

Main.workspace2.fxfilt

```

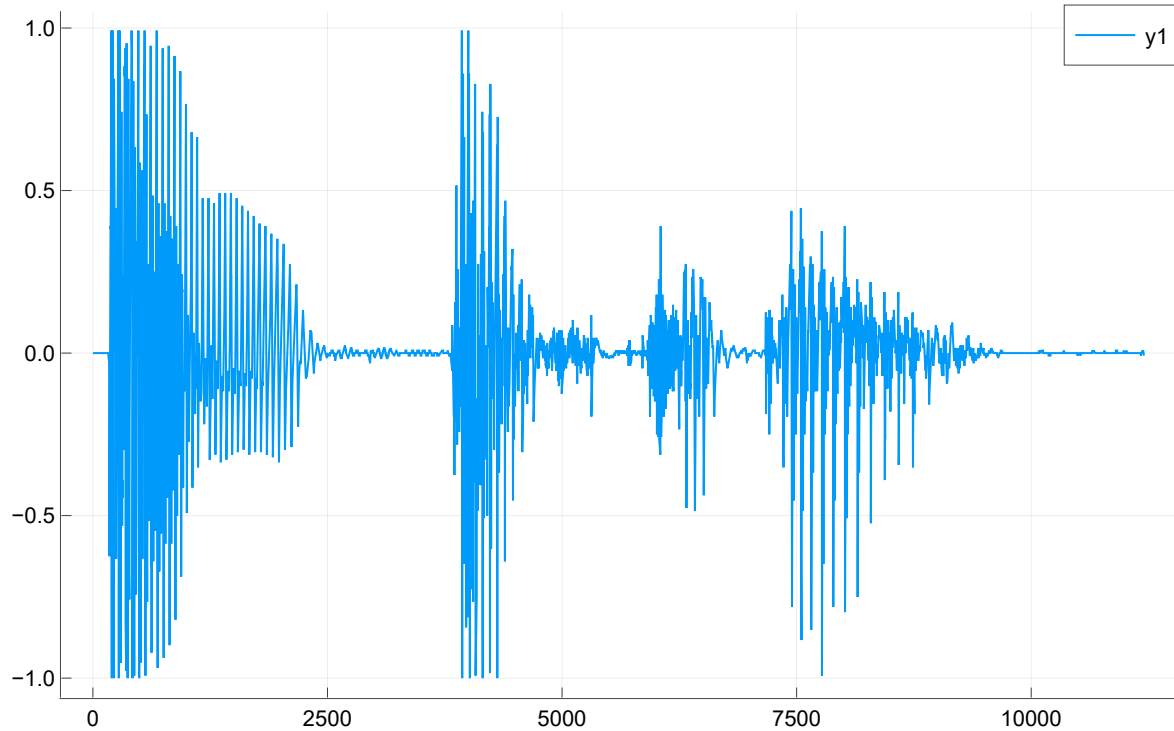
•
•   begin
•       using Pkg
•       using DSP
•       using Plots
•       using WAV
•       using FFTW
•       using SampledSignals
•       using PlutoUI
•       using LinearAlgebra
•       using FixedPoint
•       plotly()
•
•       include("fxfilt.jl")
•   end

```

```

• Pkg.add("FixedPoint")

```



```

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

```

▶ 0:00 / 0:01 — 🔊 ⋮

```

• SampleBuf(sinal,fs)

```

80

```

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

```

```
► [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```

• begin
•     K = 2
•     Q = 512
•     N = tamanho_sintese
•     func_base = randn(N,Q)
•     func_filtradas = zeros(N,Q)
•
•     zs = zeros(10)
• end

```



```

• begin
•     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, G = lpc(trecho_analise .* hamming(tamanho_analise),10)
•
•             # aq = Fixed{Int16, 7-1}.(ak)
•
•             subquadro =
trecho_analise[off_set_analise+1:off_set_analise+tamanho_sintese]
•
•
•             filtro = PolynomialRatio([1],[1;ak])
•
•             for coluna in 1:Q
•                 func_filtradas[:, coluna] = filt(filtro, func_base[:,coluna])
•             end
•
•             y0 = filt(filtro,zeros(tamanho_sintese))
•
•             e0 = subquadro .- y0
•
•             ganhos, indices = find_Nbest_components(e0, func_filtradas, K);
•
•             trecho_sintese = func_filtradas[:, indices[1]] * ganhos[1] +
func_filtradas[:, indices[2]] * ganhos[2]
•
•             sinal_sintese[i*tamanho_sintese+1:(i+1)*tamanho_sintese] =
trecho_sintese
•             else
•
•                 end
•
•             end
•
•         end
•
•     end
• end

```

159

```
• tamanho_analise - mod(length(sinal),tamanho_analise) + 80
```

▶ 0:01 / 0:01   ⋮

```
• SampleBuf(sinal_sintese, fs)
```

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

```
▶ [0.0331715, 0.0753998, 0.127185, 0.193439, 0.272028, 0.319213, 0.400425, 0.444292, 0.471
```

```
• begin
•     ak = 0
•     i = 20
•     trecho_analise = sinal_analise[i*tamanho_sintese+1-off_set_analise:
• (i+1)*tamanho_sintese+off_set_analise]
•
•     if sum(trecho_analise) != 0
•
•         ak, G = lpc(trecho_analise .* hamming(tamanho_analise),10)
•
•         # aq = Fixed{Int16, 7-1}.(ak)
•
•         subquadro =
• trecho_analise[off_set_analise+1:off_set_analise+tamanho_sintese]
•
•         filtro = PolynomialRatio([1],[1;ak])
•
•         for coluna in 1:Q
•             func_filtradas[:, coluna] = filt(filtro, func_base[:,coluna])
•         end
•
•         y0 = filt(filtro,zeros(tamanho_sintese))
•
•         e0 = subquadro .- y0
•
•         ganhos, indices = find_Nbest_components(e0, func_filtradas, K);
•
•         trecho_sintese = func_filtradas[:, indices[1]] * ganhos[1] +
• func_filtradas[:, indices[2]] * ganhos[2]
•
•         sinal_sintese[i*tamanho_sintese+1:(i+1)*tamanho_sintese] =
• trecho_sintese
•     else
•
•     end
• end
```

```
-4.898801525386531
```

```
• minimum(func_base)
```

Main.workspace2.find_Nbest_components

```
•  
•  
• """  
•     function find_Nbest_components(s, codebook_vectors, N)  
• Adaptado de T. Dutoit (2009)  
• Acha os N melhores componentes de s a partir dos vetores no livro-código  
• codebook_vectors, minimizando o quadrado do erro erro = s -  
• codebook_vectors[indices]*ganhos.  
• Retorna (ganhos, indices)  
• """  
• function find_Nbest_components(signal, codebook_vectors, N)  
•  
•     M, L = size(codebook_vectors)  
•     codebook_norms = zeros(L)  
•  
•     for j = 1:L  
•         codebook_norms[j] = norm(codebook_vectors[:,j])  
•     end  
•  
•     gains = zeros(N)  
•     indices = ones(Int, N)  
•  
•     for k = 1:N  
•         max_norm = 0.0  
•         for j = 1:L  
•             beta = codebook_vectors[:,j] * signal  
•             if codebook_norms[j] != 0  
•                 component_norm = abs(beta)/codebook_norms[j]  
•             else  
•                 component_norm = 0.0  
•             end  
•             if component_norm > max_norm  
•                 gains[k] = beta/(codebook_norms[j]^2)  
•                 indices[k] = j  
•                 max_norm = component_norm  
•             end  
•         end  
•         signal = signal - gains[k]*codebook_vectors[:,indices[k]]  
•     end  
•     return gains, indices  
• end
```