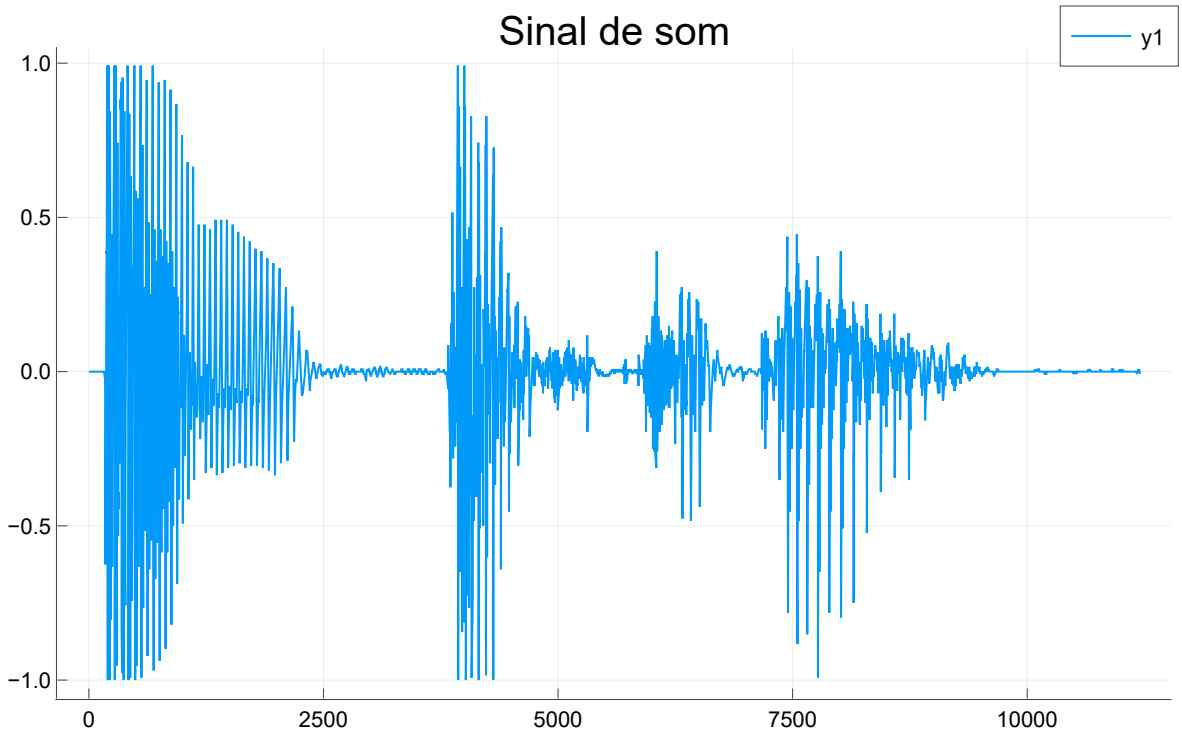
# Codificado de voz CELP

Gabriel Tavares 10773801

Guilherme Reis 10773700

`Main.workspace2.fxfilt`

## Leitura do Sinal




0:00 / 0:01

# Análise e Sintese do sinal

Para fazer a reconstrução do sinal iremos dividí-lo em trechos de 240 amostras andando a passos de 80 amostras. A cada trecho de 240 amoostras iremos reconstruir as 80 amostras centrais desse trecho.

Em todos os trechos faremos o seguinte processo:

- Fazer a análise LPC do filtro de trato vocal daquele trecho
- Filtrar a família de funções bases por esse filtro
- Passar as funções base pela função FindBest que acha a melhor correspondência entre as funções e o centro do trecho

Dessa forma para cada trecho teremos os coeficientes de trato vocal, os indices das melhores K funções da família de funções bases e os ganhos dessas funções.

```
80
```

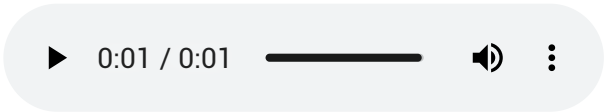
```
begin
    tamanho_analise = 240
    tamanho_sintese = 80
    off_set_analise = (Int)((tamanho_analise - tamanho_sintese)/2) #80 amostras pra tras e pra frente
end
```

```
begin
    K = 2
    Q = 512
    N = tamanho_sintese
    func_base = randn(N,Q)
    func_filtradas = zeros(N,Q)
    noprint
end
```

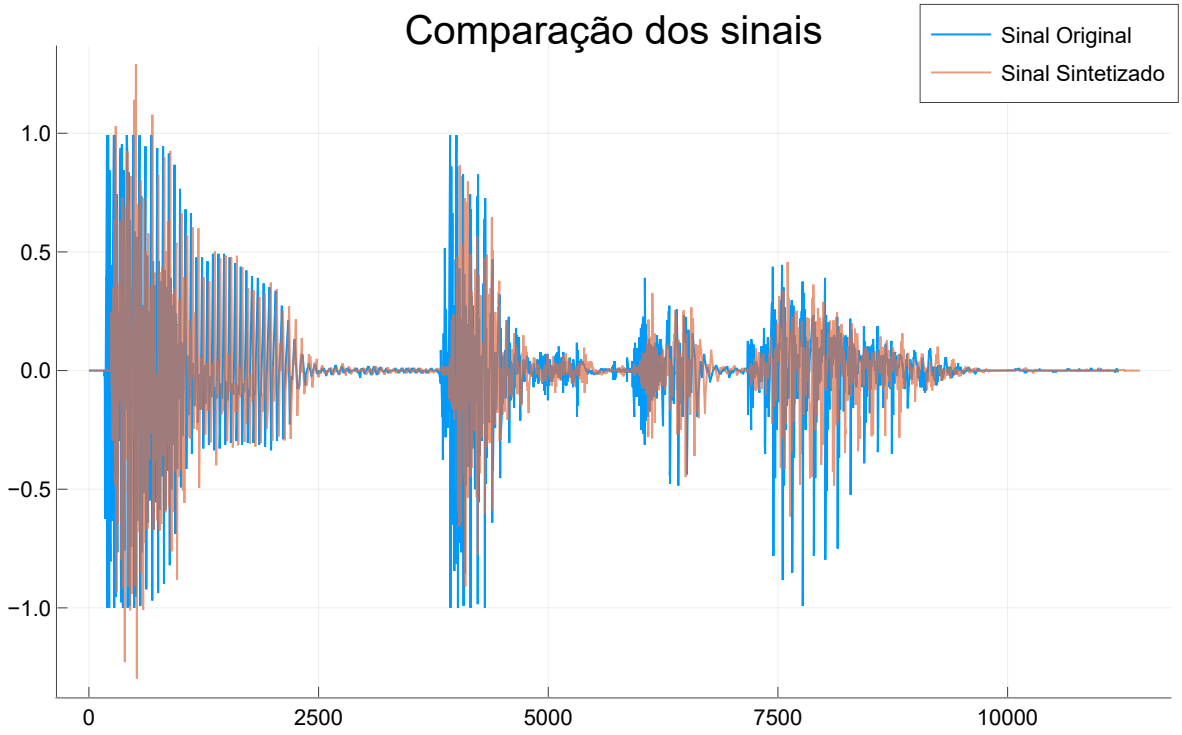
```
begin
    sinal_analise = vcat(zeros(off_set_analise), sinal, zeros(tamanho_analise - mod(length(sinal),tamanho_analise)), zeros(off_set_analise))

    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)


            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

            ganhos, indices = find_Nbest_components(subquadro, 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
```

# Resultados

Após a análise podemos fazer a reconstrução desse sinal com os parâmetros achados em cada trecho.




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




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

Podemos ver que o sinal sintetizado se assemelha bastante ao sinal original como o desejado, e o tamanho do sinal é bastante menor, pois só possui os parâmtros da sintese

# Funções

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
```

noprint =

```
noprint = md""
```