Processamento Digital de Sinal – 2019/2020

Trabalho Prático n.3

Interpolação fracionária com implementação eficiente. Grupo 8

Pedro Ribeiro
2013136821

Pedro Silva
2011149228

1 Introdução

Neste trabalho laboratorial, pretendemos fazer a conversão da frequência de amostragem de 16000 Hz para 44100 Hz usando uma implementação mais eficiente comparativamente a métodos implementados e estudados no decorrer da unidade curricular. Pretendemos fazer esta interpolação de acordo com a equação:

$$y[n] = \sum_{k=0, nM-k=mL}^{N_h-1} h[k]x[\frac{nM-k}{L}] \tag{1}$$

Onde L e M são os factores de decimação e expansão, respectivamente, e N_h o comprimento do filtro FIR a implementar.

2 Interpolação fracionária com implementação ineficiente

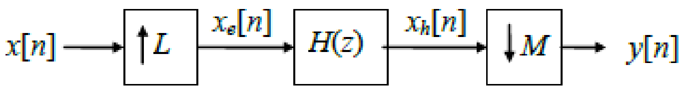

Figure 1: Implementação ineficiente

Começamos por fazer a interpolação fracionária préviamente estudada, representada em Implementação ineficiente. Começamos por descobrir os factores de expansão e decimação, que devem ser primos entre si. Estes factores são obtíveis através da factorização da frequência de amostragem final, $f_{s,out}$ = 44100Hz, e da frequência de amostragem inicial, $f_{s,in}$ = 16000Hz. Em Matlab, utilizámos a função nativa `rat`.

```
fs_in = 16000;
fs_out = 44100;
[L, M] = rat(44100/16000);
```

Obtemos assim os valores 441 e 160 para os factores de expansão e de decimação, respectivamente.

$$\frac{L}{M} = \frac{f_{s,out}}{f_{s,in}} = \frac{44100}{16000} = \frac{441}{160}$$

Seguidamente, utilizámos a função nativa de Matlab `fir1()`, que por defeito utiliza uma janela de Hamming para o desenho do filtro, para criar um filtro FIR de comprimento 3529 e ordem 3528 com uma frequência de corte $\omega_c = \frac{\pi}{max(L,M)} = \frac{\pi}{L}$. Sabendo que a energia do sinal filtrado diminui por um factor L, aplicámos um ganho $G = L$ ao filtro por forma a conservarmos a energia do sinal.

Aplicamos agora este sistema multiritmo a um segundo do sinal de entrada, `pcmtext.wav`, expandido-o pelo factor de expansão L, filtrando-o pelo filtro de interpolação H(z) representado em Ganho do filtro, e decimando-o pelo factor de decimação M, obtemos os resultados em Um segundo do sinal de entrada, interpolado e com diferentes frequências de amostragem. A inefeciência deste método, reside no facto de, uma vez que o sinal será decimado, realizámos cálculos sob amostras que não foram utilizadas.

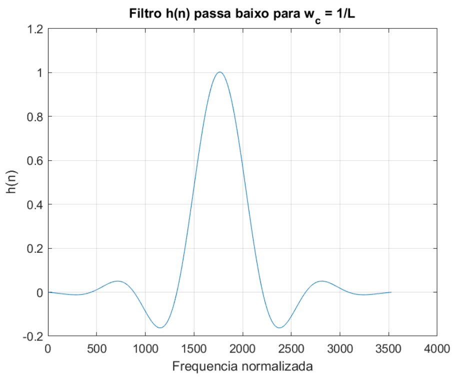

Figure 2: Filtro h[n]

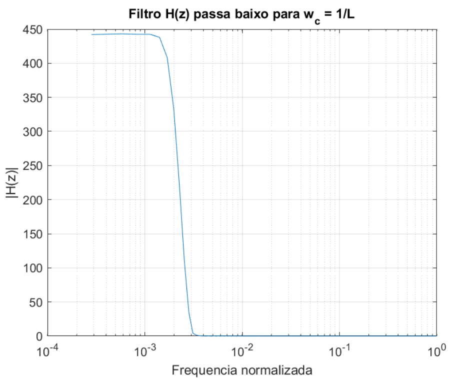

Figure 3: Ganho do filtro

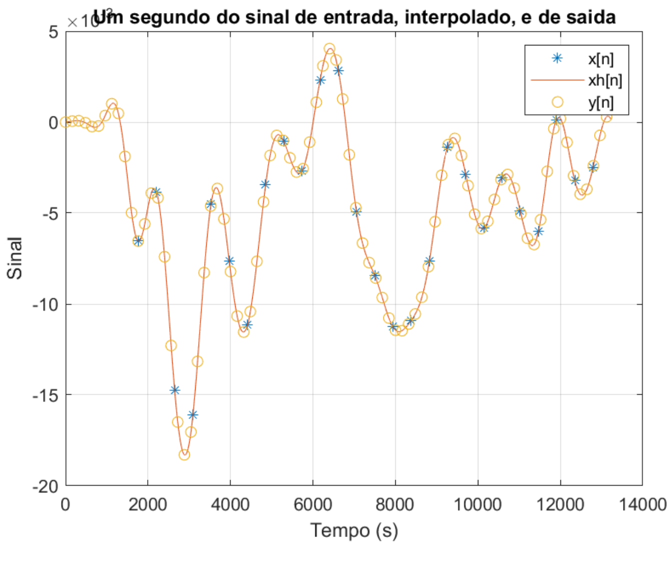

Figure 4: Um segundo do sinal de entrada, interpolado e com diferentes frequências de amostragem

3 Interpolação fracionária com implementação eficiente

Faremos agora uma implementação mais eficiente para a interpolação da frequência de amostragem, calculando apenas as amostras que serão utilizadas, utilizando a equação descrita na secção Introdução.

$$y[n] = \sum_{k=0, nM-k=mL}^{N_h-1} h[k]x[\frac{nM-k}{L}] \tag{2}$$

Esta soma foi implementada no seguinte bloco de código:

```
y2 = zeros(44100, 1);           %Alocar mem ria para o matlab
for n = 0:44100-1        %calcular 44100 sinais de saida
    nM = n * M;
    k = mod(nM, L);
    m = floor((nM - k) / L);
    %k = nM - m * L;
    sum = 0;             %inicializar somatorio
    while k < Nh && m >= 0      %condicoes do somatorio
        sum = sum + (h(k+1) * x(m+1));
        k = k + L;
        m = m - 1;
    end
    %Este ciclo while corre (Nh-1 / L) vezes. Neste caso, 8 vezes
    y2(n+1) = sum;
end
```

Comparando os dois sinais resultantes de cada implementação, temos a figura Comparação das implementações. Observamos que os dois sinais resultantes apresentam um desvio muito ligeiro, na ordem de 10^{-16}. Com esta implementação e estudando o bloco de código acima disposto, observamos que para cada amostra de saída, são necessárias 8 iterações do ciclo while, pois para uma n-ésima amostra, k toma o valor inicial do resto da divisão $(n * M)/L$ e se incrementa de L em L. Consequentemente, o ciclo while se itera $\frac{N_h-1}{L}$ vezes. Neste ciclo while são executadas 4 instruções de soma/multiplicação, e uma iteração do ciclo for, onde se executam 3 instruções de multiplicação/execução, totalizando em 35 instruções de soma/multiplicação.

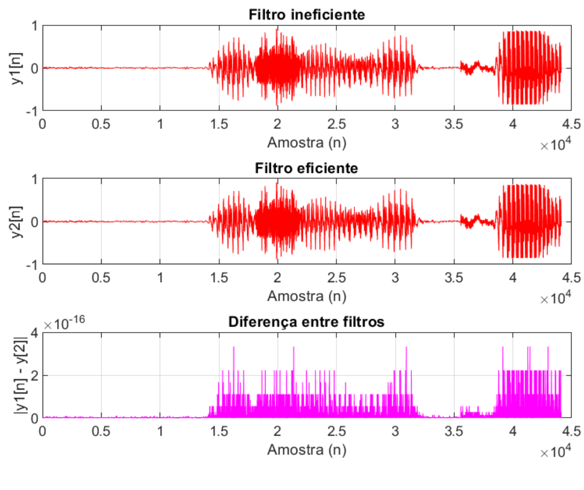

Figure 5: Comparação das implementações

4 Interpolação fracionária com ferramentas nativas a Matlab

Vamos agora utilizar as ferramentas nativas ao Matlab para a interpolação fracionária, a função `upfirdn()`. Estudando a documentação desta função observamos que tem a seguinte sintaxe:

> yout = upfirdn(xin,h) filters the input signal xin with an FIR filter with impulse response h. No upsampling or downsampling is implemented with this syntax.
>
> yout = upfirdn(xin,h,p,q) specifies the integer downsampling factor q.

Ou seja, para um sinal de entrada, x_{in}, filtramo-lo com um filtro FIR com um resposta a impulso h para um factor de expansão e decimação p e q, respectivamente. Após o uso desta ferramenta, comparamos o desvio entre esta implementação e a implementação eficiente por nós desenvolvida e verificamos que não existe qualquer desvio entre as amostras de saída das duas implementações: implementámos este método exactamente como o Matlab a implementa.

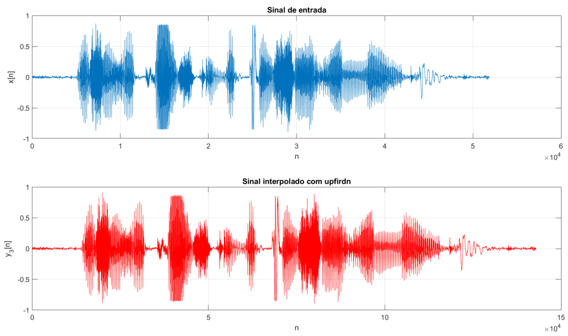

Figure 6: Comparação entre o sinal original e interpolado por upfirdn()

Observando a figura Comparação entre o sinal original e interpolado por upfirdn(), que demonstra o sinal original e o sinal após interpolação com o método `upfirdn()`, observamos que o sinal interpolado é o sinal original com o eixo das abcissas expandido: o sinal foi "esticado".

Por audição dos dois sinais às respectivas frequências, o sinal interpolado parece ter filtrado os sons mais fricativos da fala. No entanto, só poderemos tomar conclusões mais definitivas com um estudo do sonograma dos dois sinais.

5 Código Matlab

```
%Lab 3 PDS: Conversao de Frequencia de amostragem
%PL2
%Grupo 8
%Autores:
%Pedro Ribeiro
%Pedro Silva
close all; clc; clear ; mkdir('Imagens ');delete Imagens/*.*
%a) Considere um sinal de fala com frequ ncia de amostragem de fs=16 kHz, por exemplo
%'pcmtest.wav'. Identifique os fatores de expans o, L, e de decima  o, M.
[x,fs_in] = audioread('pcmtest.wav'); % l  sinal e freq. de amostragem.
fs_out = 44100;     %frequencia de entrada    16kHz, e queremos que a frequencia de saida seja 44100Hz.
%[L,M] = numden(sym(fs_out/fs_in);
[L, M] = rat(44100/16000);            %https://www.mathworks.com/help/releases/R2019a/matlab/ref/rats.html?container=jshelpbrowser
%^-- descobrir os factores de expans o e de decima  o (L, M).
%L > M, porque queremos aumentar a frequencia do sinal
% b) Projete um filtro FIR, passa-baixo, com comprimento Nh=3529 (ordem 3528) com o m todo das
% janelas (comando fir1). Aplique um ganho L para que as amplitudes de entrada e de sa da sejam
% iguais.
Nh = 3529;      %Comprimento do filtro FIR
h = fir1(Nh-1, 1/L) * L;        %nao esquecer compensar o ganho do filtro
[H,w] = freqz(h, 1, Nh);      %resposta em frequencia do filtro
fig1 = figure(1);
semilogx(w/pi, abs(H))
grid on; title('Filtro H(z) passa baixo para w_c = 1/L'); xlabel('Frequencia normalizada'); ylabel('|H(z)|')
print(fig1, '-dpng', 'Imagens/ganho_H_z')
fig2 = figure(2);
plot(h)
grid on; title('Filtro h(n) passa baixo para w_c = 1/L');
print(fig2, '-dpng', 'Imagens/filtro_h_n')
% c) Fa a a implementa  o direta do sistema para 16000 amostras (1 segundo do sinal de entrada,
% mono), em analogia com o trabalho pr tico anterior. Guarde estas amostras na vari vel y1 (para
% conferir depois o resultado com o m todo eficiente). Confirme que guardou 44100 amostras de
% sa da, isto  , 1 segundo de sinal de sa da.
x_in = x(1:16000);          %um segundo do sinal de entrada
xe = upsample(x_in, L);     %Sinal expandido
xh = filter(h, 1, xe);          %sinal interpolado (saida do filtro H(z)
y1 = xh(1:M:end);           %sinal decimado. Saida do filtro decimada por M
%gr fico de Nx1 amostras de entrada e Ny1 amostras de sa da:
Nx1=30; Nh1=Nx1*L; Ny1=ceil(Nh1/M);
fig3 = figure(3);
plot(4*L+1:L:Nh1,x(1:Nx1-4),'*',1:Nh1,xh(1:Nh1),1:M:Ny1*M,y1(1:Ny1),'o')
grid on; ; title('Um segundo do sinal de entrada, interpolado, e de saida'); xlabel('Tempo (s)'); ylabel('Sinal'); legend({'x[n]','xh[n]','y[n]'})
print(fig3, '-dpng', 'Imagens/sinais_entrada_interpolado_saida')
% d) Fa a agora uma implementa  o eficiente, calculando a sa da no vetor y2. (aloque espa o
% inicialmente para y2). Fa a um ciclo para calcular cada valor de y[n], tomando como primeiro
%  ndice  k o resto da divis o de nM por L. Pare o ciclo se k ultrapassar o valor Nh?1 ou se o  ndice
```

```
% do sinal de entrada for negativo.
% Sugest o: use  ndices  a come ar em zero e indexe as vari veis do Matlab com avan o de 1, por
% exemplo, h(k+1) para se referir a h[k].
% No final, compare as amostras de y2 com y1. Nota: fa a um gr fico do erro relativo a y1; (os
% valores devem conferir em 12 ou mais algarismos significativos). Verifique primeiro se as
% dimens es de y1 e de y2 coincidem.

y2 = zeros(44100, 1);          %Alocar mem ria para o matlab
for n = 0:44100-1       %calcular 44100 sinais de saida
    nM = n * M;
    k = mod(nM, L);
    m = floor((nM - k) / L);
    %k = nM - m * L;
    sum = 0;            %inicializar somatorio
    while k < Nh && m >= 0      %condicoes do somatorio
        sum = sum + (h(k+1) * x(m+1));
        k = k + L;
        m = m - 1;
    end
    %Este ciclo while corre (Nh-1 / L) vezes. Neste caso, 8 vezes
    y2(n+1) = sum;
end
fig4 = figure(4);
subplot(3,1,1)
plot(1:44100, y1, 'r')
grid on; title('Filtro ineficiente'); xlabel('Amostra (n)'); ylabel('y1[n]');
subplot(3,1,2)
plot(1:44100, y2, 'r')
grid on; title('Filtro eficiente'); xlabel('Amostra (n)'); ylabel('y2[n]');
subplot(3,1,3)
erro = abs(y1-y2);
plot(1:44100, erro, 'm')
grid on; title('Diferen a entre filtros'); xlabel('Amostra (n)'); ylabel('|y1[n] - y[2]|');
print(fig4, '-dpng', 'Imagens/comparacao_filtros')
% f) Repita o c lculo usando agora o comando upfirdn(), calculando a sa da na vari vel y3. Compare
% as primeiras 44100 amostras da sa da y3 com y2: devem ser exatamente iguais. Porqu ?
% Use: max(abs(y2-y3(1:44100)))
y3 = upfirdn(x, h, L, M);
max(abs(y2-y3(1:44100)))
% h) Usando o algoritmo eficiente (ou upfirdn()) calcule a sa da relativa ao sinal x[n] completo.
% Depois ou a os dois sinais  s  frequ ncias respetivas. Nota alguma diferen a?
fig5 = figure(5)
subplot(2,1,1)
plot(x)
grid on; title('Sinal de entrada'); xlabel('n'); ylabel('x[n]')
subplot(2,1,2)
plot(y3,'r')
grid on; title('Sinal interpolado com upfirdn'); xlabel('n'); ylabel('y_3[n]')
print(fig5, '-dpng', 'Imagens/comparacao_entrada_upfirdn')
disp('Sinal de entrada')
sound(x, 16000)
disp('Sinal interpolado eficientemente')
pause(3)
sound(y3, 44100)
```