FILTRAGEM DE SINAL DE ÁUDIO COM A UTILIZAÇÃO DO ALGORITMO OVERLAP-SAVE, TRANSFORMADA RÁPIDA DE FOURIER E FIR KAISER WINDOW

Santa Maria, 11 de dezembro de 2018

Keli Tauana Ruppenthal [1]
Moisés Goulart de Oliveira [2]
Victor Dallagnol Bento [3]
Yuri Oliveira Alves [4]

Resumo: Este trabalho apresenta a construção e os resultados de um filtro implementado em *Python* para a disciplina de Processamento Digital de Sinais, ministrada pelo Professor Marcos Hideo Maruo, do 2º semestre de 2018, para o curso de Engenharia de Computação, como parte da avaliação da disciplina.

Palavras - chave: *Filtro, sinal, overlap-save, fft, faixa.*

1. Introdução

As Transformadas rápidas de Fourier são de grande importância para uma vasta gama de aplicações, como de Processamento digital de sinais para a resolução de equações diferenciais parciais a algoritmos para multiplicação de grandes inteiros. Transformadas rápidas de Fourier são amplamente utilizadas para aplicações diversas em engenharia, ciência e matemática. Este tipo de transformada é um algoritmo eficiente para se calcular a Transformada discreta de Fourier (DFT) e a sua inversa. A análise de Fourier converte um sinal do seu domínio original para uma representação no domínio da frequência e vice-versa. Uma Transformada rápida de Fourier calcula rapidamente essas transformações, fatorando a matriz da Transformada discreta de Fourier em um produto de fatores esparsos.

Ademais, também foi utilizada a Kaiser Window, que pode ser definida como uma família de parâmetros de funções de janela usada no projeto de filtro de resposta de impulso finito e análise espectral. A janela do Kaiser aproxima a janela do *DPSS*, que maximiza a concentração de energia no lóbulo principal, mas que é difícil de calcular.

[1] Acadêmico (a) do curso de Engenharia de Computação da Universidade Federal De Santa Maria- UFSM, matrícula: 201520603 , e-mail: kelitauana@gmail.com
[2] Acadêmico (a) do curso de Engenharia de Computação da Universidade Federal De Santa Maria- UFSM, matrícula: 2015510256, e-mail: moises.oliveira@ecomp.ufsm.br
[3] Acadêmico (a) do curso de Engenharia de Computação da Universidade Federal De Santa Maria- UFSM, matrícula: 201520835, e-mail: victor.bento@ecomp.ufsm.br
[4] Acadêmico (a) do curso de Engenharia de Computação da Universidade Federal De Santa Maria- UFSM, matrícula: 201521755, e-mail: yuri.alves@ecomp.ufsm.br

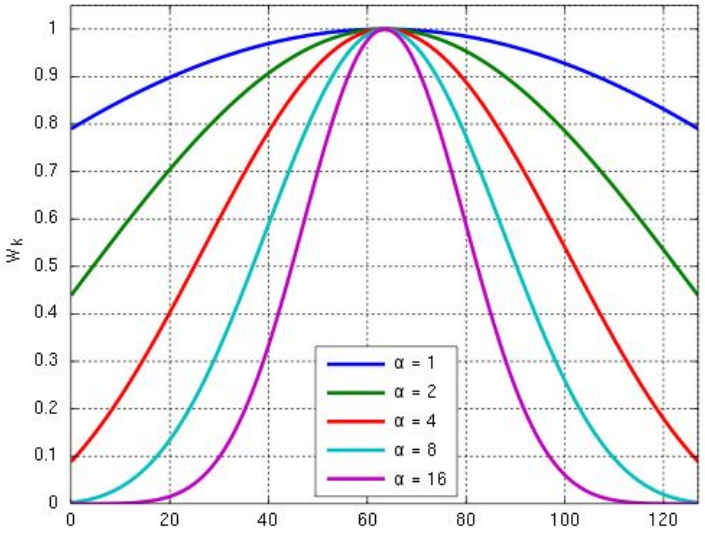


Figura 1: Janela Kaiser para vários valores de α.

Ainda, outra variável importante para a execução do trabalho é o algoritmo **Overlap – save**, o qual é definido como uma maneira eficiente de avaliar a convolução discreta entre um sinal muito longo $x(n)$ e um filtro de resposta ao impulso finito (FIR) $h(n)$. Este método divide o sinal de entrada em parcelas de tamanho L + M – 1, sendo L o tamanho da parcela do sinal a ser tomada e M o tamanho do sinal h(n), pelo qual o sinal de entrada será convoluído. Entretanto, nesse caso são adicionadas à parcela tomada algumas amostras do segmento anterior. Estas amostras do sinal anterior são do tamanho M -1, de forma que, para um filtro de M = 4 e uma largura de janela escolhida L = 5, a cada iteração ocorrerá uma convolução de 3 amostras da parcela da iteração anterior e 5 amostras da parcela atual, formando uma “fatia” de 8 amostras.

Após a convolução, as primeiras M-1 amostras da saída, são eliminadas. Caso seja a primeira iteração do algoritmo, por não haver parcela anterior, as primeiras M-1 amostras da fatia final são preenchidas com zeros. As figuras abaixo ilustram o processo de segmentação do sinal de entrada e reconstrução do sinal de saída.

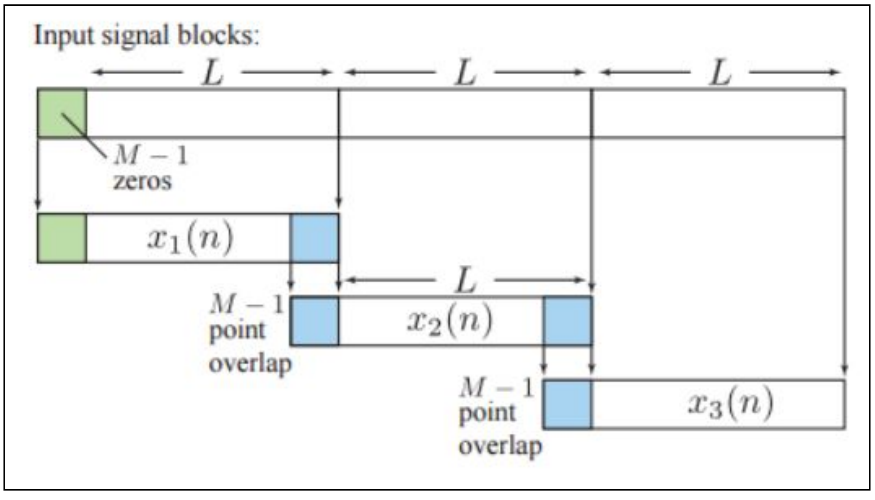


Figura 2: Segmentação do sinal de entrada na execução do overlap-save.

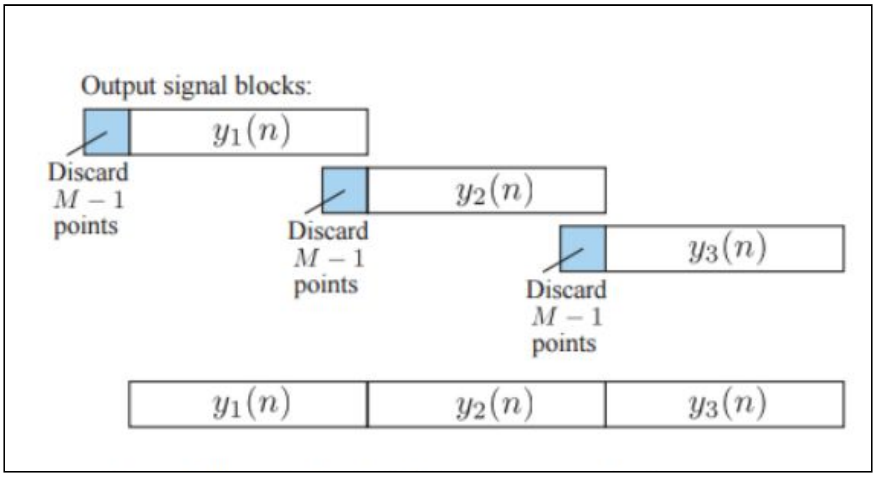


Figura 3: Reconstrução do sinal de saída na execução do overlap-save.

2. Metodologia

A linguagem utilizada para descrever o código de filtragem foi *Python*. Além disso, utilizaram-se diferentes plataformas para a elaboração do código, como *Colaboratory* (para escrita e simulação do código, além de verificação dos gráficos), *GitHub*, *LibROSA* e FIIIR! (para a confecção do filtro e configuração do mesmo). As principais bibliotecas utilizadas para o desenvolvimento do projeto foram:

- **numpy:** Gerar/Carregar áudio;
- **IPython.display:** Reprodução de áudio;
- **scipy.signal:** Auxiliar na utilização de filtros e funções;
- **matplotlib.pyplot:** Possibilitar o *plot* de gráficos.
- **librosa:** Auxiliar a análise do áudio.

Feito isso, deu-se início a escrita do código. Inicialmente, importou-se as bibliotecas necessárias.

```
import IPython.display as ipd
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sig
import librosa
import librosa.display
```

Então, é feita a carga do arquivo de áudio que encontra-se na pasta atual do projeto, utilizando o *LibROSA*, que é um pacote *python* para análise de música e áudio. Ele fornece os blocos de construção necessários para criar sistemas de recuperação/mineração de informações musicais.

```
x, sr = librosa.load('audio3.wav')
# x = sinal (signal)
# sr = amostra simples (simple rate)
```

A etapa seguinte é a definição do algoritmo overlap-save. Ele também é conhecido como algoritmo overlap-discard,e baseia-se em uma segmentação sobreposta da entrada ($x_L[k]$) e na aplicação da convolução periódica para os segmentos individuais.

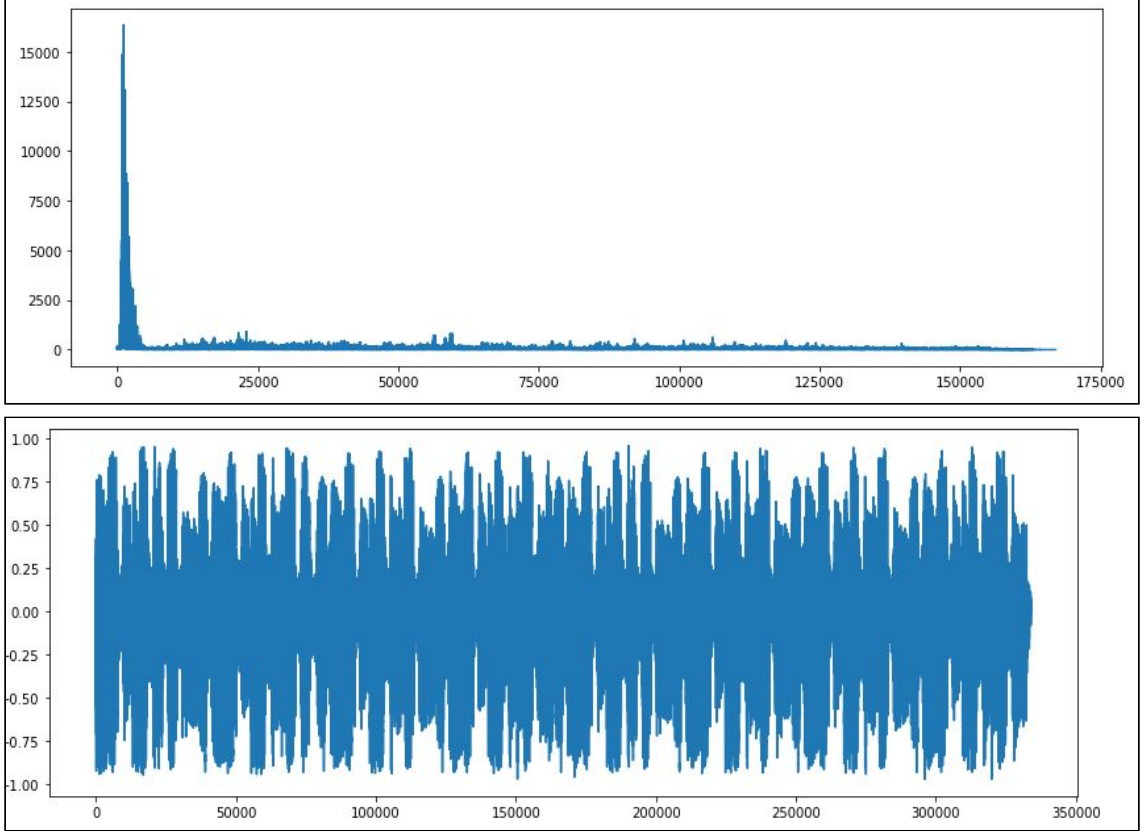
Olhando mais de perto para o resultado da convolução periódica $x_p[k] * h_n[k]$ onde $x_p[k]$ denota um segmento de comprimento P do sinal de entrada de $h_n[k]$. O resultado da convolução linear será do comprimento $(P+N-1)$. O resultado da convolução periódica do período P para $P > N$ sofreria um deslocamento circular (aliasing de tempo) e superposição das últimas amostras $(N-1)$ ao início. Assim, as primeiras amostras $(N-1)$ não são iguais ao resultado da convolução linear. No entanto, o restante $(P-N+1)$ faz isso.

Isso motiva a divisão do sinal de entrada ($x_L[k]$) em segmentos sobrepostos de comprimento P onde o segmento $p-th$ se sobrepõe ao segmento $(p-1)-th$ anterior por amostras $(N-1)$. Na imagem abaixo, verificamos a implementação feita pelo grupo.

```
def overlapsave (x, h):
    L = len(x)  # length of input signal
    N = len(h)  # length of impulse response
    P = len(h)+1 #P length of segments
    # overlap-save convolution
    nseg = (L+N-1)//(P-N+1) + 1
    x = np.concatenate((np.zeros(N-1), x, np.zeros(P)))
    xp = np.zeros((nseg, P))
    yp = np.zeros((nseg, P))
    y = np.zeros(nseg*(P-N+1))
    print(str(nseg) + ' blocos')
    for p in range(nseg):
        xp[p, :] = x[p*(P-N+1):p*(P-N+1)+P]
        yp[p, :] = np.fft.irfft(np.fft.rfft(xp[p, :]) * np.fft.rfft(h, P))
        y[p*(P-N+1):p*(P-N+1)+P-N+1] = yp[p, N-1:]
    y = y[0:N+L]
    return y
```

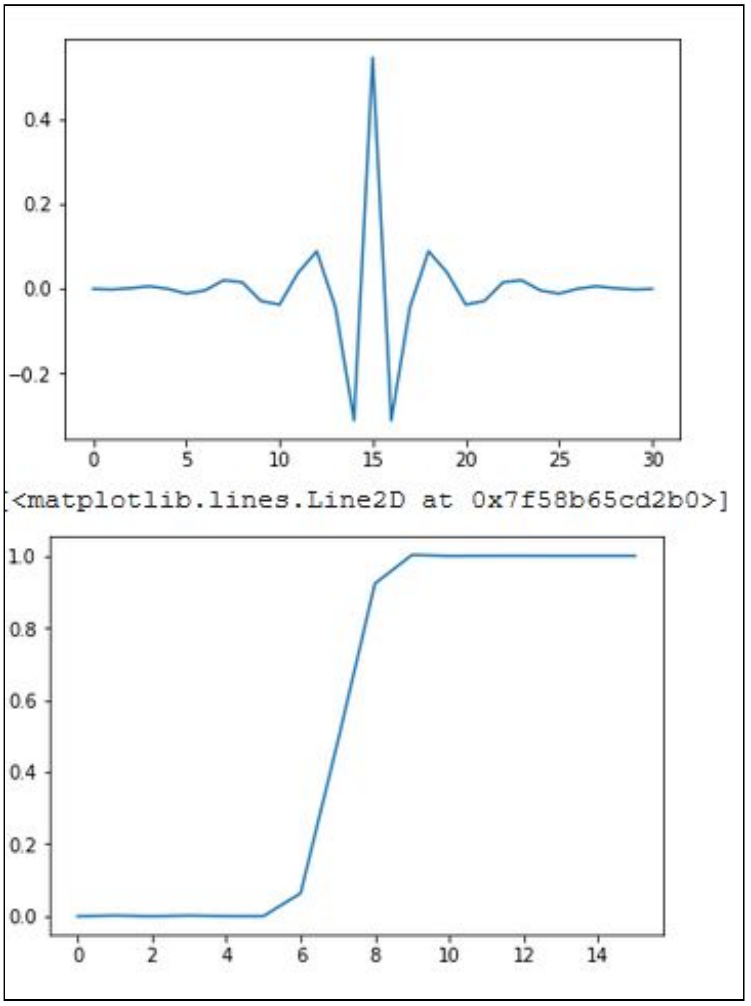
Em seguida, são plotados os gráficos da wave de entrada, onde a primeira figura é utilizando o valor absoluto da transformada discreta de Fourier.

```
# plota amostra do sinal de entrada
plt.figure(figsize=(14, 5))
plt.plot(abs(np.fft.rfft(x))) #aqui só é plotado da metade pro fim
plt.show()

plt.figure(figsize=(14, 5))
plt.plot((x)) #grafico da entrada
```



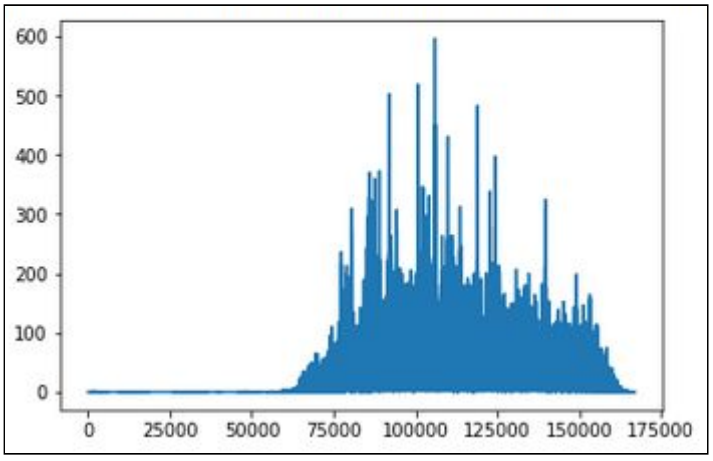
Feito isso, o filtro passa-altas, gerado no site FIIIR!, foi inserido ao código, e atribuído a variável h. Essa filtragem tem como objetivo deixar passar apenas os sinais graves do áudio. As imagens abaixo demonstram o filtro e a FFT do sinal.

```
# Filtro passa altas

# Configuration.
fS = 44100  # Sampling rate.
fH = 10000  # Cutoff frequency.
N = 31  # Filter length, must be odd.
beta = 4.534  # Kaiser window beta.

# Compute sinc filter.
h = np.sinc(2 * fH / fS * (np.arange(N) - (N - 1) / 2.))

# Apply window.
h *= np.kaiser(N, beta)

# Normalize to get unity gain.
h /= np.sum(h)

# Create a high-pass filter from the low-pass filter through spectral inversion.
h = -h
h[(N - 1) // 2] += 1


plt.plot(h)
plt.show()
plt.plot(abs(np.fft.rfft(h)))
```

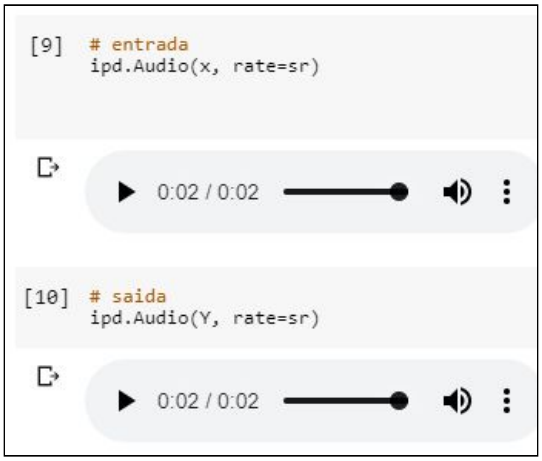


Então, aplica-se o algoritmo overlap-save, e posteriormente é plotado o resultado do sinal filtrado na variável Y:

```
Y = overlapsave(x,h)
plt.plot(abs(np.fft.rfft(Y))) # sinal de saída filtrado
```



Ainda, verificou-se a FFT do sinal de entrada, e do sinal de saída. O próximo passo tem como função verificar como o sinal de entrada se modificou.



É possível acompanhar o projeto através do *Github*, acessando o link:

- https://github.com/yurioliveiraa/python/blob/master/filter.ipynb

3. Conclusão

Apesar da qualidade dos áudios, podemos notar que de acordo com as frequências que modulamos (passa-alta) podemos ver as variações do áudio de saída, o qual deixa passar apenas os graves. Isso comprova que o projeto do filtro, utilizando a Kaiser window, foi satisfatório em um primeiro momento de experimentação da filtragem FIR. Assim, o sistema de filtragem áudio de fato conseguiu trazer os resultados de "limpeza" do sinal desejados.

4. Referências

[1] *Shashoua, Meir; Bundschuh, Paul. DSP Compensation Algorithms for Small Loudspeakers. January 2007.*

[2] *Signal Processing Documentation. Disponível em:* https://docs.scipy.org/doc/scipy/reference/signal.html.

[3] *Overlap-add filtering in Python. Disponível em:* https://github.com/jthiem/overlapadd/blob/master/Overlap-Add%20Filter%20development.ipynb.

[4] *Pylab Documentation. Disponível em:* https://matplotlib.org/api/pyplot_api.html.

[5] *Overlap-save alg. Disponível em:* https://dsp-nbsphinx.readthedocs.io/en/nbsphinx-experiment/nonrecursive_filters/segmented_convolution.html.

[6] *Design FIR & IRR Filters. Disponível em:* https://fiiir.com/.

[7] *LibROSA python package for audio. Disponível em:* https://librosa.github.io/librosa/.