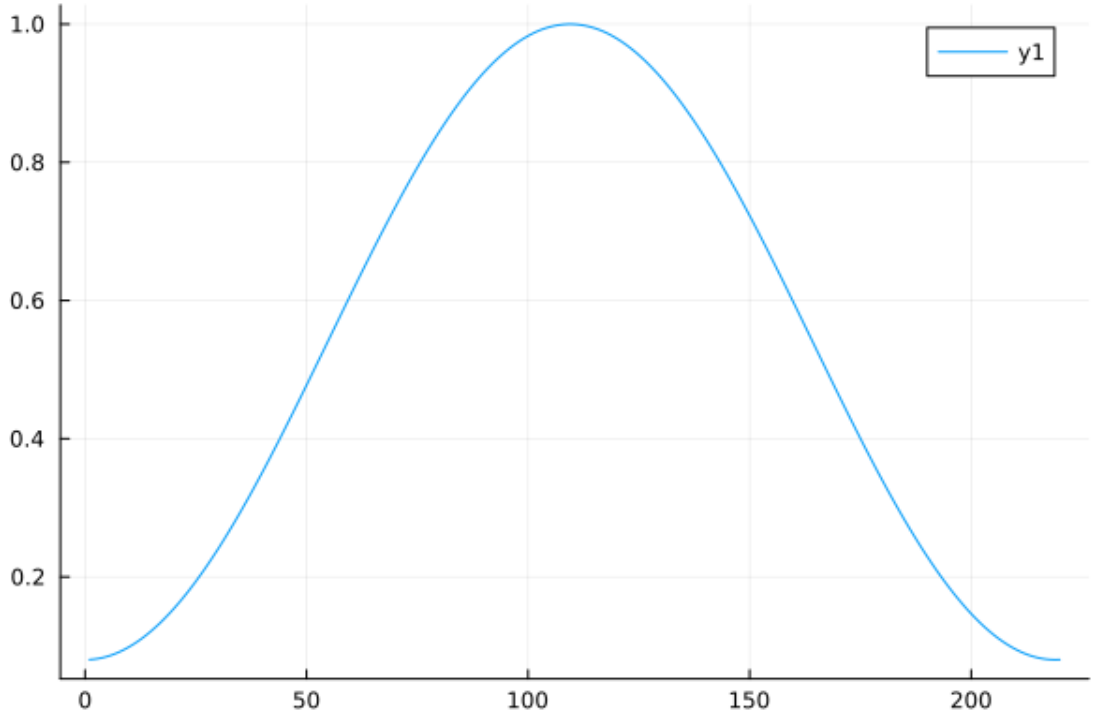
[ ]:
```
duration = 0.02
n = Int(round(fs * duration))
```

```
220
```

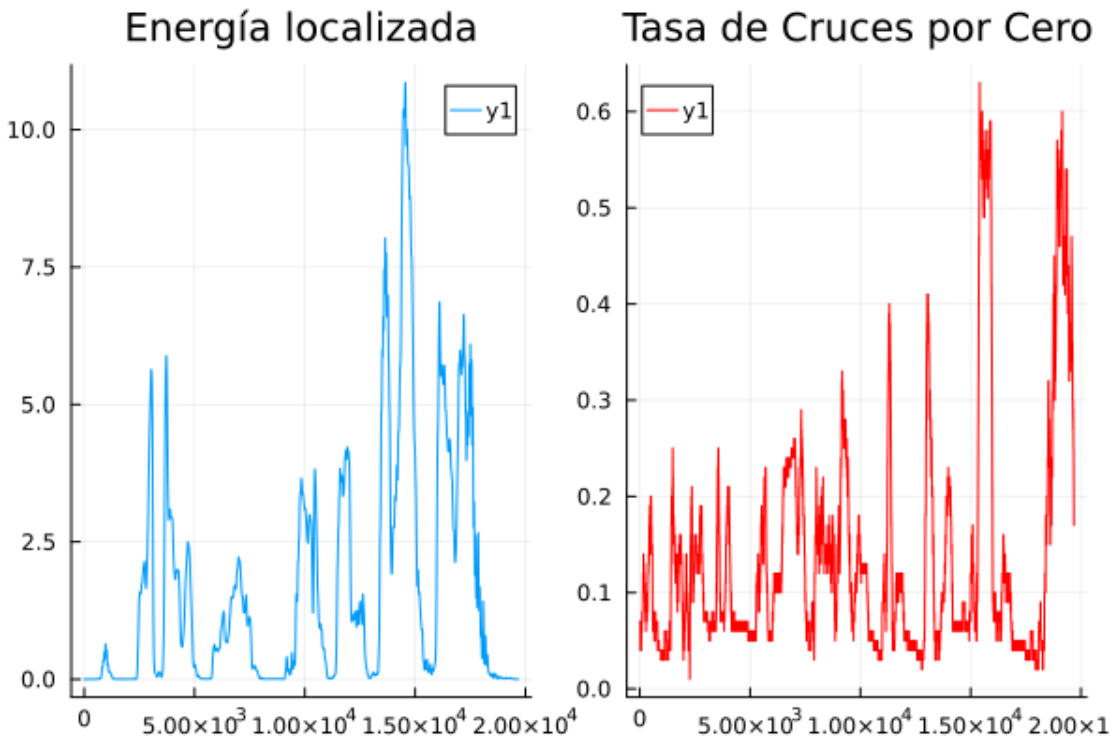
[ ]:
```
hamming = hammingWindow(n)
```

```
220-element Vector{Float64}:
 0.0801893082551276
 0.08075707720479086
 0.0817028395300804
 0.08302581679353727
 0.08472492007986876
 0.08679875089221212
 0.08924560230320866
 0.09206346035994023
 0.09525000574157239
 0.09880261566833948

 0.08924560230320866
 0.08679875089221212
 0.08472492007986876
 0.08302581679353727
 0.08170283953008045
 0.08075707720479086
 0.0801893082551276
 0.08000000000000002
 0.0801893082551276
```

[ ]:
```
plot(hamming)
```



[ ]:
```
hamming = hammingWindow(100)
energyTotal, zcrTotal = vozSS(confront,hamming)
plEnergy = plot(energyTotal, title="Energía localizada")
plZCR = plot(zcrTotal, title="Tasa de Cruces por Cero",color=:red)
plot(plEnergy, plZCR)
```



Tenemos ya la tasa de cruces por cero y la energía representadas, usando primitivas del anterior laboratorio, con un tamaño medio de ventana, como son 100 muestras.

1.0.3 Ejercicio 2: Análisis de una trama

Objetivo: Desarrollar el código de análisis-síntesis.

Contenido

`1.Seleccione y visualice en el tiempo y la frecuencia una trama sonora de la señal.`

`2.Obtenga la función de autocorrelación de la trama usando una función xcorr que reciba una trama y un orden y obtenga el vector de autocorrelación de ese orden.`

`3.Genere la matriz R usando la biblioteca ToeplitzMatrices.jl`

`4.Obtenga los coeficientes de predicción lineal usando el sistema anterior.`

Lo primero que haremos será seleccionar una trama sonora de la señal, pero como podemos saber si un tramo es sonoro o sordo, pues con las funciones graficadas anteriormente.

Si la tasa de cruces por cero, es un valor elevado, la señal cruza muchas veces por cero, por tanto, podemos deducir que ese tramo será un ruido sordo.

Sin embargo, la energía de una señal armónica es superior a la energía de una señal no armónica. Las señales sonoras tienen una forma armónica, debido a su producción, vibración de las cuerdas vocales.

Con esto, sabemos que buscamos una trama con alta energía y baja tasa de cruces por cero.

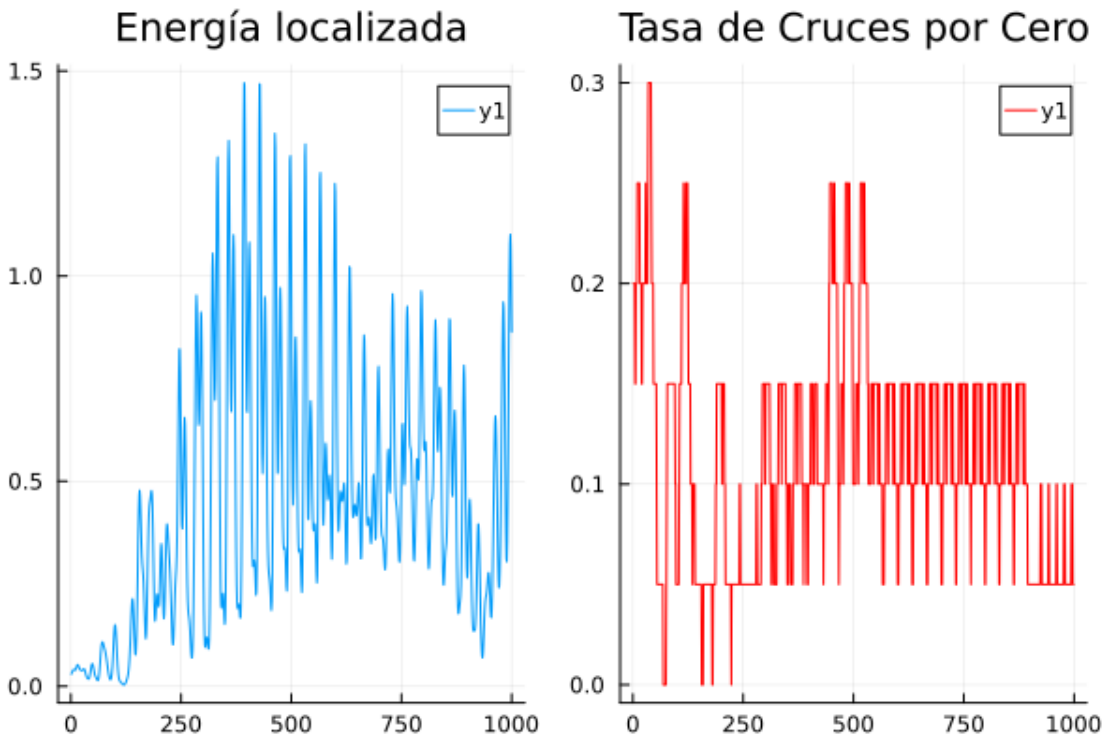
Mirando y observando las gráficas, vemos que una trama sonora de la señal puede ser la trama:

[ ]:
```
trozoSonora = confront[9481:10500]
```

```
1020-element Vector{Float64}:
  0.2313725490196079
  0.22352941176470598
  0.03529411764705892
 -0.05882352941176472
 -0.04313725490196074
  0.019607843137254832
  0.03529411764705892
  0.07450980392156858
  0.09019607843137245
  0.03529411764705892

  0.21568627450980382
  0.12941176470588234
  0.06666666666666665
 -0.019607843137254943
 -0.13725490196078427
 -0.22352941176470587
 -0.2705882352941177
 -0.2941176470588235
 -0.28627450980392155
```

Representamos la energía y la Tasa de Cruces por Cero de este trozo de señal, con una ventana de 20 muestras, algo más pequeña que la usada anteriormente. De esta manera, tendremos más detalle.

[ ]:
```
hamming = hammingWindow(20)
energyTotalS, zcrTotalS = vozSS(trozoSonora,hamming) # Ventana tamaño 100
plEnergySonora = plot(energyTotalS, title="Energía localizada")
plZCRSonora = plot(zcrTotalS, title="Tasa de Cruces por Cero",color=:red)
plot(plEnergySonora, plZCRSonora)
```



Nos quedaremos con este trozo de señal, que cumple con nuestros requisitos. Energía alta, y tasa de cruces por cero reducido.

[ ]:
```
tramaSonora = trozoSonora .* hammingWindow(1020)
```

```
1020-element Vector{Float64}:
  0.01851182717208519
  0.017890171529873537
  0.0028263070242730433
 -0.004714112133402848
 -0.003460410080041438
  0.0015747993952129388
  0.002838650052311951
  0.006002475527032859
  0.00727955679000673
  0.002854382875774458

  0.017347305875239645
  0.010393676008405432
  0.005347906487336772
 -0.0015713707111342855
 -0.010991193983284023
 -0.01789017152987353
 -0.02164942499786234
```

```
-0.023529411764705882
-0.022904464128173176
```

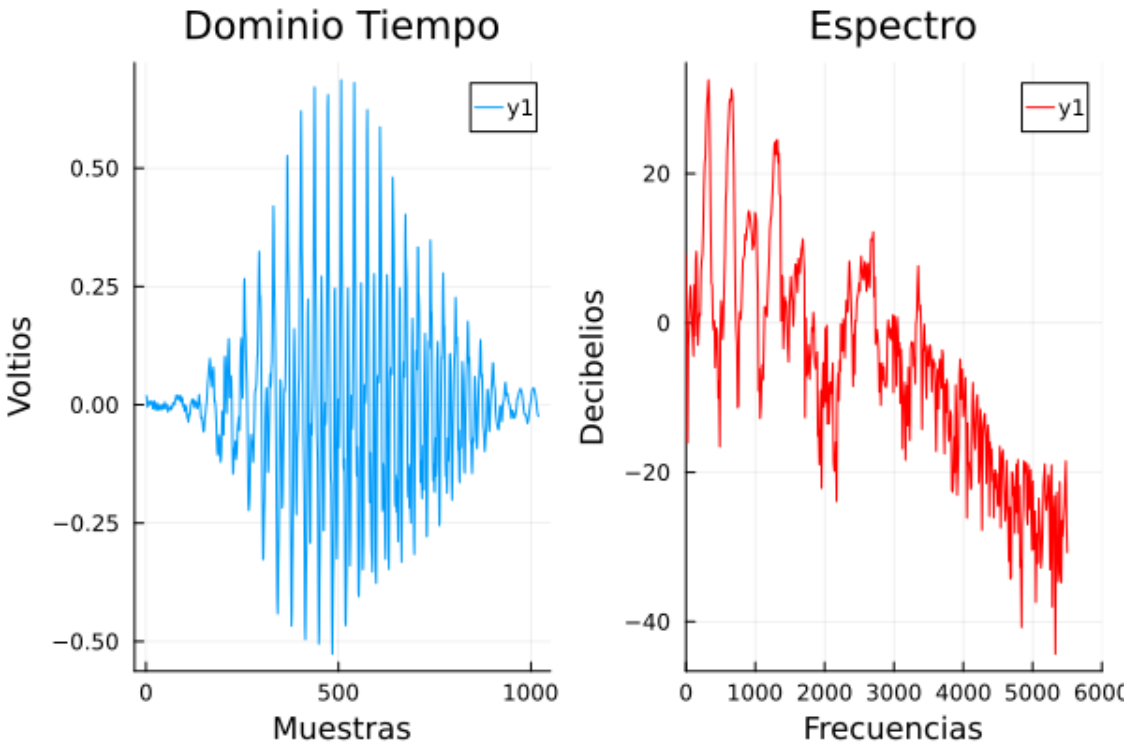
Tenemos una señal de 1680 muestras. Veamos su representación en el dominio del tiempo y en el dominio de la frecuencia:

[ ]:
```
using FFTW
```

[ ]:
```
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.
    args:
    y: señal
    fs: frecuencia de muestreo
    returns:
    F: FFT de la señal
    freqs: frecuencias de la FFT
    """
    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)
```

[ ]:
```
FFTconfront, freqsConfront = fftSignal(tramaSonora, fs)
timeDomain = plot(tramaSonora,title="Dominio Tiempo",xlabel="Muestras",ylabel="Voltios",xticks=(0:500:2000))
freqDomain = plot(freqsConfront, 20*log10.(abs.(FFTconfront)), title ="Espectro",xlabel="Frecuencias",ylabel="Decibelios",xticks=(0:1000:6000),xlim=(0,6000),color=:red)
plot(timeDomain, freqDomain)
```



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

1020

Ahora, con nuestra trama seleccionada, y sabiendo que efectivamente es sonora, podemos calcular y representar su autocorrelación de diferente orden.

Para ello, nos ayudaremos de la función *xcorr* del paquete *DSP*.

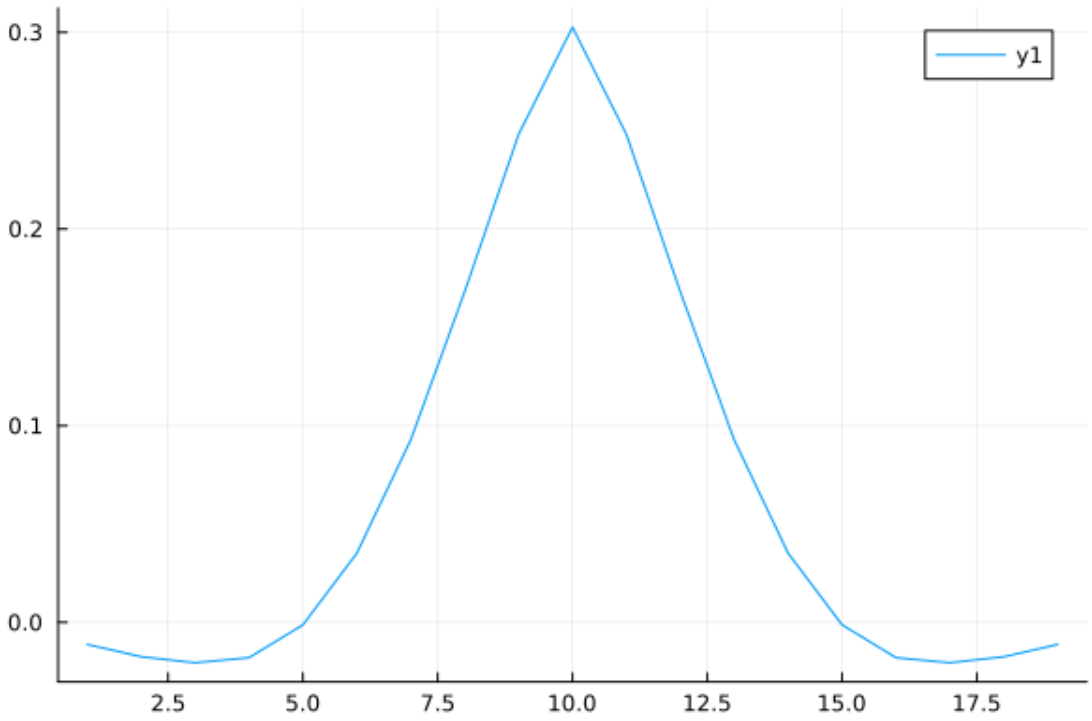
[ ]:
```
using DSP
using Statistics
```

[ ]:
```
function autocorrelation(signal, orden)
    """
    Calcula la autocorrelación de una señal con una orden dada. Para una trama␣
 ↪de tamaño del orden deseado,
    se calcula su autocorrelación. La autocorrelación de un orden, es cuanto se␣
 ↪parece la señal a si misma con
    el orden de muestras desplazadas.
    Esto nos devuelve un vector, que guardaremos para todas las sub-tramas␣
 ↪dentro
    de la trama total a calcular la autocorrelación. Como queremos estimar la␣
 ↪autocorrelación de toda la trama,
```

```
    hacemos una estimación de la autocorrelación, basándonos del estimador más básico, la media.

    Daremos por hecho que la trama ya viene enventanada.

    args:
    signal: señal
    orden: orden de la autocorrelación
    returns: media de la autocorrelación de las tramas
    """
    autocorr = Vector{Any}(undef,0)

    # Calcula la autocorrelación utilizando la función xcorr del paquete DSP
    for i in 1:(length(signal)/orden)
        j = Int(i)
        s = signal[(orden*(j-1))+1:orden*j]
        acf = xcorr(s,s)
        push!(autocorr, acf)
    end
    return mean(autocorr)
end
```

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

[ ]:
```
orden = 10
auto = autocorrelation(tramaSonora,orden)
plot(auto)
```



Calculando de esta manera la autocorrelación, obtenemos un vector de tamaño:

$$length = 2 * order - 1$$

La idea en la que se basa este cálculo de la autocorrelación, es en que en una trama de una señal de voz, muestras muy alejadas entre sí, no se parecen en absoluto. Sin embargo, las muestras cercanas, sí son parecidas. Esto es la autocorrelación, cuanto se parece a sí misma una señal desplazada en el tiempo.

Analizando la gráfica obtenida, vemos como en el centro de la gráfica, la autocorrelación alcanza su máximo, ya que estamos comparando exactamente las mismas gráficas, pero según nos alejamos de este centro, ya sea, hacia valores pasados o futuros, esta correlación disminuye drásticamente. Esto confirma la idea con la que estabamos trabajando de que señales cercanas se parecen, y las señales lejanas no se parecen.

Ahora, generaremos la matriz R, ayudándonos del paquete *ToeplitzMatrices.jl*.

[ ]:
```
using ToeplitzMatrices
```

[ ]:
```
R = Toeplitz(auto[1:orden],auto[1:orden])
```

```
10×10 Toeplitz{Float64, Vector{Float64}, Vector{Float64}}:
 -0.0112121   -0.017464    -0.0205308   …   0.24809      0.302627
 -0.017464    -0.0112121   -0.017464        0.168099     0.24809
 -0.0205308   -0.017464    -0.0112121       0.092766     0.168099
```

```
 -0.0179537   -0.0205308   -0.017464        0.0351501    0.092766
 -0.00126231  -0.0179537   -0.0205308      -0.00126231   0.0351501
  0.0351501   -0.00126231  -0.0179537   …  -0.0179537   -0.00126231
  0.092766     0.0351501   -0.00126231     -0.0205308   -0.0179537
  0.168099     0.092766     0.0351501      -0.017464    -0.0205308
  0.24809      0.168099     0.092766       -0.0112121   -0.017464
  0.302627     0.24809      0.168099       -0.017464    -0.0112121
```

Siguiendo la definición de la Matriz Toeplitz, tenemos la matriz R como nuestra matriz Toeplitz.

Con esta matriz, y con los coeficientes obtenidos de la autocorrelación, podemos obtener los coeficientes de predicción. Estos coeficientes son calculables, multiplicando esta Matriz de Toeplitz, R, con nuestros valores calculados antes de la autocorrelación.

[ ]:
```
a_k = R * auto[1:orden]
```

```
10-element Vector{Float64}:
  0.1924060965384098
  0.13673392386683447
  0.0801508397026748
  0.035336298205033465
  0.005864699570437913
 -0.00991433160026398
 -0.01639620995488186
 -0.018953431658296903
 -0.02178933410609981
 -0.02577358852208431
```

Tenemos ya los coeficientes calculados de nuestro predictor lineal, guardados en la variable a_k.

Dejaremos escrita una función, la cual nos devuelva los coeficientes LPC ya calculados.

[ ]:
```
function get_coeff(trama, orden)
    """
    Para una trama dada, se supone enventanada, calcula los coeficientes LPC de␣
 ↪esa trama.
    args:
    trama: trama a calcular
    orden: orden de la autocorrelación, serán el número de coeficientes.
    returns:
    a_k: coeficientes LPC
    """
    auto = autocorrelation(trama,orden)
    R = Toeplitz(auto[1:orden],auto[1:orden])
    a_k = R * auto[1:orden]
    return a_k
end
```

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

1.0.4 Ejercicio 3: Síntesis de una trama

Objetivo

Depurar el código de análisis-síntesis.

Contenido

1.Obtenga, para la trama dada, el error de predicción como una señal, filtrando la trama a trav
del filtro FIR inverso.

2.Sintetice la trama de voz analizada mediante el filtro de síntesis calculado en el apartado

3.Inspeccione, auditivamente, en el tiempo y en la frecuencia las tramas de voz original y sint
¿Son correctos los coeficientes de análisis-síntesis

Un filtro FIR inverso, es un filtro FIR en el que, el numerador y el denominador están invertidos Para obtener el error de predicción, tenemos primero que obtener la predicción, y después comparar esa señal con la señal original, así seremos capaces de obtener el error.

Para obtener una prediccion de la señal, la obtendremos de la siguiente manera:

$$[n] = \sum_{k=1}^{p} a_k * s[n-k]$$

Esta es la manera teórica para predecir la señal. Pero lo podemos hacer de otra manera, ayudándonos de primitivas ya existentes en los paquetes que estamos utilizando.

Como hemos dicho, el filtro FIR inverso, es la inversa la división de coeficientes del filtro. Tenemos que los coeficientes son a_k, calculados anteriormente, y b_k, que en este caso son iguales a 1. Por tanto, si el filtro FIR es calculado con los coeficientes b_k y a_k.

[ ]:
```
function obtener_error_prediccion(trama, coeficientes)
    """
    Obtiene el error de predicción de una trama, calculando primero la trama␣
 ↪que hubiera sido predicha.
    Se ayuda en las transformadas Z.
    args:
    trama: trama a calcular su error de predicción.
    coeficientes: coeficientes LPC.
    returns:
    error_prediccion: error de la predicción de la trama
    """
    trama_predicha = filt(coeficientes, 1, trama)
    error_prediccion = trama - trama_predicha
    return error_prediccion
end
```

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

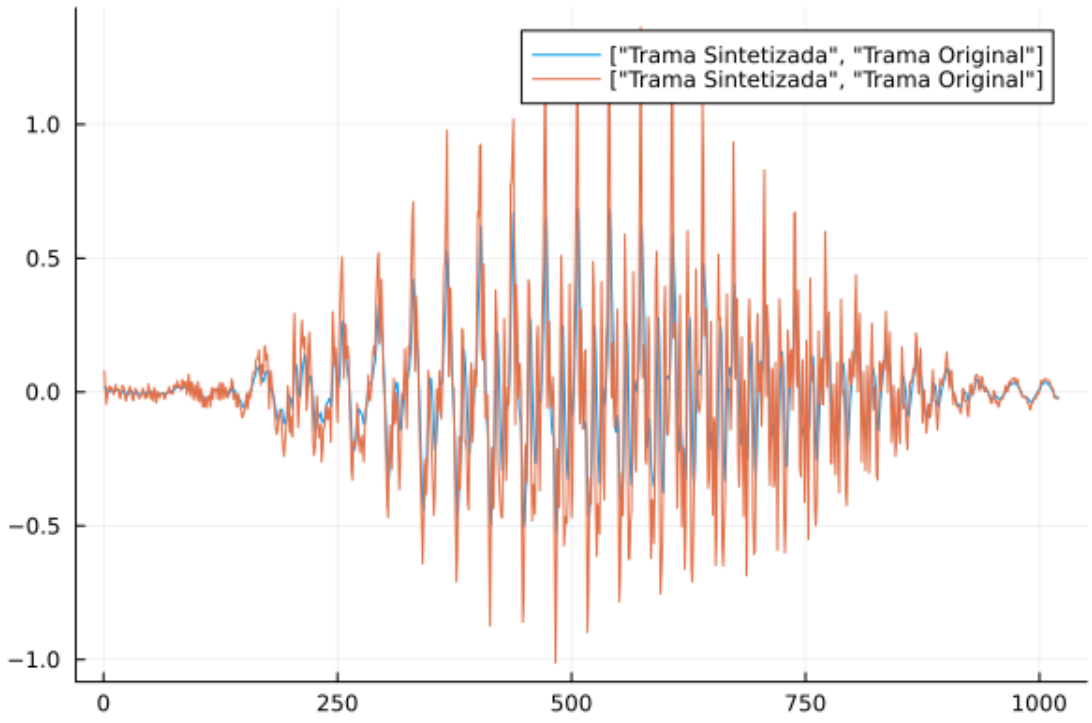
[ ]:
```
function sintetizar_trama(coeficientes, error_prediccion)
    """
    Sintetiza una trama a partir del error de predicción de una trama respecto␣
 ↪a unos coeficientes LPC.
    args:
    coeficientes: coeficientes LPC.
    error_prediccion: error de la predicción de la trama
    returns:
    trama_sintetizada: la síntesis del error de predicción
    """
    trama_sintetizada = filt(1, coeficientes, error_prediccion)
    return trama_sintetizada
end
```

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

[ ]:
```
# Supongamos que tienes una trama y coeficientes_lpc y coeficientes_sintesis␣
 ↪del Ejercicio 2
trama_original = tramaSonora  # Tu trama original
error_prediccion = obtener_error_prediccion(trama_original, a_k)
trama_sintetizada = sintetizar_trama(a_k, error_prediccion);
```

Para calcular la trama sintetizada, nos ayudaremos del error de predicción. Veamos la trama original comparada con la trama sintetizada.

[ ]:
```
plot([tramaSonora,trama_sintetizada],label=["Trama Sintetizada", "Trama␣
 ↪Original"])
```



Podemos también escuchar estas tramas.

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

1020

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

1020

Si aumentamos el orden de la autocorrelación, la señal será más parecida a la señal original que teníamos en un principio.

1.0.5 Ejercicio 4: Síntesis LPC

Objetivo

Realizar el análisis-síntesis LPC de una elocución de voz.

Contenido

En este ejercicio vamos a obtener el resto de los parámetros necesarios para codificar, transmitir y reconstruir la señal de voz con efectividad.

`1. Identificar si cada trama de voz es sonora o sorda usando tanto la tasa de cruces por cero e energía localizada.`

2. Calcular la frecuencia fundamental de las tramas sonoras, si puede, usando la FFT.

3. De nuevo, calcule la frecuencia fundamental pero usando en este caso la autocorrelación. ¿En
propiedad fundamental de la teoría de señales se basa este método? Para responder, reflexione
acerca de lo que es la función de autocorrelación desarrollada más arriba.

4. Encuentre una trama sorda y una trama sonora de su señal de voz como en 1.3 y calcule y visu
sus autocorrelaciones como en 2.2. Para la trama sonora, estime el tono fundamental como en 4.

5. Halle los espectros de las tramas seleccionadas y represéntelos para explorarlos.

6. Construya una función que reciba una trama y devuelva todos los parámetros de análisis neces
para re-sintetizar la trama.

7. Construya una función que reciba todos los parámetros de análisis y reconstruya una trama de
en el tiempo.

8. Para toda la señal de voz s[n] diseñe e implemente un esquema de enventanado con ventanas de
longitud N que se solapen un 50% de la longitud. Ayúdese de alguna referencia técnica si no
entiende qué tiene que hacer. También, decida qué orden de filtro LPC va a usar…

9. Sobre el diseño de enventanado anterior, cuando pase por una trama, analícela y sintetícela
construir una señal aproximada sZ[n].

10. Compare s[n] y su aproximación sZ[n] visual y auditivamente. Investigue diferentes valores
orden del filtro. ¿Puede observar algún efecto en la señal reconstruida?

Hay varias maneras posibles para decidir si una trama específica es sonora o sorda. La que usaremos será la decisión por el error cuadrático medio.

Ahora, haremos un bucle por toda la señal, para decidir que sonidos son sordos y cuales sonoros. Este bucle debe coger trozos de la señal, en la que la señal se pueda considerar constante, por ser las señales de voz *cuasiestacionarias*. Cogeremos tramos de 20ms.

[ ]:
```
function trama_to_tramas(s, r)
    """
    Subdivide una señal en tramas consecutivas y las guarda en un vector.
    args:
    s: signal
    r: tamaño de las sub-tramas
    returns:
    subvectores: vector de todas las tramas separadas.
    """
    size_sub = r
    size_s = length(s)
    subvectores = Vector{Vector{Float64}}()
    for inicio in 1:size_sub:size_s
        fin = min(inicio + size_sub - 1, size_s)
```

```
        subvector_actual = s[inicio:fin]
        subvector_actual .*=  hammingWindow(length(subvector_actual))
        push!(subvectores, subvector_actual)
    end
    return subvectores
end
```

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

[ ]:
```
quasi_time = 0.02
long_tramas = round(Int,fs * quasi_time)

tramas = trama_to_tramas(confront, long_tramas)
print(length(tramas[1]))
```

```
220
```

Podríamos utilizar el concepto de ruido de fondo, que se utiliza en la transcripción automática de voz a texto. Para decidir si una trama de voz es sonora o sorda, se puede calcular el valor cuadrático medio (RMS) de la señal de voz y compararlo con un umbral preestablecido. Si el RMS de la señal es mayor que el umbral, entonces la trama de voz es considerada sonora, de lo contrario, es considerada sorda.

[ ]:
```
function is_sonora(s, umbral_rms)
    """Decide si una trama es sonora o sorda mediante el error cuadrático medio.
 ↪ Si ese error está por encima de
    un umbral, decide sonido sordo
    args:
    s: trama a decidir
    umbral_rms: umbral del decisor
    returns: bool:
    true: sonido sonoro
    false: sonido sordo
    """
    rms = sqrt(mean(abs2, s))
    return rms > umbral_rms
end
```

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

[ ]:
```
umbral_rms = 0.07
voiced = []
unvoiced = []
for trama in tramas
    if is_sonora(trama, umbral_rms)
        push!(voiced,1)
    else
        push!(unvoiced,1)
```

```
    end
end
t = length(voiced)
l = length(unvoiced)
println("Hay ", t, " tramas sonoras en la señal original")
println("Hay ", l, " tramas sordas en la señal original")
```

```
Hay 50 tramas sonoras en la señal original
Hay 40 tramas sordas en la señal original
```

Ahora intentaremos calcular la frecuencia fundamental de las tramas sonoras, que son las que tenemos que analizar y sintetizar.

Podemos calcular la FFT de las tramas, y cogiendo su argumento máximo de su envolvente, calcular su frecuencia fundamental.

Calculemos la frecuencia fundamental de la primera trama, y dejemos creada una función que nos diga la frecuencia fundamental de la trama sonora.

[ ]:
```
for trama in tramas
    if is_sonora(trama, umbral_rms)
        F_trama,freqs_trama = fftSignal(trama, fs)
        magnitude_values = abs.(F_trama)
        f0 = freqs_trama[argmax(magnitude_values)]
        println("La frecuencia fundamental de la primera trama sonora de la␣
 ↪señal es: ", abs(f0), " Hz")
        break
    end
end
```

```
La frecuencia fundamental de la primera trama sonora de la señal es: 701.5909 Hz
```

Dejemos escrita la función que nos devuelva la f0, pasandole una trama, que ya sabríamos que sería sonora.

[ ]:
```
function calculate_f0(s,fs)
    """
    Calcula la frecuencia fundamental de una trama mediante el método de la FFT.
 ↪ La frecuencia fundamental será la
    frecuencia donde se alcanza el máximo de la envolvente espectral
    args:
    s: trama a calcular
    fs: frecuencia de muestreo
    returns:
    f0: frecuencia fundamental
    """
    F_trama,freqs_trama = fftSignal(s, fs)
    magnitude_values = abs.(F_trama)
    f0 = freqs_trama[argmax(magnitude_values)]
    return f0
```

```
end
```

```
calculate_f0 (generic function with 1 method)
```

Para calcular ahora la frecuencia fundamental basándonos en la autocorrelación, debemos saber el porque de este fundamento.

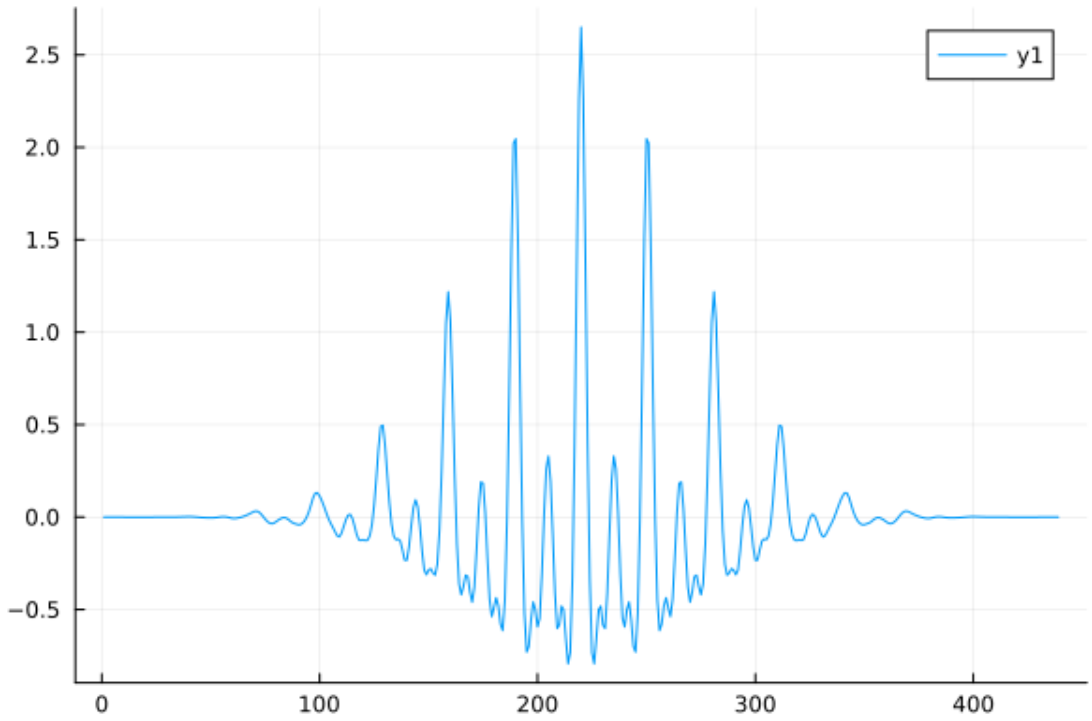
Si a una señal periódica, como lo son los sonidos sonoros, le calculamos su autocorrelación, obtendremos un pico, o un máximo, en el valor del período. Como lo que estamos buscando precisamente es la inversa del periodo, por la relación que una a la frecuencia con el periodo, podemos encontrar la f0, de la siguiente manera.

[ ]:
```
autocorr=0
for trama in tramas
    if is_sonora(trama, umbral_rms)
        autocorr = xcorr(trama,trama)
        f0 = (1:length(autocorr))[argmax(autocorr)]
        println("La frecuencia fundamental de la primera trama sonora de la␣
 ↪señal es: ", abs(f0), " Hz")
        break
    end
end
```

```
La frecuencia fundamental de la primera trama sonora de la señal es: 220 Hz
```

Podemos además dibujar la autocorrelación.

[ ]:
```
plot(autocorr)
```



Como vemos, tiene la forma de una señal sonora.

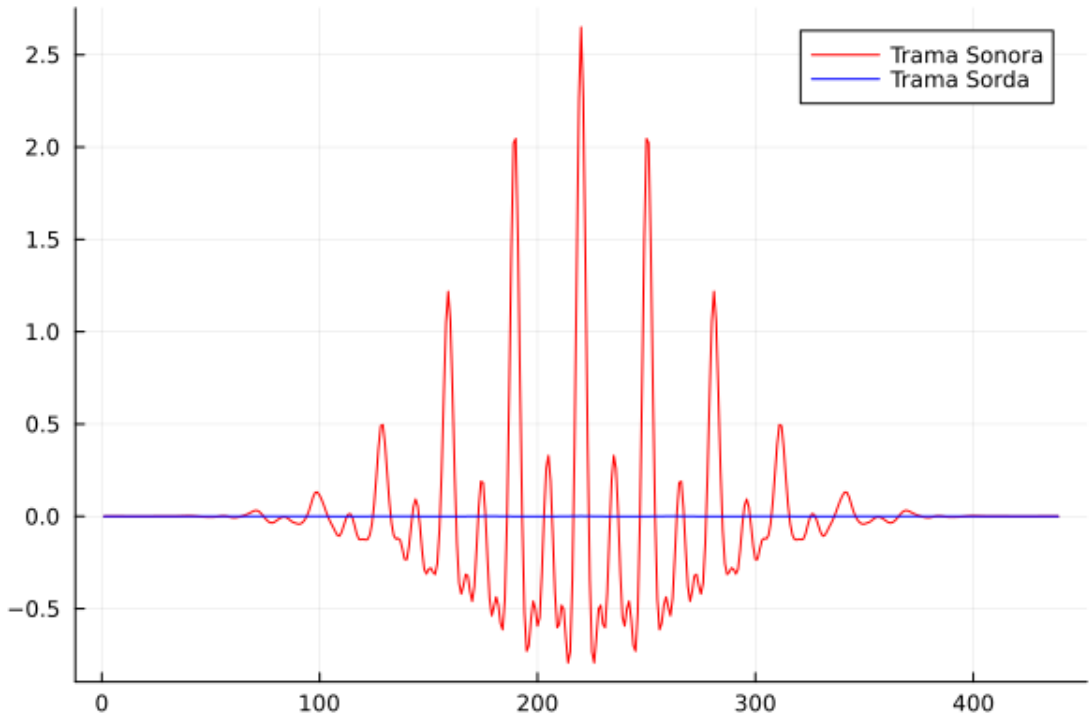
Las frecuencias entre ambos métodos no son iguales, pero esto tiene una explicación. Ambos métodos no son exactos al 100%, ambos cometen errores a la hora de calcularlo.

Elegiremos la primera manera de calcular la frecuencia fundamental

Encontraremos una trama sonora y una sorda, en caso de ser sonora, calcularemos su f0, y representaremos ambas autocorrelaciones.

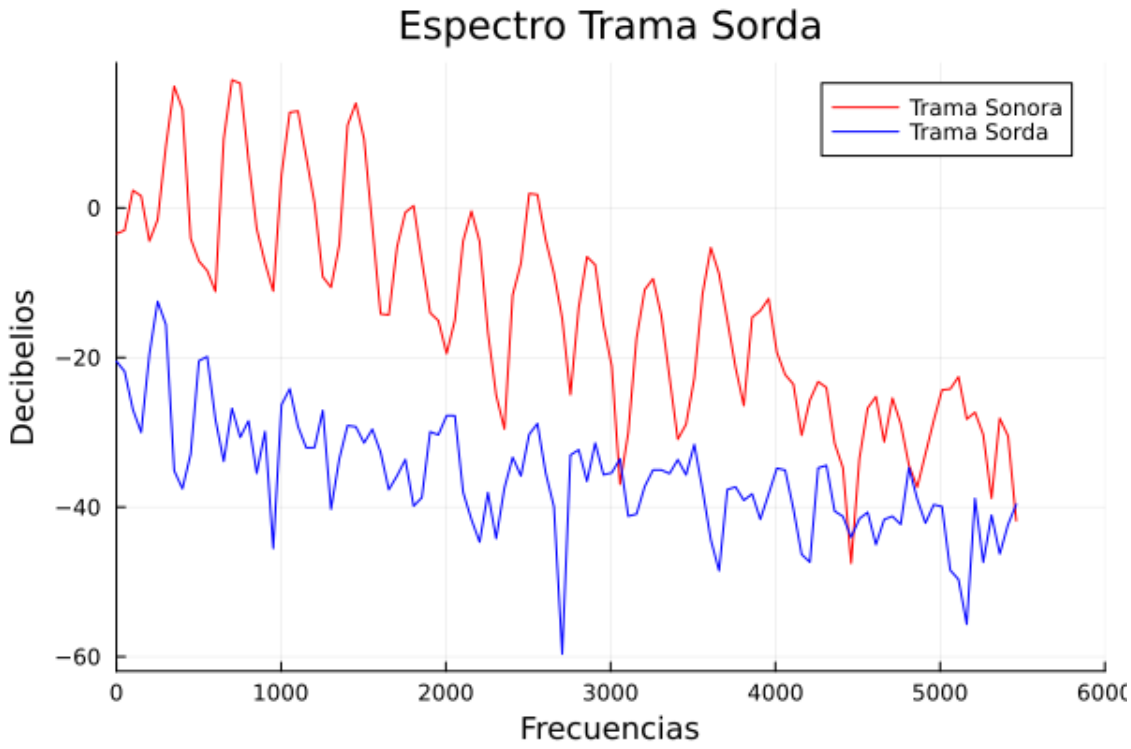
[ ]:
```
autocorrVoiced = 0
autocorrUnvoiced=0
F_trama_v = []
freq_v = []
F_trama_s = []
freq_s = []
for trama in tramas
    if is_sonora(trama, umbral_rms)
        autocorrVoiced = xcorr(trama,trama)
        f0 = calculate_f0(trama,fs)
        F_trama_v, freq_v = fftSignal(trama,fs)
        println("La frecuencia fundamental de la primera trama sonora de la␣
 ↪señal es: ", abs(f0), " Hz")
        break
    end

end
for trama in tramas

    if false == is_sonora(trama, umbral_rms)
        autocorrUnvoiced = xcorr(trama,trama)
        F_trama_s, freq_s = fftSignal(trama,fs)


        break
    end
end


plVoiced = plot(autocorrVoiced, label="Trama Sonora", color=:red)
plUnvoiced = plot!(autocorrUnvoiced, label="Trama Sorda",color=:blue)
```

```
La frecuencia fundamental de la primera trama sonora de la señal es: 701.5909 Hz
```



Los espectros los hemos hallado antes, pero los representamos ahora:

[ ]:
```
plot(freq_v, 20*log10.(abs.(F_trama_v)), title = "Espectro Trama␣
  ↪Sonora",xlabel="Frecuencias",ylabel="Decibelios",label="Trama␣
  ↪Sonora",xticks=(0:1000:6000),xlim=(0,6000),color=:red)
plot!(freq_s, 20*log10.(abs.(F_trama_s)), title = "Espectro Trama␣
  ↪Sorda",xlabel="Frecuencias",ylabel="Decibelios",label="Trama␣
  ↪Sorda",xticks=(0:1000:6000),xlim=(0,6000),color=:blue)
```



Vemos como la amplitud del espectro de la sonora es mayor en todo momento.

Tenemos que tener claro cuales son los parámetros que caracterizan una trama. Lo primero es saber si esa trama es sonora o sorda. Tras saber esto, las cosas varían ligeramente. En el caso de ser una trama sorda, no necesitaremos calcular ningún parámetro de síntesis. Se puede modelar como un ruido aditivo gaussiano blanco, por las propiedades de los sonidos sordos. Sin embargo, si es un sonido sonoro, deberemos calcular sus coeficientes LPC, calculado a partir de su autocorrelación y con la matriz Toeplitz, como hemos visto en ejercicios anteriores.

[ ]:
```
function get_params_4_lpc(trama, fs, orden=10)
    """
    Calcula los parámetros que deben ser transmitidos, en caso de que la trama␣
 ↪sea sonora, sino solo devuelve
    la decisión.
    params:
    trama: trama a calcular
    fs: frecuencia de muestreo
    orden: orden del filtro de análisis. Por defecto a 10, cuando empieza a␣
 ↪saturar el vocoder LPC.
    returns:
    [decision Sonora/Sorda, f0, coeficientes LPC]
    """
    voice_decision = is_sonora(trama, umbral_rms)
    if voice_decision
```

```
        f0 = abs(calculate_f0(trama,fs))
        coef_lpc = get_coeff(trama, orden)
        return [voice_decision,f0,coef_lpc]
    else
        return voice_decision
    end
end
```

```
get_params_4_lpc (generic function with 2 methods)
```

Ahora si le pasamos una trama sonora, nos devolverá los valores de análisis de la trama, con la que podremos trabajar.

Probaremos con diferentes tramas.

[ ]:
```
voiced_params = get_params_4_lpc(tramas[14],fs,10)
```

```
3-element Vector{Any}:
 true
  400.90912f0
     [0.3210046554958909, 0.2454468825245288, 0.17413923153829403, 0.
↪11452506642363505, 0.06250047598668608, 0.02078681681655124, -0.
↪005276024802932764, -0.015123368097560954, -0.012126345298006897, 0.
↪0020359898641452673]
```

[ ]:
```
unvoiced_params = get_params_4_lpc(tramas[1],fs,10)
```

```
false
```

Con esos coeficientes, y la frecuencia fundamental seremos capaces de regenerar la señal. Necesitaremos también la frecuencia de muestreo, obtenida al principio, y recordar el tamaño de ventana que estamos usando.

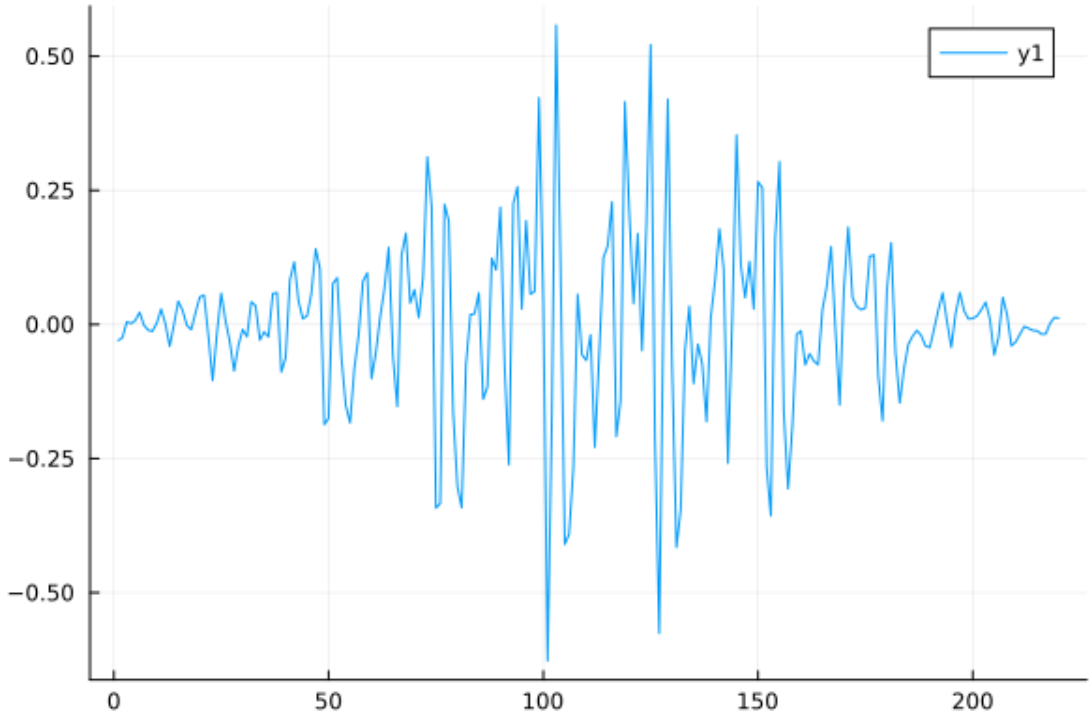
Lo que deberemos hacer será, con los coeficientes LPC, obtenidos en la fase de análisis, sintetizaremos la señal como hicimos en los ejercicios anteriores.

[ ]:
```
function synth_signal(trama, fs, orden)
    """Dada una trama, suponiéndola enventanada, se calcula todo hasta que se␣
 ↪sintetiza.
    args:
    trama: trama a sintetizar
    fs: frecuencia de muestro
    orden: orden del filtro de análisis
    returns:
    synth_sign: señal sintetizada
    """
    params = get_params_4_lpc(trama,fs,10)
    quasi_time = 0.02
    num_samples = Int(length(trama))
    if params[1]
```

```
        a_k = params[3]
        error_prediccion = obtener_error_prediccion(trama, a_k)
        trama_sintetizada = sintetizar_trama(a_k, error_prediccion);
        return trama_sintetizada
    else
        g =
        synth_sign = randn(num_samples) .*0.001 # Hay que reducir el ruido,␣
↪sino será muy alto.
        return synth_sign
    end
end
```

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

[ ]:
```
synth = synth_signal(tramas[14],fs,10)
plot(synth)
```



Ahora, crearemos una función para que recoja toda la señal, y la separe por tramas, las analices y las sintetice.

[ ]:
```
function synth_all_sign(s,fs,quasi,orden)
    """
    Recibe una señal entera de voz, y la sintetiza con los parámetros deseados.␣
↪La divide en subtramas, sintetiza
```

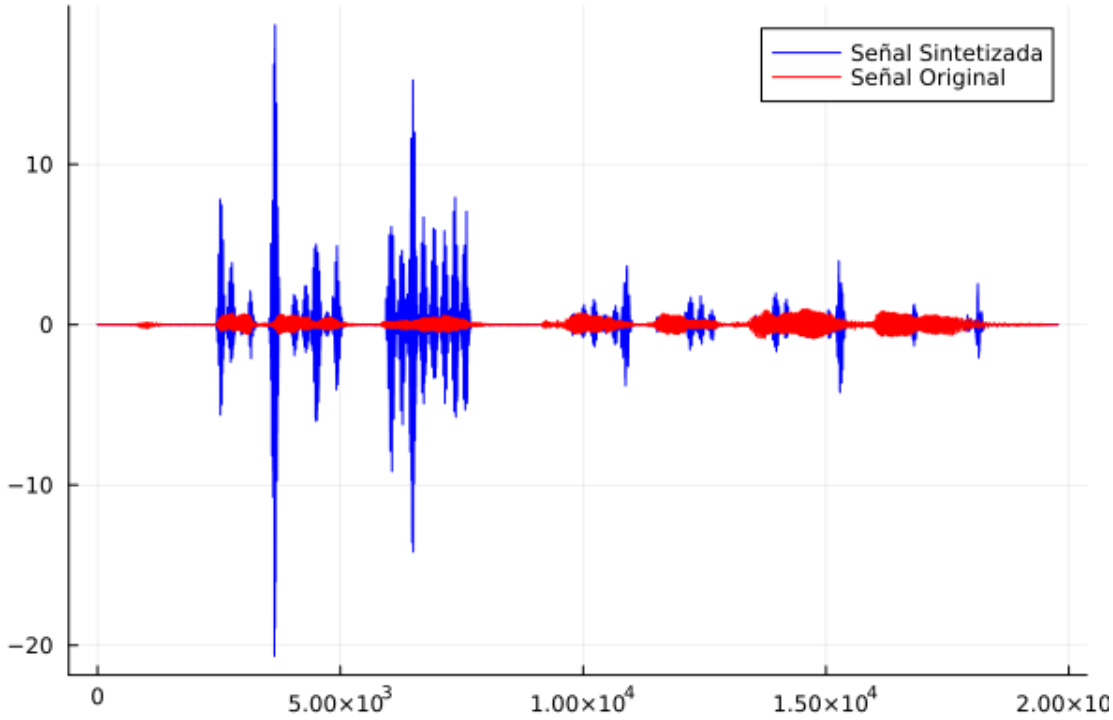
```
    cada trama y vuelve a unir todas las tramas.
    args:
    s: señal de voz
    fs: frecuencia de muestro
    quasi: tiempo mediante el cual la voz es cuasi-estacionaria
    orden: orden del filtro de análisis.
    """
    synth_sign = []
    num_samples_window = Int(round(fs*quasi))
    tramasSign = trama_to_tramas(s, num_samples_window)
    for trama in tramasSign
        trama_synth = synth_signal(trama,fs,orden)
        synth_sign = vcat(synth_sign,trama_synth)
    end
    return synth_sign
end
```

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

[ ]:
```
synth_all = synth_all_sign(confront, fs,0.02,10);
```

Veamos ahora la diferencia entre las gráficas de la señal original y la señal sintetizada.

[ ]:
```
plot(synth_all,label="Señal Sintetizada",color=:blue)
plot!(confront, label="Señal Original",color=:red)
```



Escuchemosla y lleguemos a una conclusión:

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

```
19778
```

Como vemos, las gráficas y el sonido son parecidos, pero no iguales. Hay diversos motivos por los que esto puede pasar:

```
- Desde errores acumulados en los cálculos de los coeficientes LPC, lo que puede provocar que e

- Afecta también la decisión de sonido sonoro o sordo, si se tuviera un algoritmo más potente p
```

Vemos que en la gráfica de la señal sintetizada hay ciertos picos, esos picos son porque el decisor no está decidiendo bien el tipo de sonido que es, se generan durante unos microsegundos, sonido como si fueran sonidos sordos pero realmente no lo son.

1.1 Ejercicio 5: Reflexión

Durante esta práctica hemos aprendido a como de sensibles son las señales de voz. Para poder parametrizarla, debemos trabajar con extremo cuidado en todas las fases de la parametrización, si en alguna etapa cometieramos un error, podríamos hacer que la calidad se vea reducida drásticamente.

Es importante tener en cuenta, que cuando se desea diseñar un vocoder como se ha intentado aquí, se debe tener bien claro cuales son los pasos a seguir, pues no vale de nada malgastar recursos, por ejemplo, calculando los coeficientes de la autocorrelación para un sonido sordo, cuando más tarde, eso no se transmitirá.

El vocoder LPC es muy útil para disminuir la tasa binaria necesaria para transmitir voz. Pues la transmisión se reduce a unos parámetros más sencillos que todo los valores de la voz, y además, proporciona más robustez frente a ruido e interferencias.

Se ha obtenido en esta práctica diversos conocimientos, como el funcionamiento de los filtros de síntesis y análisis, la cuasi-estacionaridad de la voz, propiedad que permite funcionar a los vocoder LPC. Mediante el desarrollo de la práctica, también se han adquirido diversos métodos y algoritmos, que al final no fueron utilizados por complejidad en la implementación, pero que podrían ser usados para, cálculo de energías, cálculo de tasas, decisiones de sonidos, cálculos espectrales, etc.