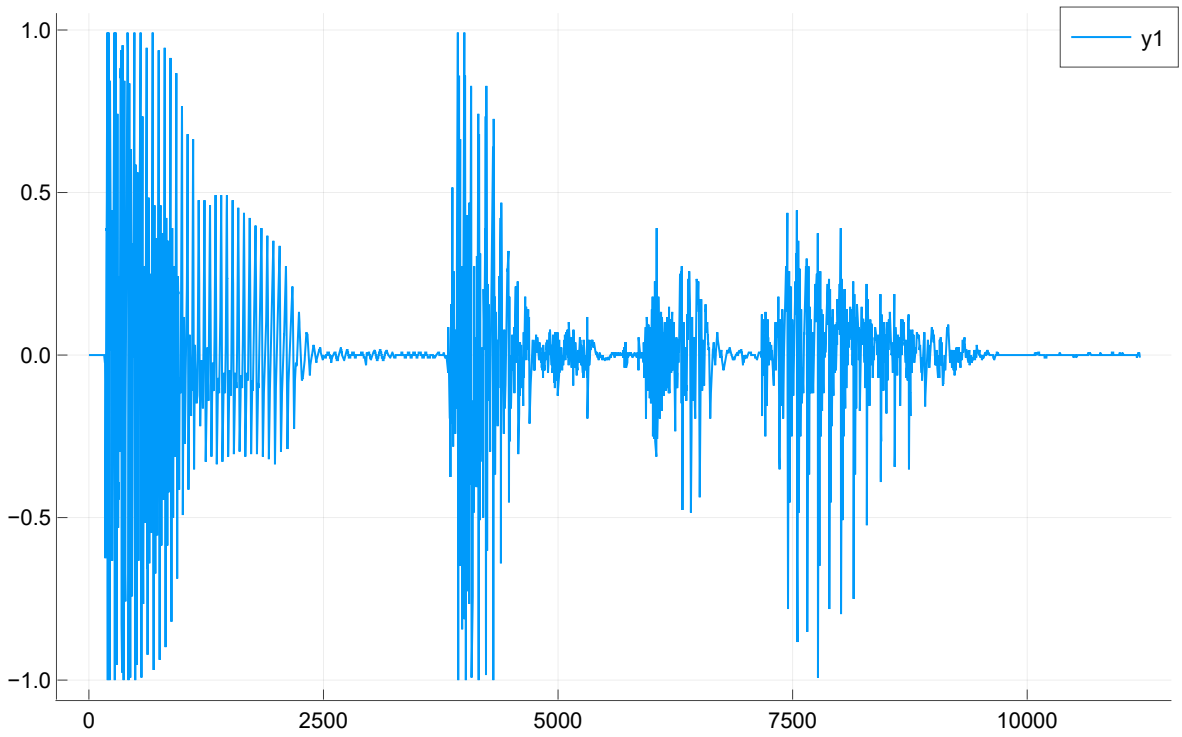
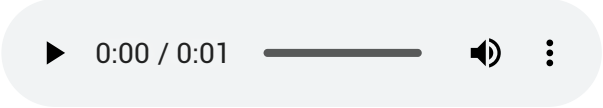
```
PlotlyBackend()
```

```
begin
    using Pkg
    using DSP
    using Plots
    using WAV
    using FFTW
    using SampledSignals
    using PlutoUI
    plotly()
end
```

Leitura do sinal




```
begin
    sinal, fs = wavread("antarctica.wav")
    plot(sinal)
end
```




```
SampleBuf(sinal,fs)
```

Estimação do filtro de trato vocal

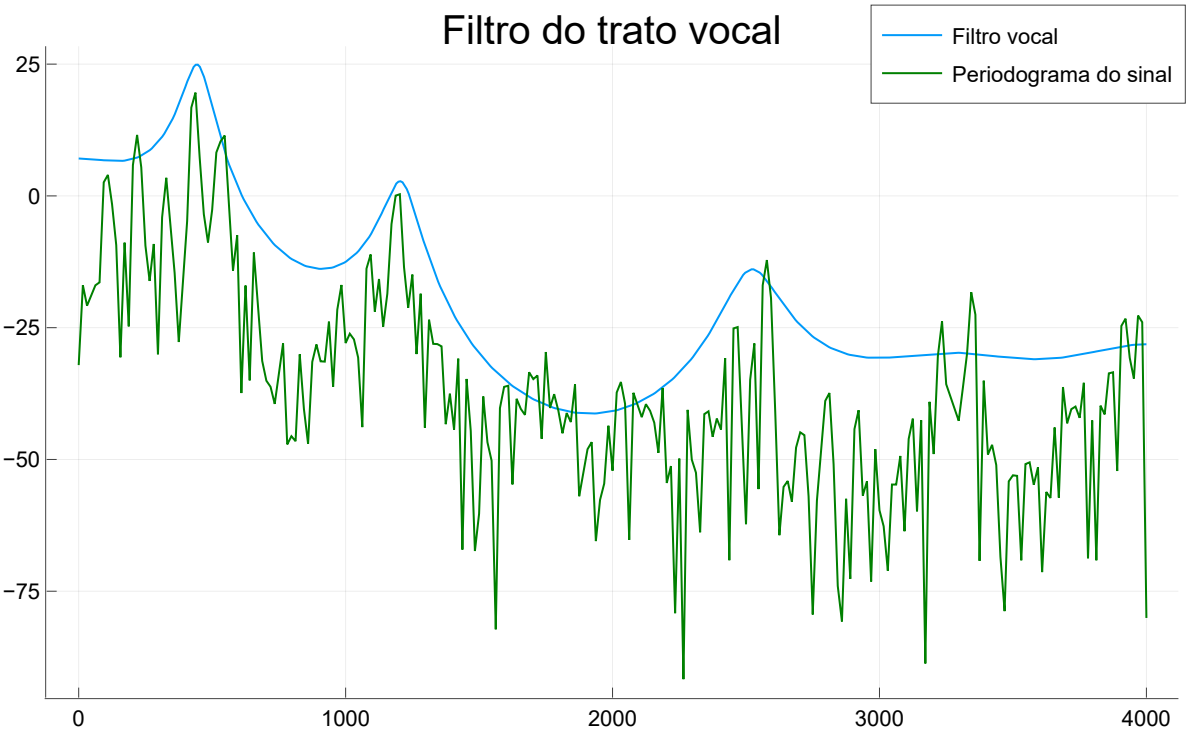
```
▸ Periodogram([9.75362e-6, 5.58422e-5, 3.56856e-5, 4.56347e-5, 5.57062e-5, 5.94225e-5, 0.0
```

```
begin
    #Estimação dos coeficientes do filtro
    trecho = sinal[200:439]
    ak10, pot_erro = lpc(trecho.*hamming(240), 10)

    filtro10 = PolynomialRatio([1],[1;ak10])
    ω = range(0,π, length= 512)
    H = freqz(filtro10, ω)

    #peridiograma do trecho
    per = periodogram(trecho; fs = fs, nfft = 512)
end
```

Ganho = 0.1

```
0.1
```

```
G_
```

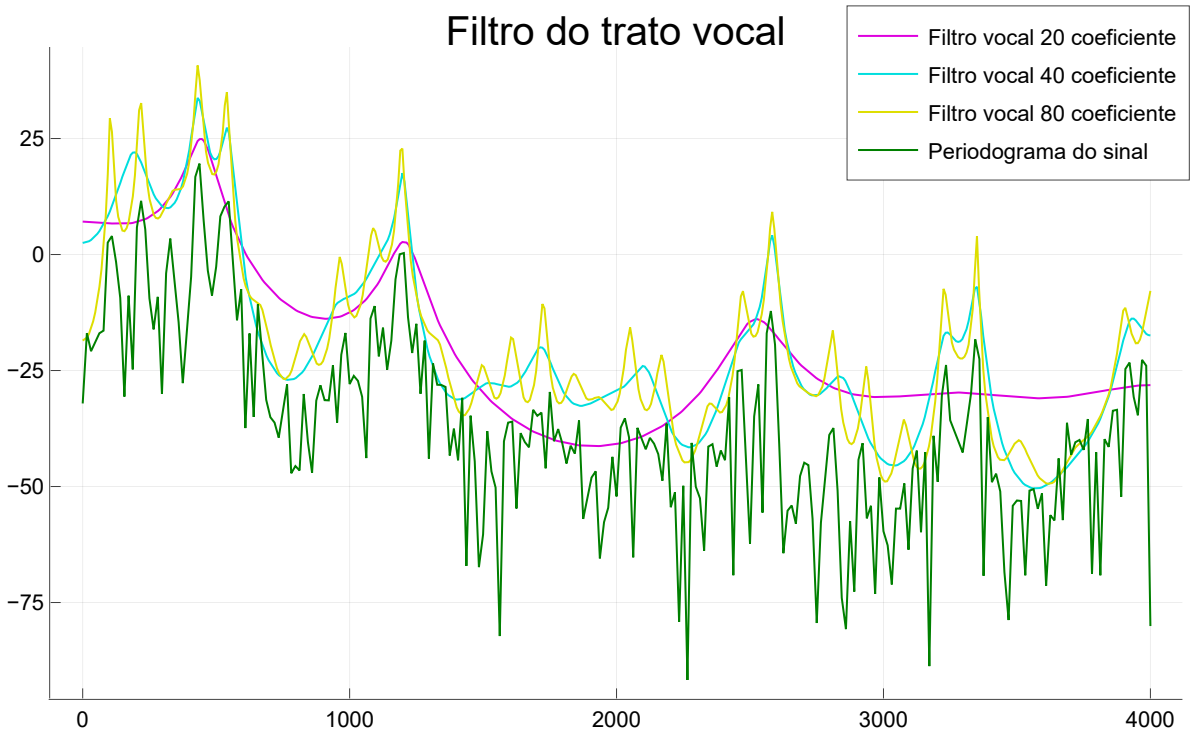



```
begin
    plot(ω/π*fs/2, pow2db.(G_*abs.(H).^2), label = "Filtro vocal")
    plot!(per.freq, pow2db.(per.power*fs/π), label = "Periodograma do sinal", color
= :green)
    plot!(title="Filtro do trato vocal")

end
```

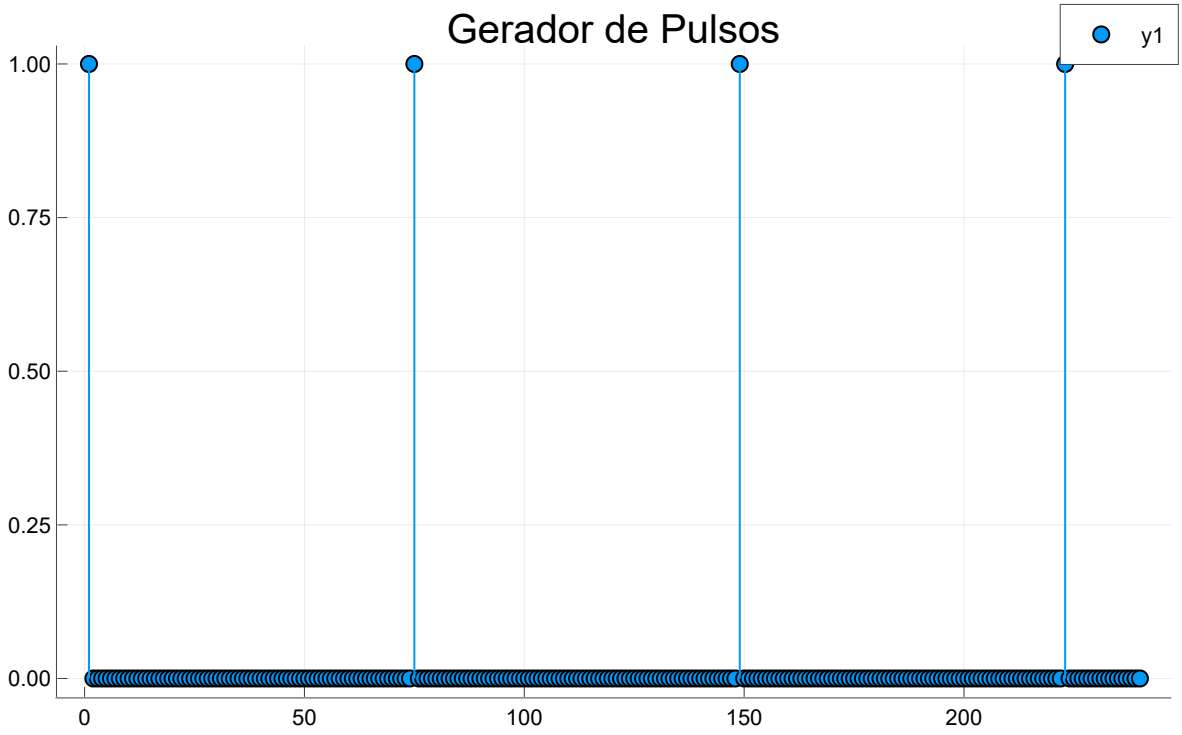
Filtro com mais coeficientes

```
[1.08769+0.0im, 1.08222+0.183253im, 1.06545+0.37099im, 1.03628+0.568212im, 0.9927+0.781
```

```
begin
    ak20, p = lpc(trecho.*hamming(240), 10)
    ak40, p = lpc(trecho.*hamming(240), 40)
    ak80, p = lpc(trecho.*hamming(240), 80)

    filtro20 = PolynomialRatio([1],[1;ak20])
    filtro40 = PolynomialRatio([1],[1;ak40])
    filtro80 = PolynomialRatio([1],[1;ak80])

    H20 = freqz(filtro20, ω)
    H40 = freqz(filtro40, ω)
    H80 = freqz(filtro80, ω)
end
```



Testando estimador de Pitch




```
begin
    Tp = pitch(trecho)
    pulsos = zeros(length(trecho))
    pulsos[1:Tp:end] .= 1
    plot(pulsos, marker=:circle, line= :stem)
    plot!(title = "Gerador de Pulsos")
end
```

0:00 / 0:00

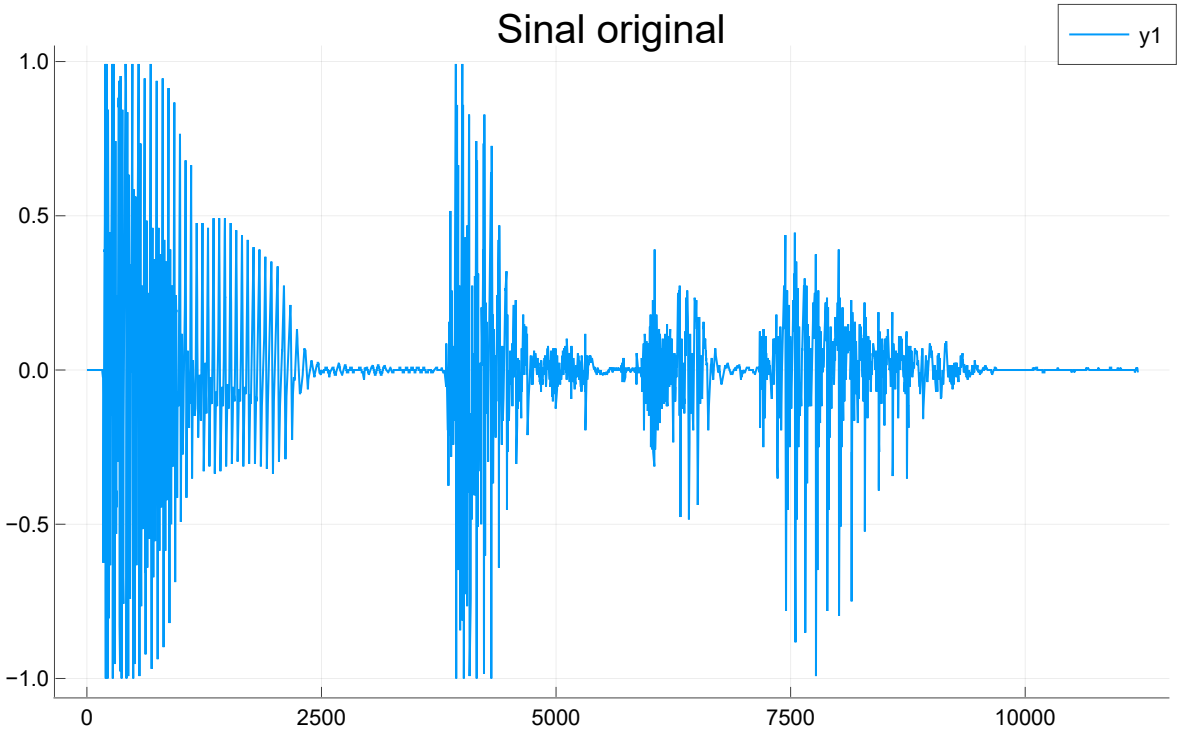
```
SampleBuf(filt(filtro10,pulsos), fs)
```

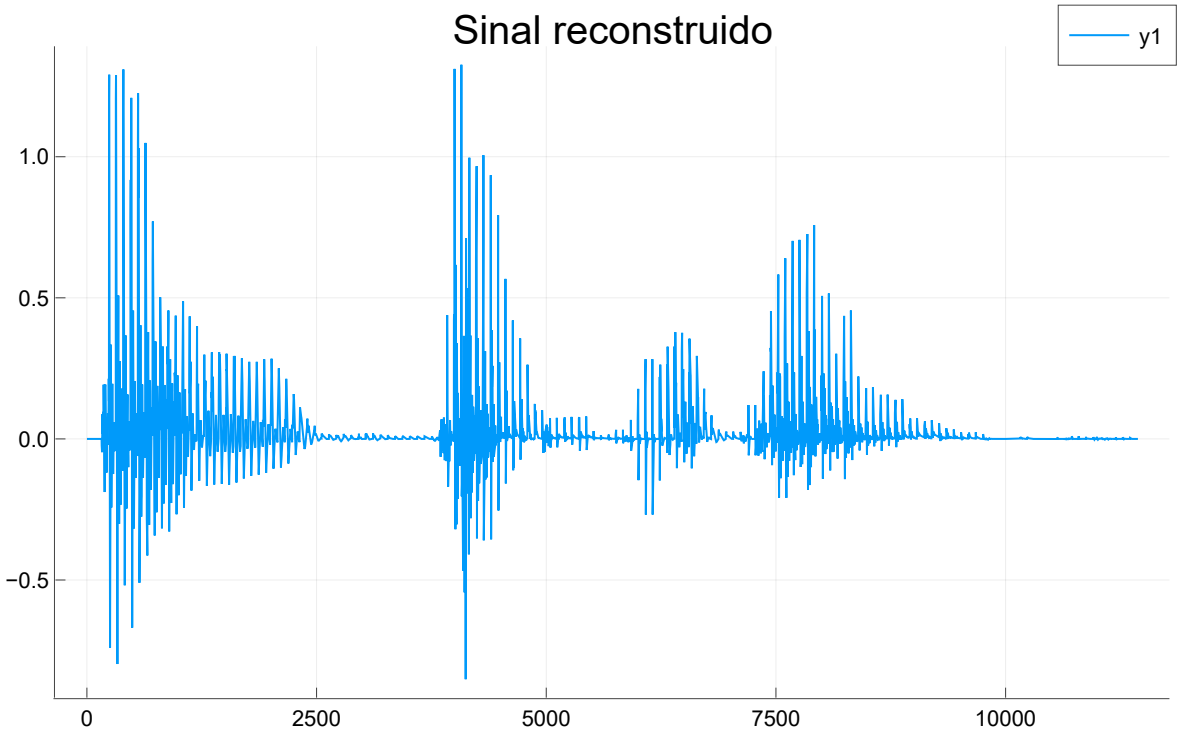
0:00 / 0:00

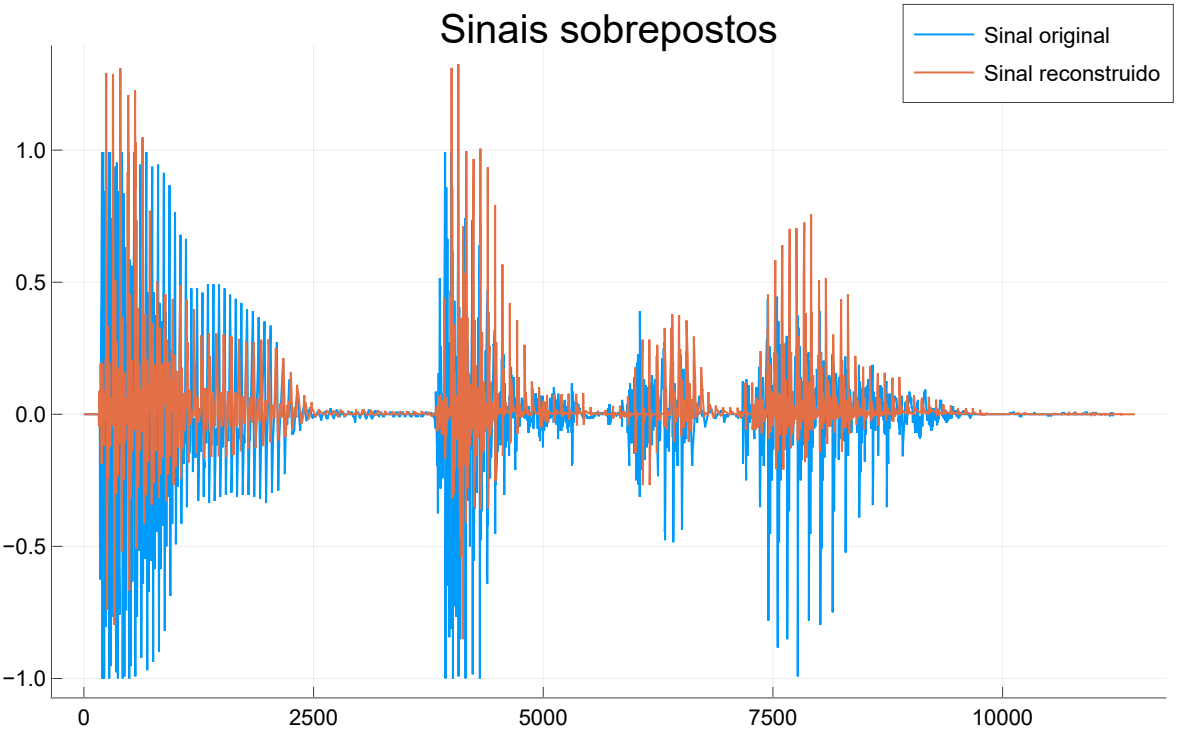
```
SampleBuf(trecho, fs)
```

Reconstrução do sinal completo

```
begin

    tamanho_analise = 240
    tamanho_sintese = 80
    off_set_analise = (Int)((tamanho_analise - tamanho_sintese)/2) #80 amostras pra
tras e pra frente



    sinal_analise = vcat(zeros(off_set_analise),sinal, zeros(tamanho_analise -
mod(length(sinal),tamanho_analise)), zeros(off_set_analise))

    sinal_sintese =  zeros(length(sinal_analise))

    for i in range(1,length = (Int)(length(sinal_analise)/tamanho_sintese)-2)

        trecho_analise = sinal_analise[i*tamanho_sintese+1-off_set_analise:
(i+1)*tamanho_sintese+off_set_analise]

        if sum(trecho_analise) != 0
            ak, pot = lpc(trecho_analise.*hamming(tamanho_analise), 10)
            tp = pitch(trecho_analise)

            filtro = PolynomialRatio([1],[1;ak])

            if tp == 0
                G = sqrt(pot)
                excitacao = randn(tamanho_sintese)
            else
                G = sqrt(pot*tp)
                excitacao = zeros(tamanho_sintese)
                excitacao[1:Tp:end] .= 1
            end
            trecho_sintese = filt(G*filtro,excitacao)

            sinal_sintese[i*tamanho_sintese+1:(i+1)*tamanho_sintese] =
trecho_sintese
        else
            sinal_sintese[i*tamanho_sintese+1:(i+1)*tamanho_sintese] .= 0
        end

    end

end
```


Sinal original
y1
1.0
0.5
0.0
−0.5
−1.0
0
2500
5000
7500
10000


Sinal reconstruido
y1
1.0
0.5
0.0
−0.5
0
2500
5000
7500
10000



Sinal Original

0:00 / 0:01

Sinal reconstruído

0:00 / 0:01

```
wavwrite(sinal_sintese, "sintese_LPC.wav", Fs = fs)
```

Functions

Main.workspace2.pitch

```
"""
    pitch(quadro)
Adaptado de T. Dutoit (2007)
Calcula o período fundamental de um trecho de voz de 30ms amostrado a 8kHz.  Retorna
0 se o trecho é surdo (não-vozeado).

"""
function pitch(quadro)
    ai,ξ = lpc(quadro,10)
    h = PolynomialRatio([1],[1;ai])
    lpc_residual=filt(h,quadro)
    M = length(lpc_residual)
    C=xcorr(lpc_residual, lpc_residual; padmode = :longest)
    Cxx=C[M:end] / C[M] # normaliza a autocorrelação pela variância
    Cxx[1:26] .= 0
    Amax,Imax = findmax(Cxx[1:min(133,length(quadro))]) # Limita T para 60-300Hz
    if (Amax > 0.20) # teste bem simples para sinal sonoro/surdo
       T0 = Imax - 1
    else
       T0 = 0
    end

    return T0
end
```

pow2db (generic function with 1 method)

```
function pow2db(x)

    return 20log10(x)
end
```