Gabriel Tavares

nusp : 10773801

```
PlotlyBackend()
```

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

Leitura do sinal

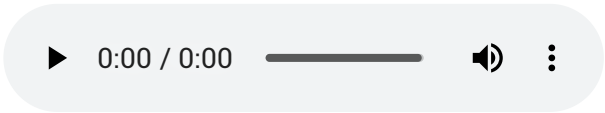
```
md" ## Leitura do sinal"
```


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

Velocidade = **2.5**

Mudança de taxa de amostragem

Aqui iremos alterar a velocidade de reprodução a partir da taxa de amostragem do sinal. Dessa forma o sinal irá tocar mais devagar ou mais rápido conforme a velocidade desejada




```
SampleBuf(sinal, vel_reproducao*fs)
```

Vemos que a velocidade do sinal se altera, mas a tonalidade do sinal também é modificada. Quando aumentamos a velocidade, a voz fica mais aguda e quando diminuimos a velocidade, ela fica mais grave

Mudança do codificação de voz

Para corrigir o problema a cima, a estratégia é usar um algoritmo de reconstrução de voz.

O algoritimo irá reconstruir trechos maiores ou menos de voz, dependendo da velocidade desejada.

Por exemplo, se queremos uma velocidade de x2.0, o algoritimo irá analisar trechos de 240 amostras a passos de 80 amostras, mas irá reconstruir apenas 40 amostras por passo (e não 80). Dessa forma o tamanho do sinal irá diminuir (ele ira tocar mais rápido) mas a tonalidade da voz não irá se alterar, porque os fonemas ainda estão sendo analisados e reconstruídos conforme a velocidade original.

```
80
```

```
begin
    tamanho_analise = 240
    tamanho_sintese = floor(Int,80/vel_reproducao)
    passo_analise = 80 #quantas amostras andam a cada trecho
    sobreposicao = 80  #numero de amostras de sobreposicao entre analises
                         # significa 80 amostras do trecho anterior e do futuro
end
```

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

```
begin
    #adiciona zeros no sinal para deixar o número certo de analises no sinal
    sinal_analise = vcat(zeros(passo_analise), sinal, zeros(passo_analise - mod(length(sinal),passo_analise)), zeros(passo_analise))

    num_trechos = floor(Int,length(sinal_analise)/(tamanho_analise - 2*sobreposicao))

    sinal_sintese =  zeros(tamanho_sintese*(num_trechos-2))
    md""
end
```

```
begin
    for i in 1:num_trechos-3

        trecho_analise = sinal_analise[i*passo_analise+1-sobreposicao:
(i+1)*passo_analise+sobreposicao]

        if sum(trecho_analise) != 0

            ak, G = lpc(trecho_analise .* hamming(tamanho_analise),10)


            subquadro =
trecho_analise[floor(Int,(tamanho_analise -tamanho_sintese)/2) + 1 : floor(Int,
(tamanho_analise -tamanho_sintese)/2)+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

        end

    end

end
```

0:00 / 0:00

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

Observamos que o sinal foi acelerado, mas manteve a tonalidade da voz do locutor, como desejado.

Functions

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
```