Universidade Federal de Minas Gerais

2023/1

Processamento de Sinais - ELE042
Prof. Hilton de Oliveira Mota

Trabalho Prático de Processamento de Sinais

Autor: *Matrícula:*
Pedro Henrique de Oliveira Barbosa *2019108156*
Giovanni Martins de Sá Júnior *2017001850*

13 de janeiro de 2024

Sumário

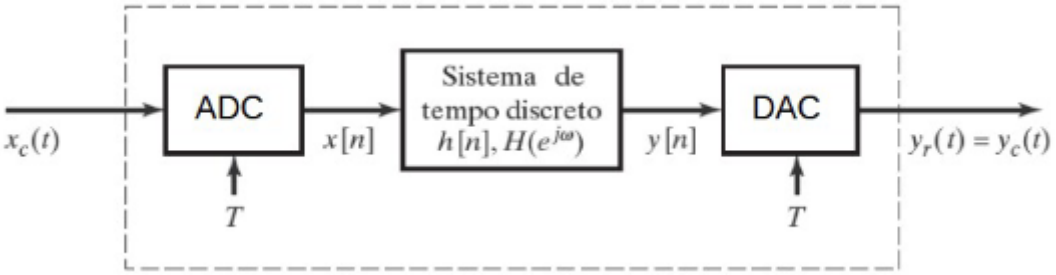
1 Introdução

O áudio digital é uma representação amostrada de ondas de pressão do ar, sendo captado e convertido em uma corrente elétrica por meio de um dispositivo de áudio, geralmente um microfone. A digitalização proporciona vantagens significativas em relação ao domínio analógico, permitindo a aplicação de ferramentas de processamento avançadas, tais como compressão, filtragem, equalização e reconhecimento de fala. Essa abordagem digital não apenas viabiliza a manipulação de sinais existentes, mas também possibilita a geração artificial de sons, ampliando consideravelmente as aplicações em síntese musical, fala, efeitos sonoros, entre outros.

Sistemas digitais de processamento de áudio seguem uma estrutura que envolve a digitalização do sinal, processamento por algoritmos implementados em sistemas computacionais e, por fim, a reconstrução do sinal digitalizado. A manipulação da frequência de amostragem e o descarte de amostras emergem como técnicas para alterar os tons do áudio, mas é crucial evitar valores excessivamente altos no descarte, a fim de evitar aliasing, conforme estabelecido pelo limite de Nyquist. A análise do espectro de frequências, através da Transformada de Fourier, revela as alterações provocadas pela compressão ou dilatação do sinal. A compensação dessas alterações, particularmente através de filtros anti-aliasing digitais, é essencial para preservar a integridade do espectro durante o processamento digital.

No âmbito da manipulação da taxa de amostragem, introduzem-se conceitos de dizimadores (redução da taxa de amostragem) e interpoladores (aumento da taxa de amostragem). A interpolação, efetuada por algoritmos que "suavizam"a forma da onda, possui efeitos similares a uma filtragem passa-baixas. Esses processos, exemplificados numericamente, demonstram a influência na compressão do espectro de frequências. Com isso, almeja-se desenvolver um sistema que permita dizimar ou interpolar sinais de áudio em diferentes taxas M e L. Este sistema visa atender a aplicações específicas, como compressão de dados e modificação da velocidade de reprodução, sem afetar os tons originais, abrindo possibilidades inovadoras no processamento de áudio digital.

Figura 1: Sistema de Processamento de Áudio



2 Descrição da Solução

A implementação desenvolvida tem como objetivo realizar a dizimação e interpolação de um sinal de áudio, permitindo ao usuário escolher as taxas de dizimação (M) e interpolação (L). Nesse sentido, a abordagem é valiosa em aplicações de compressão de dados para redução de armazenamento e modificação da velocidade de reprodução sem alterar os tons. Mais abaixo, será detalhado a implementação desenvolvida.

2.1 Implementação Básica

A implementação do código começa carregando um sinal de áudio e sua taxa de amostragem. Em seguida, o sinal é reproduzido para audição. O usuário pode especificar as taxas de dizimação (M = 2) e interpolação (L = 3).

O código utiliza funções para visualização no domínio do tempo e frequência do sinal original. Em seguida, a fase de compressão é executada usando a função *dizimarSinal*. Esta função utiliza um filtro passa-

baixa (implementado pela função *filtroKaiser*) para pré-processar o sinal antes da dizimação propriamente dita. Os resultados da compressão são visualizados no domínio do tempo, e são exibidas informações sobre a taxa de dizimação (M) e os parâmetros do filtro utilizado. Após a fase de compressão, o código realiza a interpolação do sinal comprimido usando a função *interpolarSinal*. Semelhante à etapa de compressão, a interpolação envolve o uso de um filtro passa-baixa. Os resultados da interpolação são novamente visualizados no domínio do tempo, com informações sobre a taxa de interpolação (L) e os parâmetros do filtro.

Finalmente, os espectros antes e depois da dizimação e interpolação são exibidos, e os sinais resultantes são reproduzidos para permitir a comparação auditiva.

```
% Entradas do usuario
[x, fs] = audioread('EuGostoDeDSP.wav');
% Carrega o arquivo de audio e sua taxa de amostragem
% Reproduz o audio carregado
sound(x, fs);
% Fator de compressao
M = 2;
% Fator de interpolacao
L = 3;

% Visualizacao e FFT
figure(1);
% Plota o sinal no dominio do tempo
plotis_tempo(x, fs);
ylabel('x(t)');
title('x(t)')
axis tight;

figure(2);
% Plota o espectro de frequ ncia
plotis_freq(x, fs)
ylabel('X(ejw)');
title('X(ejw)')
axis tight;

% Compressao
[dizimado, window, N, beta, fil_coefs] = dizimarSinal(x, M, fs);

figure(3);
% Plota o sinal comprimido no dominio do tempo
plotis_tempo(dizimado, fs)
ylabel('x(t)');
title('x(t) comprimido')
axis tight;
dim = [0.2 0.5 0.3 0.3];
str = {'M=' num2str(M,'%d')};
annotation('textbox',dim,'String',str,'FitBoxToText','on');

figure(4);
% Plota a janela utilizada para compressao
plotis_n(window);
ylabel('w[n]');
title('Janela para compress o');
axis tight;
dim = [0.2 0.5 0.3 0.3];
str = {'N=' num2str(N,'%d'), 'Beta=' num2str(beta,'%d')};
annotation('textbox',dim,'String',str,'FitBoxToText','on');

figure(5);
% Plota a resposta ao impulso do filtro utilizado
impz(fil_coefs);

figure(6);
```

```
% Plota a resposta em frequ ncia do filtro utilizado
freqz(fil_coefs);

figure(7);
% Plota o espectro apos a dizimacao
plotis_freq(dizimado, fs)
ylabel('X(ejw)');
title('Espectro p s  dizima  o')
axis tight;

sound(dizimado, fs);                        % Reproduz o audio comprimido

% Interpolacao
[interpolado, window, N, beta, fil_coefs] = interpolarSinal(dizimado, L, fs);

figure(8);
% Plota o sinal interpolado no dom nio do tempo
plotis_tempo(interpolado, fs)

ylabel('x(t)');
title('x(t) interpolado')
axis tight;
dim = [0.2 0.5 0.3 0.3];
str = {'L=' num2str(L,'%d')};
annotation('textbox',dim,'String',str,'FitBoxToText','on');

figure(9);
% Plota a janela utilizada para interpolacao
plotis_n(window);
ylabel('w[n]');
title('Janela para interpola  o');
axis tight;
dim = [0.2 0.5 0.3 0.3];
str = {'N=' num2str(N,'%d'), 'Beta=' num2str(beta,'%d')};
annotation('textbox',dim,'String',str,'FitBoxToText','on');

figure(10);
% Plota a resposta ao impulso do filtro utilizado
impz(fil_coefs);

figure(11);
% Plota a resposta em frequ ncia do filtro utilizado
freqz(fil_coefs);

figure(12);
% Plota o espectro ap s a interpolacao
plotis_freq(interpolado, fs)
ylabel('X(ejw)');
title('Espectro p s  interpola  o')
axis tight;

sound(interpolado, fs);                    % Reproduz o  udio  interpolado

% Fim

function [dizimado, window, N, beta, fil_coefs] = dizimarSinal(sinal, M, fsamp)
    wp = fsamp/(2*M);
    wr = (5*fsamp)/(2*4*M);

    [filtrado, window, N, beta, fil_coefs] = filtroKaiser(sinal, fsamp, [wp, wr]);

    dizimado = [];

    for n = 1:(length(sinal)/M)
```

```
        dizimado(n) = sinal(n*M);
    end
end

function [interpolado, window, N, beta, fil_coefs] = interpolarSinal(sinal, L, fsamp)
    inter = [];
    aux = 1;

    for n = 1:(L*length(sinal))
        if(aux > L)
            aux = 1;
        end

        if(aux == 1)
            if(1+round(n/L) < length(sinal))
                inter(n) = sinal(1+round(n/L));
            end
        else
            inter(n) = 0;
        end
        aux = aux + 1;
    end

    wp = fsamp/(2*L);
    wr = (5*fsamp)/(2*4*L);

    [interpolado, window, N, beta, fil_coefs] = filtroKaiser(inter, fsamp, [wp, wr]);
end

function [filtrado, window, N, beta, fil_coefs] = filtroKaiser(sinal, fsamp, fcuts)
    mags = [1 0];
    devs = [0.01 0.01];

    [N,Wn,beta,FILTYPE] = kaiserord(fcuts,mags,devs,fsamp);
    window = kaiser(N+1, beta);

    fil_coefs = fir1(N,Wn,window);
    filtrado = filter(fil_coefs,1,sinal);
end

function plotis_tempo(sinal, fsamp)
    time = [];
    Ts = 1/fsamp;
    for i = 1:length(sinal)
        time(i) = i*Ts;
    end
    plot(time,sinal);
    xlabel('t(s)');
end

function plotis_n(sinal)
    n = [];
    for i = 1:length(sinal)
        n(i) = i;
    end
    plot(n, sinal);
    xlabel('n');
end

function plotis_freq(sinal,fs)
    N = length(sinal);
    f = [];
    f(1) = -fs/2;
    Xw = fftshift(abs(1/length(sinal)*fft(sinal)));
```

```
    for i = 2:length(sinal)
        f(i) = f(i-1)+(fs*1/N);
    end
    f = f/1000;
    plot(f,Xw)
    xlabel('f(kHz)');
end
```

Listing 1: Implementação do Código

2.2 Projeto pelo método da janela de Kaiser, a partir de um filtro FIR (Questão 1.4)

Para a elaboração do algoritmo tanto na dizimação quanto a interpolação foram utilizados filtros passa-baixas – utilizando a função *filter()* do *MATLLAB* – projetados dinamicamente segundo os critérios discutidos na introdução. Em consoante com os requisitos especificados no item a) e b) e c). A definição do filtro passa-baixas projetado dinamicamente pelo método da janela de Kaiser ocorre na função *filtroKaiser*.

```
function [filtrado, window, N, beta, fil_coefs] = filtroKaiser(sinal, fsamp, fcuts)
    % Par metros do filtro
    mags = [1 0];
    devs = [0.01 0.01];

    % Projeto do filtro usando o m todo da janela de Kaiser
    [N, Wn, beta, FILTYPE] = kaiserord(fcuts, mags, devs, fsamp);
    window = kaiser(N+1, beta);  % Gera a janela de Kaiser

    fil_coefs = fir1(N, Wn, window);  % Projeta o filtro FIR usando a janela

    % Filtra o sinal de entrada
    filtrado = filter(fil_coefs, 1, sinal);
end
```

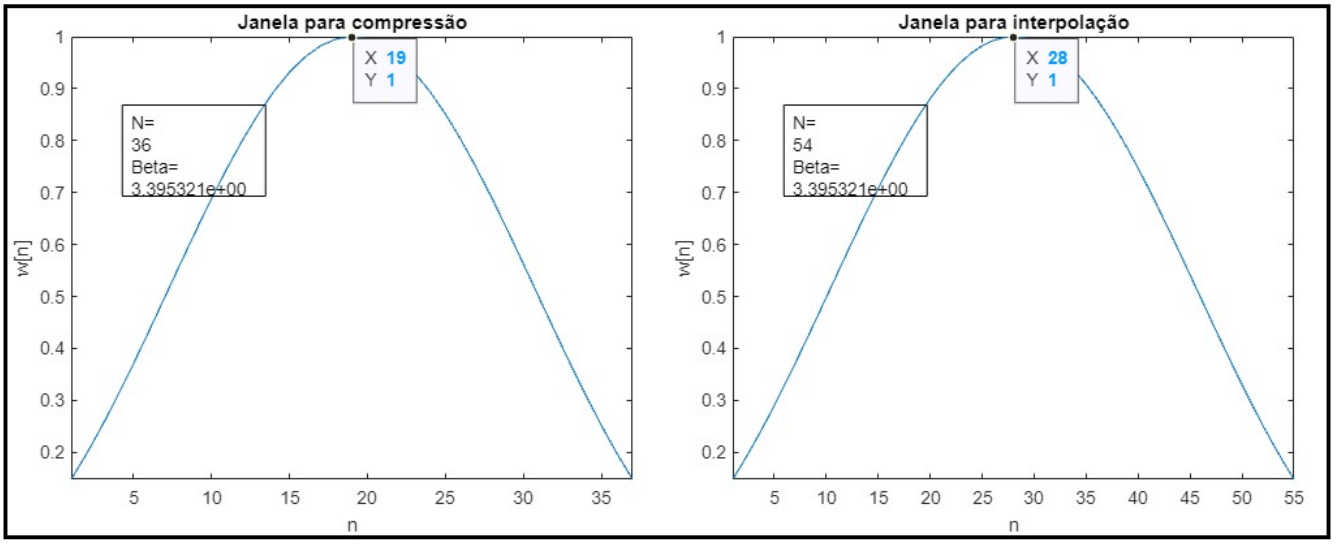
Listing 2: Implementação do Código

Nesta função:

- **mags** e **devs** especificam as magnitudes desejadas nas faixas de passagem e rejeição e as tolerâncias de desvio, respectivamente.
- **kaiserord** é usado para calcular a ordem **N**, a frequência de corte normalizada **Wn**, o parâmetro **beta** e o tipo de filtro.
- **kaiser** gera a janela de Kaiser com base nos parâmetros calculados.
- **fir1** projeta os coeficientes do filtro FIR utilizando a janela de Kaiser gerada.
- **filter** aplica o filtro FIR ao sinal de entrada.

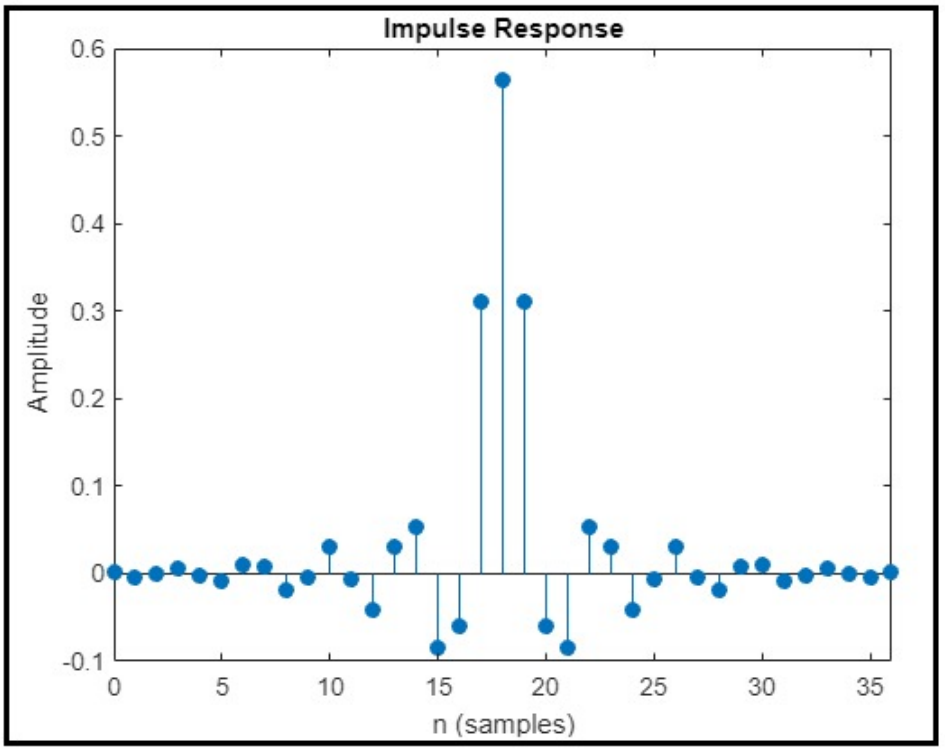
Este processo visa atender aos critérios discutidos na introdução, considerando as frequências de bordas das faixas de passagem e rejeição, bem como a máxima distorção nas faixas de passagem e rejeição. A função *filtroKaiser* é utilizada tanto na etapa de dizimação quanto na de interpolação para projetar os filtros passa-baixas necessários.

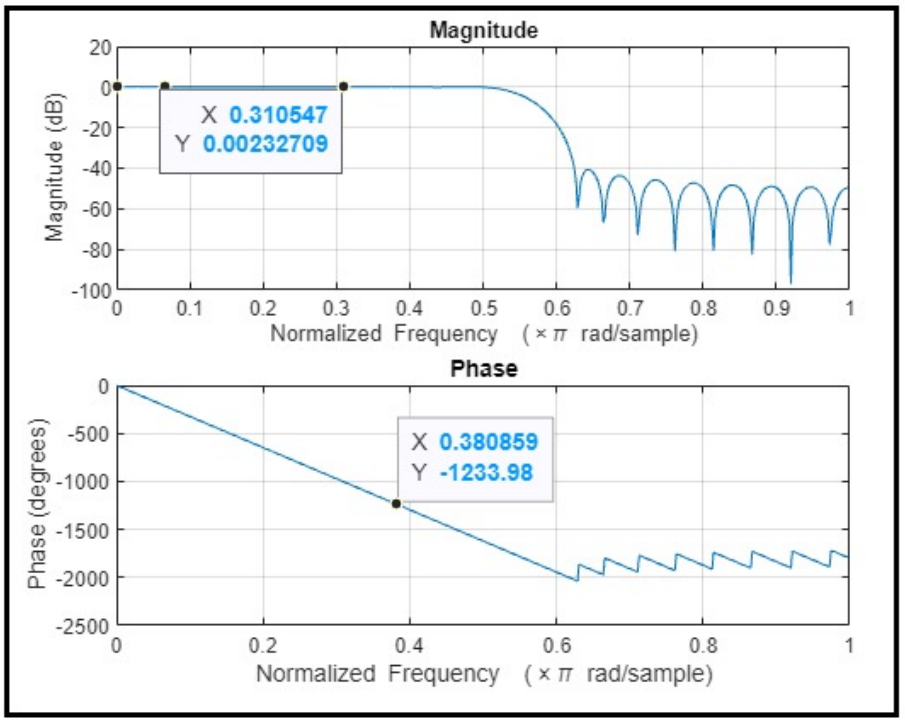
2.2.1 Gráficos das janelas w[n]×n e reporte as ordens N e os parâmetro beta encontrados para cada filtro (Questão 1.4.d)



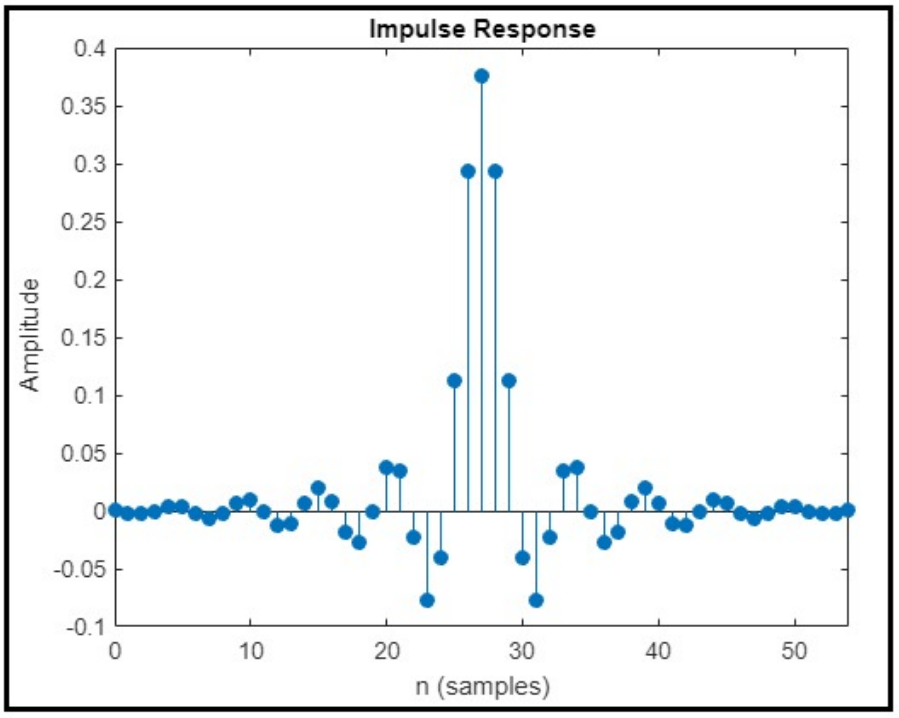
2.2.2 Resposta ao impulso h[n]×n e as respostas de magnitude e fase na faixas de frequências (Questão 1.4.e)

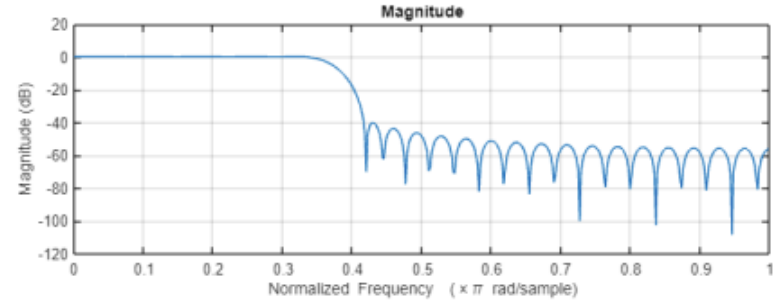
- Para dizimação:

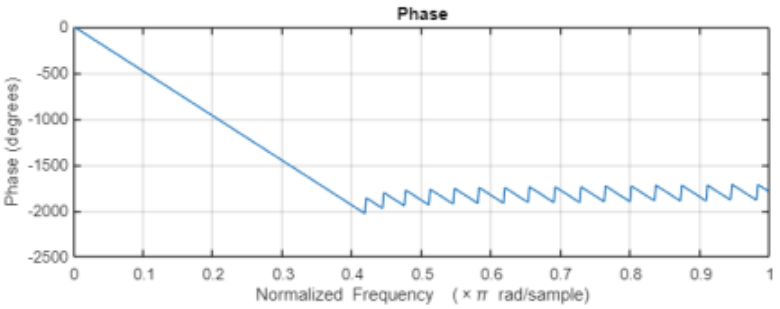



Magnitude
X 0.310547
Y 0.00232709
Magnitude (dB)
Normalized Frequency (× π rad/sample)
Phase
X 0.380859
Y -1233.98
Phase (degrees)
Normalized Frequency (× π rad/sample)

- Para interpolação:

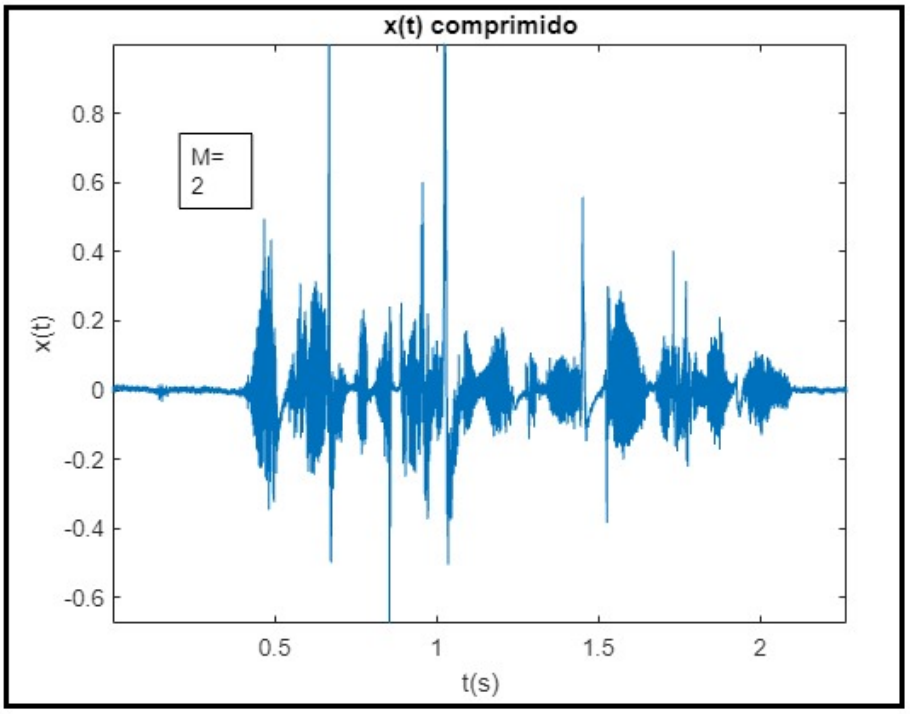


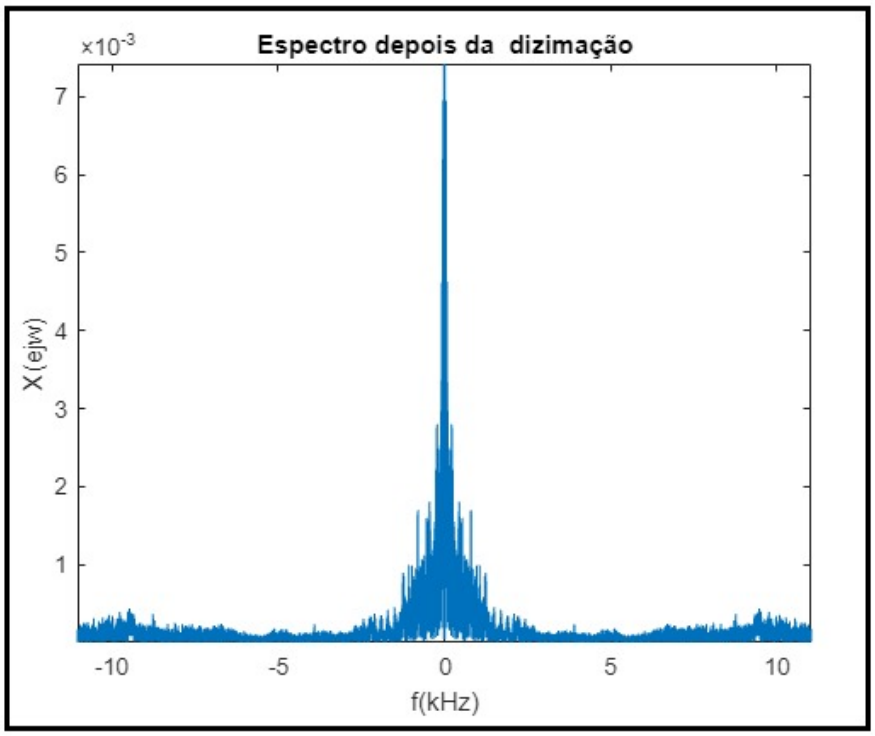




2.2.3 Expectros de frequência dos sinais após a dizimação/interpolação (Questão 1.4.f)

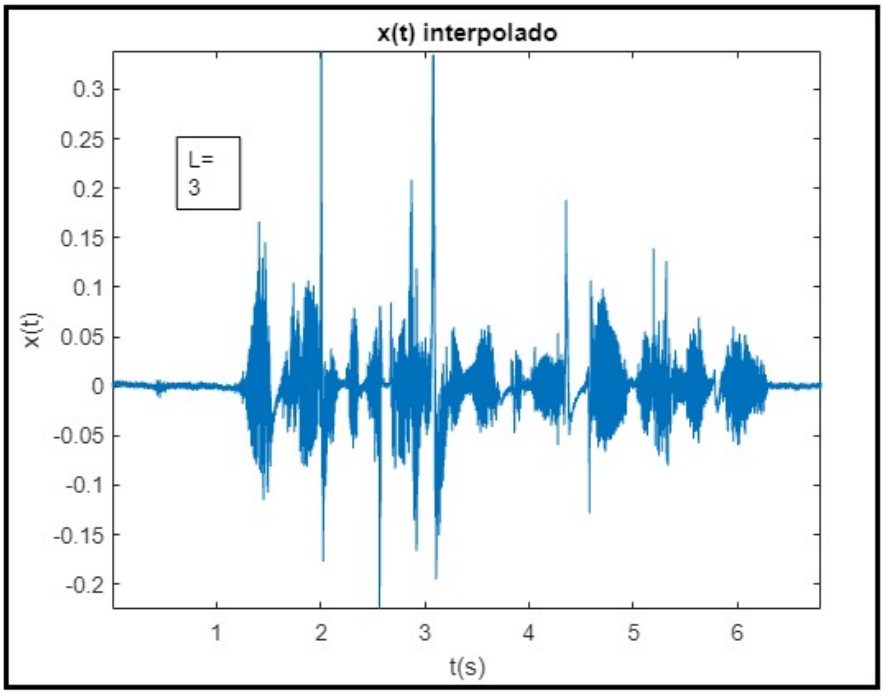
- Para dizimação:

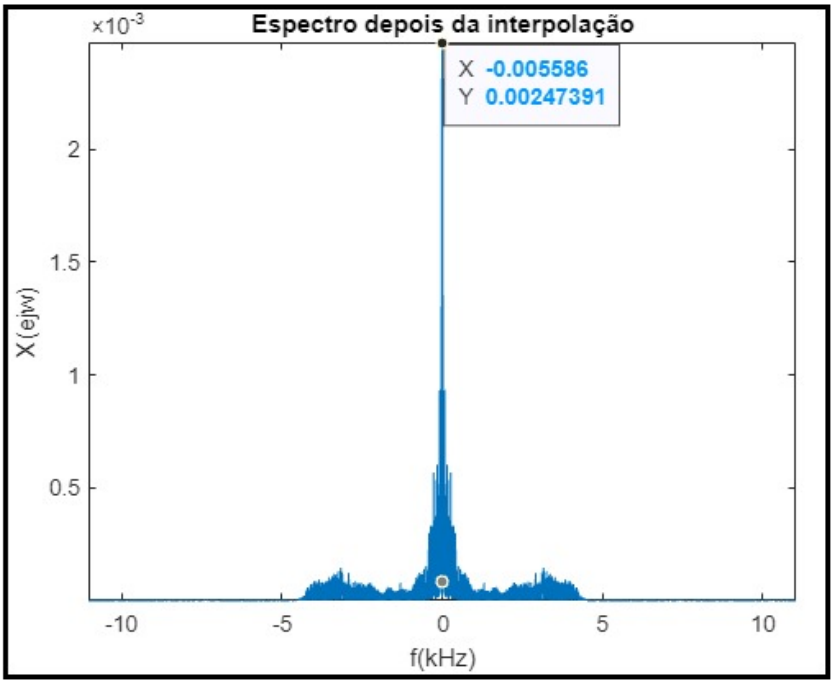



Espectro depois da dizimação
×10-3
7
6
5
4
3
2
1
X(ejw)
-10
-5
0
5
10
f(kHz)

- Para interpolação:

Figura 2




Espectro depois da interpolação
×10-3
X -0.005586
Y 0.00247391
2
1.5
1
0.5
X(ejw)
-10
-5
0
5
10
f(kHz)

2.3 Algoritmo da dizimação a partir de uma taxa de compressão M (Questão 1.5)

Na sequência, implementou-se um algoritmo para receber do usuário um sinal e uma taxa de compressão M. A partir disso, fez-se a dizimação e o resultado foi armazenado em um arquivo com as informações da frequência de amostragem original f_s e da taxa de compressão M.

```
[x, fs] = audioread('EuGostoDeDSP.wav');
comprimir_sinal(x, 4, fs);

function comprimir_sinal(sinal, M, fsamp)
    filename = 'comprimido.csv';

    wp = fsamp/(2*M);
    wr = (5*fsamp)/(2*4*M);

    [filtrado, window, N, beta, fil_coefs] = filtroKaiser(sinal, fsamp, [wp, wr]);

    % Dizima  o do sinal
    dizimado = sinal(1:M:end);

    % Cabe alho com taxa de amostragem e taxa de compress o
    cabecalho = [fsamp; M];

    % Concatena o cabe alho com o sinal dizimado
    resultado = [cabecalho; dizimado];

    % Salva o resultado no arquivo CSV
    csvwrite(filename, resultado);
end

function [filtrado, window, N, beta, fil_coefs] = filtroKaiser(sinal, fsamp, fcuts)
    mags = [1 0];
    devs = [0.01 0.01];

    [N,Wn,beta,FILTYPE] = kaiserord(fcuts, mags, devs, fsamp);
    window = kaiser(N+1, beta);

    fil_coefs = fir1(N, Wn, window);
    filtrado = filter(fil_coefs, 1, sinal);
end
```

Listing 3: Algoritmo da dizimação

Abaixo segue um descrição detalhada das partes que compõem o código:

- **x** e **fs** recebem o sinal de áudio e a taxa de amostragem do arquivo de áudio "EuGostoDeDSP.wav", respectivamente.
- A função comprimir-sinal é chamada com os argumentos **x** (sinal de áudio), **4** (fator de compressão M), e **fs** (taxa de amostragem).
- **filename** é definido como o nome do arquivo CSV onde os resultados serão armazenados.
- **wp** e **wr** são calculados com base na taxa de amostragem original (**fsamp**) e no fator de compressão (**M**).
- **filtroKaiser** é chamada para projetar um filtro FIR passa-baixas com base nos parâmetros calculados.
- O sinal é dizimado usando a taxa de compressão **M**.
- **cabecalho** é criado como um vetor que contém a taxa de amostragem e a taxa de compressão.
- **resultado** é criado concatenando o **cabeçalho** com o sinal dizimado.

- Os resultados são salvos no arquivo CSV definido por **filename**. **filtroKaiser** é uma função auxiliar que projeta um filtro passa-baixas usando o método da janela de Kaiser.
- **mags** e **devs** são especificados para definir as características desejadas do filtro. **kaiserord** é usado para calcular a ordem **N**, a frequência de corte normalizada **Wn**, o parâmetro **beta**, e o tipo de filtro. **kaiser** é utilizado para gerar a janela de Kaiser.
- **fir1** projeta os coeficientes do filtro FIR usando a janela de Kaiser gerada. **filter** aplica o filtro FIR ao sinal de entrada.

O código finaliza com os resultados sendo salvos no arquivo CSV "comprimido.csv", contendo informações sobre a taxa de amostragem original e a taxa de compressão, bem como o sinal dizimado após a aplicação do filtro passa-baixas.

2.4 Algoritmo da reconstrução do sinal (Questão 1.6)

Como solicitado no enunciado, o código reconstruir-comprimido realiza a reconstrução de um sinal a partir de um arquivo comprimido no formato CSV.

```
reconstruir_comprimido('comprimido.csv');

function reconstruir_comprimido(filename)

    data = csvread(filename)

    fsamp = data(1);
    L = data(2);

    sinal = [];

    for i = 3:length(data)
        sinal(i-2) = data(i);
    end

    inter = [];
    aux = 1;

    for n = 1:(L*length(sinal))
        if(aux > L)
            aux = 1;
        end

        if(aux == 1)
            if(1+round(n/L) < length(sinal))
                inter(n) = sinal(1+round(n/L));
            end
        else
            inter(n) = 0;
        end
        aux = aux + 1;
    end

    wp = fsamp/(2*L);
    wr = (5*fsamp)/(2*4*L);

    [interpolado, window, N, beta, fil_coefs] = filtroKaiser(inter, fsamp, [wp, wr]);

    figure(1);
    plotis_tempo(interpolado, fsamp);
    ylabel ('x(t)');
    title('x(t)')
```

```
    axis tight;

    figure(2);
    plotis_freq(interpolado,fsamp)
    ylabel ('X(ejw)');
    title('X(ejw)')
    axis tight;

    sound(interpolado, fsamp);

end

function [filtrado, window, N, beta, fil_coefs] = filtroKaiser(sinal, fsamp, fcuts)
    mags = [1 0];
    devs = [0.01 0.01];

    [N,Wn,beta,FILTYPE] = kaiserord(fcuts,mags,devs,fsamp)
    window = kaiser(N+1, beta);

    fil_coefs = fir1(N,Wn,window);
    filtrado = filter(fil_coefs,1,sinal);
end

function plotis_tempo(sinal, fsamp)
    time = [];
    Ts = 1/fsamp;
    for i = 1:length(sinal)
        time(i) = i*Ts;
    end
    plot(time,sinal);
    xlabel ('t(s)');
end

function plotis_freq(sinal,fs)
    N = length(sinal);
    f = [];
    f(1) = -fs/2;
    Xw = fftshift(abs(1/length(sinal)*fft(sinal)));
    for i = 2:length(sinal)
        f(i) = f(i-1)+(fs*1/N);
    end
    f = f/1000;
    plot(f,Xw)
    xlabel ('f(kHz)');
end
```

Listing 4: Algoritmo da dizimação

Abaixo são feitas ponderações acerca da implementação do código:

Leitura dos dados: data = csvread(filename): O código lê os dados do arquivo CSV especificado pelo usuário e armazena-os na matriz data. Cada linha do arquivo representa um vetor coluna em data.

Extração de Parâmetros: fsamp = data(1); L = data(2): Extrai a taxa de amostragem (fsamp) e o fator de interpolação (L) do cabeçalho do arquivo. Essas informações são armazenadas nas primeiras duas linhas da matriz data.

Reconstrução do Sinal: sinal = []; : Inicializa um vetor para armazenar os dados do sinal reconstruído. for i = 3:length(data) sinal(i-2) = data(i); end: O loop extrai os dados do sinal armazenados a partir da terceira linha da matriz data e os coloca no vetor sinal

Dizimação Inversa: inter = [ ]; aux = 1; for n = 1:(L*length(sinal)): Realiza a dizimação inversa

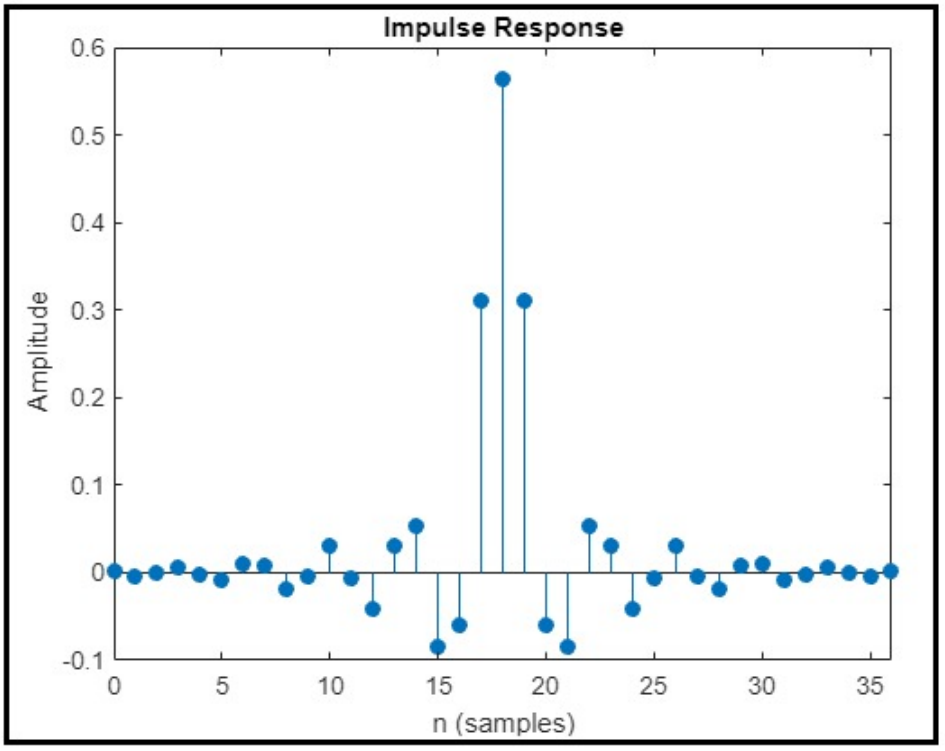
para expandir o sinal comprimido. O vetor inter armazena os valores do sinal interpolado.

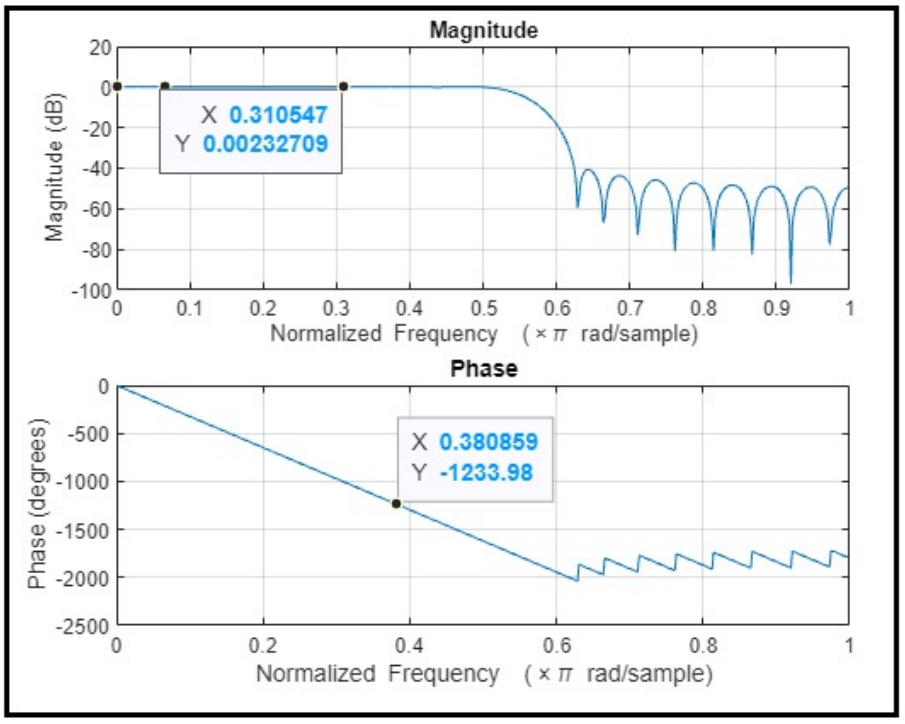
Filtro de Kaiser e Reconstrução Final: wp = fsamp/(2*L); wr = (5*fsamp)/(2*4*L) : Calcula as frequências de borda para o filtro de Kaiser. [interpolado, window, N, beta, fil-coefs] = filtroKaiser(inter, fsamp, [wp, wr]);: Aplica o filtro de Kaiser ao sinal interpolado, resultando no sinal final reconstruído, interpolado.

Visualização e Reprodução do Sinal Reconstruído: figure(1); plotis-tempo(interpolado, fsamp) : Plota o sinal reconstruído no domínio do tempo. figure(2); plotis-freq(interpolado, fsamp) : Plota o espectro de frequência do sinal reconstruído. sound(interpolado, fsamp): Reproduz o sinal reconstruído.

Função filtroKaiser: Uma função auxiliar que implementa um filtro passa-baixas de Kaiser para suavizar o sinal.

Em suma, o código realiza a reconstrução de um sinal a partir de um arquivo comprimido, aplicando a dizimação inversa, filtro de *Kaiser* e interpolação, antes de visualizar e reproduzir o sinal reconstruído.





2.5 Análises Finais (Questão 1.7)

A compressão amplia o espectro, no entanto, o limite é fs/2. Consequentemente, as frequências que são deslocadas para além de fs/2 durante o processo de compressão são eliminadas, resultando na perda das altas frequências e conferindo um caráter abafado ao som. A eficiência computacional é satisfatória, uma vez que os algoritmos executam rapidamente em computadores comuns de uso geral.

2.6 Bônus: modificação da velocidade de reprodução sem alteração de tons.

Nesta etapa bônus, foi tentado implementar um processo de compressão ou descompressão (aceleração ou desaceleração) de um sinal de áudio carregado a partir do arquivo 'EuGostoDeDSP.wav'. Assim, foi possível escolher o tipo de operação por meio da variável tipo, que pode ser 'acelerar' ou 'desacelerar', e também especificar a taxa de compressão ou descompressão através da variável taxa.

A operação de aceleração (tipo = 'acelerar') envolvia a aplicação da função dizimarSinal, que realiza a dizimação do sinal por um fator taxa. Durante esse processo, é utilizado um filtro Kaiser para evitar o aliasing, e o resultado é armazenado em dizimado. Em seguida, a função calc-format foi chamada para calcular os coeficientes do filtro que serão utilizados na interpolação do sinal comprimido.

A operação de desaceleração (tipo = 'desacelerar') não está completamente implementada no código implementado, pois a variável L não foi definida. O trecho de código que realiza a interpolação (interpolarSinal) ainda precisa ser implementado para a desaceleração.

Ao final do código, há funções auxiliares que implementam as operações de filtragem (filtroKaiser), visualização temporal (plotis-tempo), visualização em frequência (plotis-freq), cálculo do espectro (espectro), e cálculo dos coeficientes do filtro para ajuste de formato (calc-format).

Quanto à observação de que o resultado obtido para o áudio não saiu tão bem como era esperado, pode-se presumir que existem desafios na implementação, especialmente no tratamento adequado do espectro e na manipulação dos sinais para garantir a qualidade do áudio resultante. Além disso, a operação de desaceleração ainda não está completamente implementada. Isso pode explicar a percepção de que o resultado não atendeu às expectativas.

```
1 [x, fs] = audioread('EuGostoDeDSP.wav');
```

```
sound(x, fs);
taxa = 2;
tipo = 'acelerar';

%tipo = 'desacelerar';
%{
X = fft(x);
%Xw = fftshift(abs(1/length(sinal)*fft(sinal)));
Xw = fftshift(abs(1/length(x)*X));

Xz = ifftshift(Xw);

x2 = ifft(Xz*length(x));

t = (0:length(X)-1)*(1/fs);
figure(40);
plot(t,x2)
sound(x2, fs);

return
%}

%Espectro original
figure(10);
plotis_tempo(x, fs)
[f_ori, Xw_ori] = espectro(x,fs);
figure(12);
plotis_freq(x, fs);
axis tight;

if(tipo == 'acelerar')
    [dizimado, window, N, beta, fil_coefs] = dizimarSinal(x, taxa, fs);
    figure(14);
    plotis_freq(dizimado, fs);
    axis tight;
    figure(11);
    plotis_tempo(dizimado, fs)
    [f_diz, Xw_diz] = espectro(dizimado,fs);
    [b,a] = calc_format(Xw_ori, f_ori, Xw_diz, f_diz, fs);
    %convoluir sinal comprimido com a fun   o de transferencia definida por [b,a]

    %formato = tf(b,a);
    y = conv(x, tf, 'same');
    %[y,t]=lsim(tf,dizimado);
    %plot(t,y);

elseif(tipo == 'desacelerar')
    [interpolado, window, N, beta, fil_coefs] = interpolarSinal(dizimado, L, fs);
end



%-----------


function [dizimado, window, N, beta, fil_coefs] = dizimarSinal(sinal, M, fsamp)

    wp = fsamp/(2*M);
    wr = (5*fsamp)/(2*4*M);

    [filtrado, window, N, beta, fil_coefs] = filtroKaiser(sinal, fsamp, [wp, wr]);

    dizimado = [];
```

```
66
67      for n = 1:(length(sinal)/M)
68          dizimado(n) = sinal(n*M);
69      end
70
71  end
72
73  function [interpolado, window, N, beta, fil_coefs] = interpolarSinal(sinal, L, fsamp)
74
75      inter = [];
76      aux = 1;
77
78      for n = 1:(L*length(sinal))
79          if(aux > L)
80              aux = 1;
81          end
82
83          if(aux == 1)
84              if(1+round(n/L) < length(sinal))
85                  inter(n) = sinal(1+round(n/L));
86              end
87          else
88              inter(n) = 0;
89          end
90          aux = aux + 1;
91      end
92
93      wp = fsamp/(2*L);
94      wr = (5*fsamp)/(2*4*L);
95
96      [interpolado, window, N, beta, fil_coefs] = filtroKaiser(inter, fsamp, [wp, wr]);
97
98  end
99
100
101 function [filtrado, window, N, beta, fil_coefs] = filtroKaiser(sinal, fsamp, fcuts)
102     mags = [1 0];
103     devs = [0.01 0.01];
104
105     [N,Wn,beta,FILTYPE] = kaiserord(fcuts,mags,devs,fsamp)
106     window = kaiser(N+1, beta);
107
108     fil_coefs = fir1(N,Wn,window);
109     filtrado = filter(fil_coefs,1,sinal);
110 end
111
112 function plotis_tempo(sinal, fsamp)
113     time = [];
114     Ts = 1/fsamp;
115     for i = 1:length(sinal)
116         time(i) = i*Ts;
117     end
118     plot(time,sinal);
119     xlabel ('t(s)');
120 end
121
122 function [f, Xw] = espectro(sinal,fs)
123     N = length(sinal);
124     f = [];
125     f(1) = -fs/2;
126     %Xw = fftshift(1/length(sinal)*fft(sinal));
127     Xw = fft(sinal);
128     for i = 2:length(sinal)
129         f(i) = f(i-1)+(fs*1/N);
```

```
    end
    f = f/1000;
    %plot(f,Xw)
    %xlabel ('f(kHz)');
end

function [b,a] = calc_format(espec_original, freqs_original, espec_alterado,
    freqs_alterado, fs)

    %   necess rio alterar espec_alterado para que ele possua as mesmas
    %frequencias de espec_original
    %Ou pegar valores interpolados de espec_original para as frequencias de
    %espec_alterado

    h = [];
    for i = 1:length(espec_alterado)
        A = interp1(freqs_original,espec_original,freqs_alterado(i),'nearest');
        h(i) = A/espec_alterado(i);
    end

    %figure(3);
    %plotis_freq(h,fs);
    %axis tight;

    figure(17);
    %plotis_n(h);
    axis tight;

    rec = h.*espec_alterado;
    figure(15);
    %plotis_n(rec);
    axis tight;

    %deshifted = 5*length(espec_alterado)*ifftshift(espec_alterado);

    %X = ifft(deshifted);
    rec1 = fftshift(rec);
    X = ifft(rec1);

    t = (0:length(X)-1)*(1/fs);
    figure(40);
    %plot(t,X)
    audiowrite('out.wav',abs(X),fs)
    sound(abs(X), fs);

    w = []
    w(1) = 0;
    N = length(h)

    for i = 2:N
        w(i) = w(i-1)+(pi*1/N);
    end

    %figure(4);
    %plot(w,h);
    %axis tight;

    [b,a] = invfreqz(h,w,10,10);
    %figure(20);
    %freqz(b,a);

end

function plotis_n(sinal)
```

```
    n = [];
    for i = 1:length(sinal)
        n(i) = i;
    end
    plot(n, sinal);
    xlabel ('n');
end


function plotis_freq(sinal,fs)
    N = length(sinal)
    f = [];
    f(1) = -fs/2;
    Xw = fftshift(abs(1/length(sinal)*fft(sinal)));
    for i = 2:length(sinal)
        f(i) = f(i-1)+(fs*1/N);
    end
    f = f/1000;
    plot(f,Xw)
    xlabel ('f(kHz)');
end
```

Listing 5: Algoritmo da dizimação

Uma alternativa ao problema de implementação seria usar uma função já disponível no *MATLAB* chamada de *stretchAudio( )*. Não foi possível implementar a função no código em tempo hábil para entrega do trabalho.

3 Referências

[1] - OPPENHEIM, A. V.; WILLSKY, A. S.; NAWAB, Syed H. Sinais e sistemas. 2. ed.