Projeto Processamento de Voz

Gustavo Vieira Alcantara

September 23, 2024

4.1 Filtragem com esquecimento

4.1.a Gráficos dos espectros

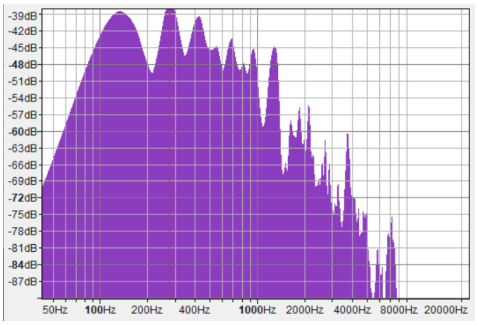

Figure 1: Espectro base

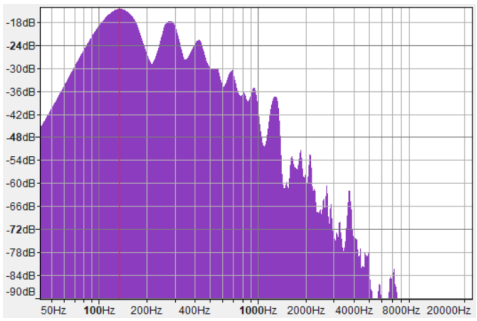

Figure 2: Utilizando alfa = 0,98

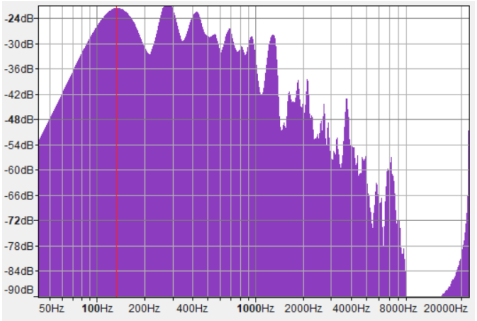

Figure 3: Utilizando alfa = -0,98

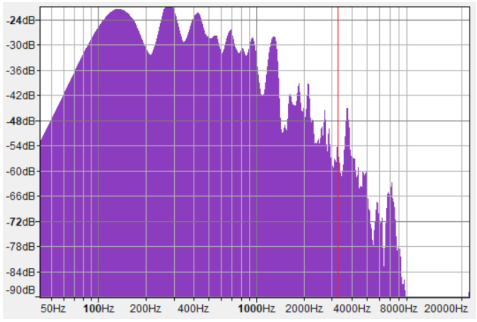

Figure 4: Utilizando alfa = 0,5

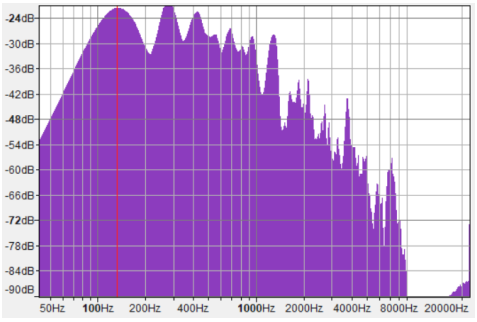

Figure 5: Utilizando alfa = -0,5

4.1.b Análise dos resultados

Um valor de alfa próximo de 1, como 0.98, resulta em uma resposta lenta, ou seja, o filtro dá maior peso às amostras passadas. Isso gera um efeito de suavização forte no sinal de áudio, preservando as frequências de baixa variação. O som será mais estável, mas pode parecer mais "lento" ou atenuado.

Um valor negativo de alfa com magnitude próxima de 1, como -0.98, inverte as fases das amostras anteriores, criando um efeito de distorção ou oscilação no áudio. O som pode ter um efeito inverso, com certas frequências sendo realçadas de maneira não natural, gerando uma sensação de eco ou distorção agressiva.

Um valor positivo menor, como 0.5, faz com que o filtro tenha uma resposta mais rápida, dando mais peso às amostras atuais em comparação às passadas. Isso resulta em menos suavização e maior fidelidade ao sinal original, mas ainda com alguma atenuação de ruídos. O som permanece relativamente claro, com suavização moderada.

Com alfa = -0.5, o filtro aplica uma inversão de fase com menor magnitude, criando uma distorção perceptível no áudio, mas de forma menos intensa que no caso de alfa = -0.98. Esse valor causa uma modificação das frequências, resultando em um som com efeito de "interferência" ou um leve distorcimento, mas menos agressivo.

4.1.c Código

```
function normalized = normalize(x)
    normalix = max ( abs(min(x)), abs(max(x)) );
    normalized = x/normalix;
endfunction
function filtered = oblivion_filter(signal, alfa)
    y = zeros(1,length(signal));
    y(1) = signal(1);
    for i = 2:length(y)
        y(i) =  alfa*y(i-1) + signal(i);
    end
    filtered = y;
endfunction

fs = 44100;
voice = wavread('source');
channels = size(voice);

mono_voice = voice(1,:);

//Filtro do esquecimento
alfa_filtered = oblivion_filter(mono_voice, 0.98);
normalized = normalize(alfa_filtered);
wavwrite(normalized, fs, 'alfa098.wav');

alfa_filtered = oblivion_filter(mono_voice, -0.98);
normalized = normalize(alfa_filtered);
wavwrite(normalized, fs, 'alfa-098.wav');

alfa_filtered = oblivion_filter(mono_voice, 0.5);
normalized = normalize(alfa_filtered);
wavwrite(normalized, fs, 'alfa05.wav');

alfa_filtered = oblivion_filter(mono_voice, -0.5);
normalized = normalize(alfa_filtered);
wavwrite(normalized, fs, 'alfa-05.wav');
```

4.2 Filtragem de média móvel

4.2.a Gráficos dos espectros

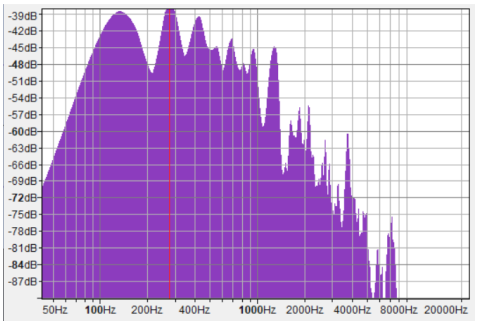

Figure 6: Espectro base

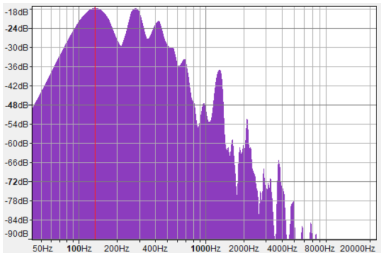

Figure 7: Primeira imagem.

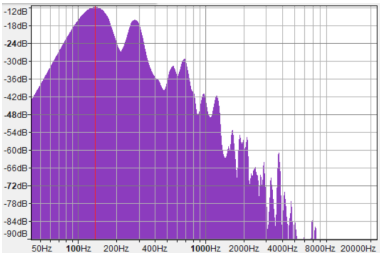

Figure 8: Segunda imagem.

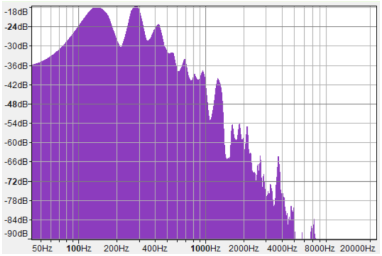

Figure 9: Terceira imagem.

4.2.b Análise dos resultados

Com m = 50, o filtro de média móvel atua suavemente, resultando em uma atenuação leve nas frequências mais altas, enquanto as baixas frequências são preservadas. O espectro de frequência mostra uma redução discreta nos componentes de alta frequência, mas sem afetar drasticamente as características principais do sinal.

Quando m aumenta para 100, o filtro se torna mais agressivo na redução de frequências altas. Isso começa a reduzir significativamente o conteúdo de altas frequências no espectro. O efeito é que as frequências médias e baixas são mais preservadas, enquanto frequências acima de uma certa faixa são progressivamente eliminadas.

Com m = 1000, o filtro suaviza intensamente o sinal, o que equivale a uma filtragem muito forte das frequências altas. O espectro de frequência mostra uma remoção quase completa das frequências mais altas, e o sinal se torna dominado apenas pelas frequências mais baixas. Na prática, a maior parte das variações rápidas (frequências altas) é eliminada.

4.2.c Código

```
function normalized = normalize(x)
    normalix = max ( abs(min(x)), abs(max(x)) );
    normalized = x/normalix;
endfunction

function filtered =  mean_avarage(signal, M)
    y = zeros(1,length(signal));
    y(1) = signal(1);
    slice =  zeros(1,M);

    for i = 1:length(signal)
        for j = 1:(M-1)
            slice(j) = slice(j+1);
        end
        slice(M) = signal(i);
        if i < M then
            y(i) = sum(slice)/i;
        else
            y(i) = sum(slice)/M;
        end
    end
    filtered = y;
endfunction

fs = 44100;
voice = wavread('source');
channels = size(voice);

mono_voice = voice(1,:);

avarage_filtered = mean_avarage(mono_voice, 50);
normalized = normalize(avarage_filtered);
wavwrite(normalized, fs, 'window50.wav');

avarage_filtered = mean_avarage(mono_voice, 100);
normalized = normalize(avarage_filtered);
wavwrite(normalized, fs, 'window100.wav');

avarage_filtered = mean_avarage(mono_voice, 1000);
normalized = normalize(avarage_filtered);
wavwrite(normalized, fs, 'window1000.wav');
```

4.3 Correlação de sinais

4.3.a Gráficos dos espectros

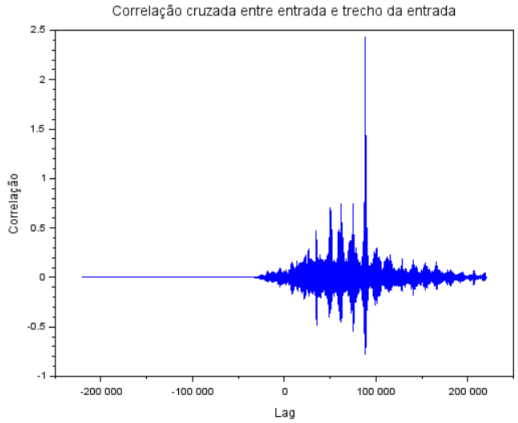

Figure 10: Correlação

4.3.b Análise dos resultados

Quando aplicamos a correlação cruzada entre um sinal completo e uma fatia dele, estamos essencialmente verificando em que ponto do sinal original a fatia se alinha melhor. Isso resulta em um pico de correlação na posição onde a fatia se encaixa com mais precisão dentro do sinal original. Nesse caso, como aplicamos uma fatia localizada em 2 segundos do sinal podemos ver o pico em l = 2*fs

4.3.c Código

```
fs = 44100;
voice = wavread('source');
channels = size(voice);

mono_voice = voice(1,:);

sliced = mono_voice(2*fs:3*fs);
correlation = xcorr(mono_voice,sliced);

n = length(correlation);
lag = -(n-1)/2 : (n-1)/2;


plot(lag, correlation);
xlabel('Lag');
ylabel('Correlacao');
title('Correlacao cruzada entre entrada e trecho da entrada');
```

4.4 Alteração de taxa

4.4.a Gráficos dos espectros

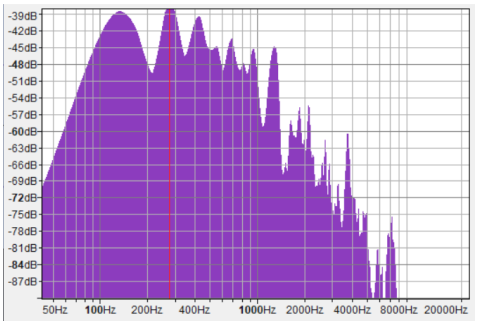

Figure 11: Espectro base

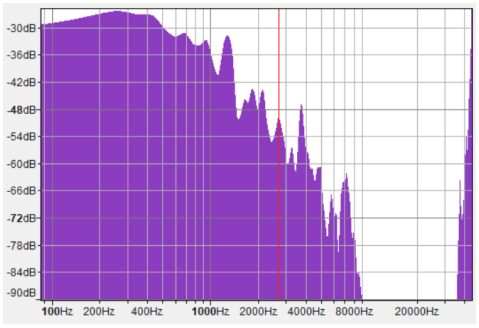

Figure 12: Taxa dobrada pela inserção de zeros

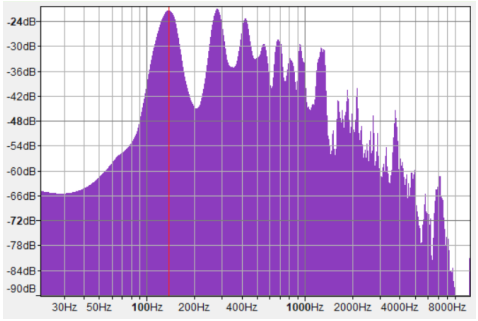

Figure 13: Taxa reduzida pela metade

4.4.b Análise dos resultados

A inserção de zeros no domínio do tempo introduz imagens espectrais no espectro de frequência. Isso ocorre porque a adição de zeros cria réplicas do espectro original nas frequências mais altas, tornando o espectro periódico com componentes adicionais acima da frequência de Nyquist do sinal original. Essas réplicas podem causar distorções e aliasing se não forem removidas. Já a subamostragem introduziu aliasing. Isso acontece porque a frequência de amostragem reduzida pode não ser suficiente para capturar corretamente as variações do sinal, violando o teorema de Nyquist. O espectro do sinal subamostrado foi diferente do espectro original. As frequências que estavam presentes no sinal original apareceram distorcidas. O espectro resultante apresentou componentes em frequências indesejadas devido ao aliasing. Com a redução da taxa de amostragem, a resolução em frequência também diminui, resultando em uma representação menos precisa do conteúdo espectral do sinal original.

4.4.c Código

```
function normalized = normalize(x)
    normalix = max ( abs(min(x)), abs(max(x)) );
    normalized = x/normalix;
endfunction
function doubled = tax_double(signal)
    y = zeros(1,2*length(signal));
    for i = 1:length(signal)
        y(2*i) = signal(i);
    end
    doubled = y;
endfunction


fs = 44100;
voice = wavread('source');
channels = size(voice);

mono_voice = voice(1,:);

double_tax_signal = tax_double(mono_voice);
normalized = normalize(double_tax_signal);
wavwrite(normalized, 2*fs, 'double_tax.wav');

half_tax_signal = mono_voice(1:2:length(mono_voice));
normalized = normalize(half_tax_signal);
wavwrite(normalized, fs, 'half_tax.wav');
```