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

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, mostrando de forma prática a utilização dos conceitos difundidos na disciplina para o melhoramento do áudio da fala de uma pessoa em um ambiente barulhento. A disciplina foi ministrada pelo Professor Marcos Hideo Maruo, no 2º semestre de 2018, para o curso de Engenharia de Computação, e este trabalho foi realizado como parte da avaliação da disciplina.

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

1. Introdução

Como já visto no trabalho anterior, as aplicações dos conceitos da disciplina de Processamento Digital de Sinais são inúmeras. No entanto, como o grupo se adaptou bem à metodologia utilizada para a elaboração do Trabalho 1, o grupo concordou seguir nesta mesma linha de estudos.

Os formatos de música digital, estão em todos os cantos e são eles que compõe a trilha sonora da vida de cada usuário. Não importa o nosso estilo musical, a qualidade de áudio em uma música digital é algo de grande importância. Cada formato, cada tamanho, cada arquivo possui uma qualidade diferente e isto influencia diretamente em como nós ouvimos uma música.

Pensando nisso, o grupo resolveu que o foco da aplicação dos conceitos vistos na disciplina para este trabalho seria a qualidade do áudio musical. Dessa forma, a base da

[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

construção do Trabalho 2 se dará conforme o Trabalho 1. Algoritmos, ferramentas para criação do filtro e manipulação deste serão mantidas nesta etapa. Isso porque, anteriormente o grupo não tinha conhecimento das possibilidades de aplicações em *waves*, mas, conhecendo o funcionamento trabalhado anterior, surgiu uma boa oportunidade de continuar com o estudo e aplicação destes métodos.

As Transformadas rápidas de Fourier, Kaiser Window e o algoritmo Overlap – save continuarão sendo muito utilizados neste trabalho. Isso demonstra a grande importância de se ter o conhecimento da aplicação deste e outros métodos para a manipulação de qualquer tipo de sinal digital.

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).

O trabalho consiste em duas etapas. Na primeira etapa ocorre a carga e a filtragem de uma música através da função *OverLap-Save* com um filtro *Passa Faixas*. A segunda etapa consiste na carga de um áudio, onde há cantos de passáros dificultando a audição da fala de uma pessoa. É então aplicado um passa baixas para filtrar apenas a voz da pessoa, e posteriormente um passa altas para intensificá-la.

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.
- **librosa.display:** Possibilitar o *plot* de gráficos através da libROSA.

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, na primeira etapa, é feita a carga do arquivo de áudio *audio/strokes.wav* 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('audio/strokes.wav', sr=44100)
print(x.shape, sr)
```

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.

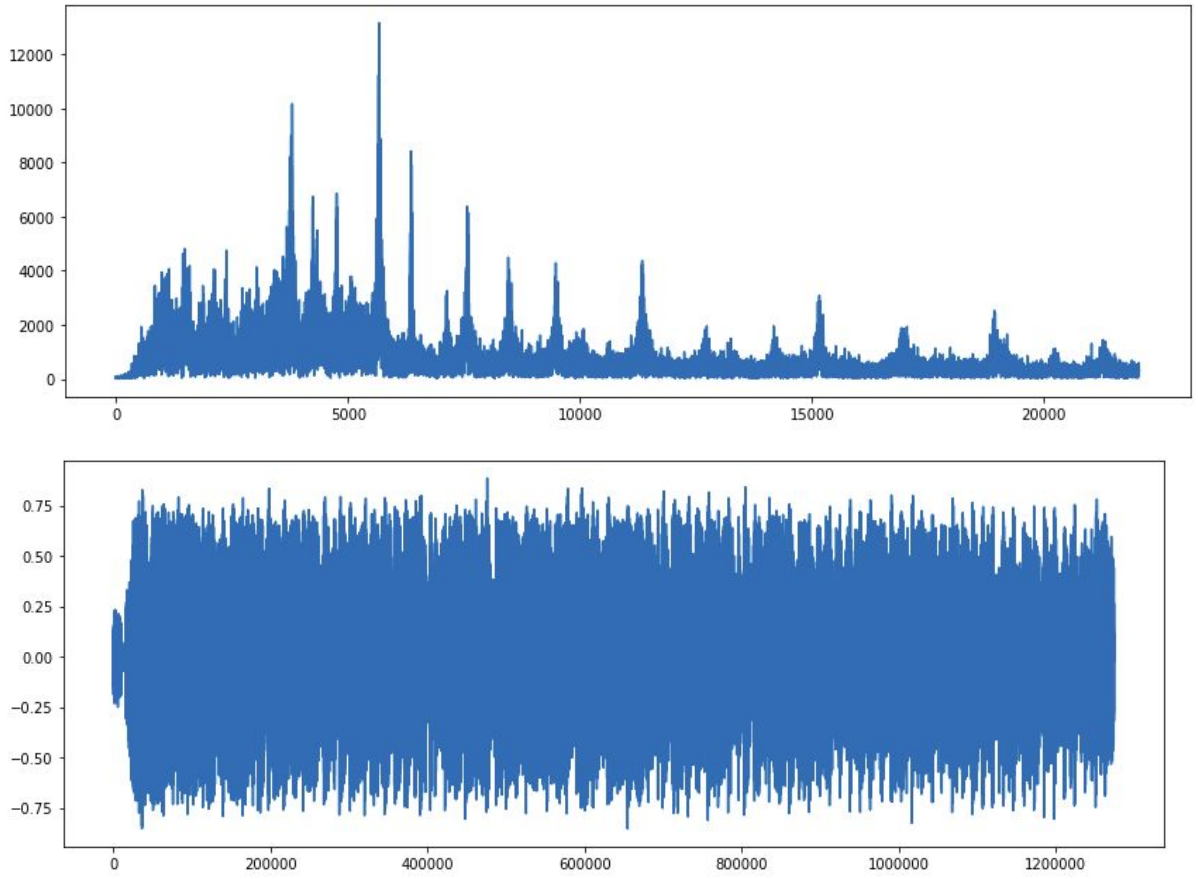
Olhando mais de perto para o resultado da convolução periódica $x_p[k] * h_n[k]$ onde $x_p[k]$ denota um segmento de comprimento P do sinal de entrada de $h_n[k]$. O resultado da convolução linear será do comprimento $(P+N-1)$. O resultado da convolução periódica do período P para $P > N$ sofreria um 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
```

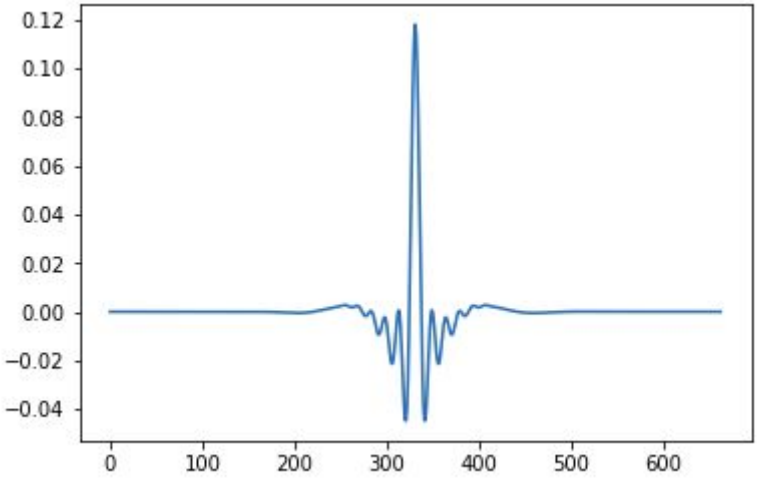
Em seguida, são plotados os gráficos da wave de entrada, onde a primeira figura é utilizando o valor absoluto da transformada discreta de Fourier.

```
# plota amostra do sinal de entrada
fftSig = abs(np.fft.rfft(x))
plt.figure(figsize=(14, 5))
plt.plot(fftSig[:sr//2]) #até 20k que é onde ouvimos
plt.show()

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

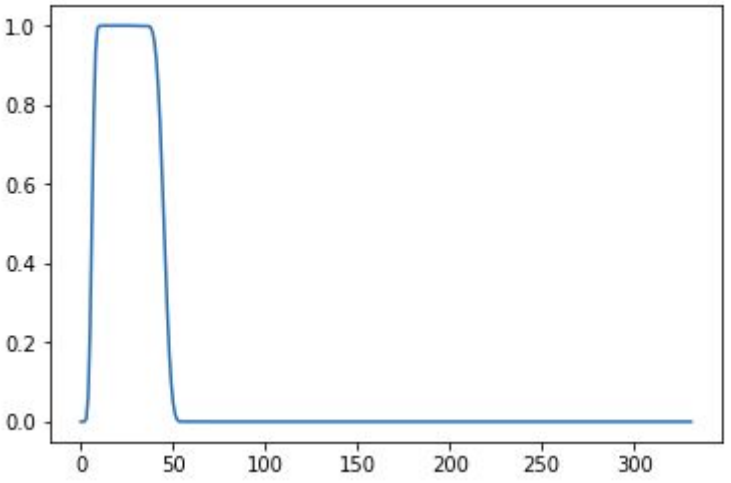


Feito isso, o filtro passa-faixa, gerado no site FIIIR!, foi inserido ao código, e atribuído a variável h. Essa filtragem tem como objetivo permitir a passagem das frequências de uma certa faixa e rejeita (atenua) as frequências fora dessa faixa. As imagens abaixo demonstram o filtro e a FFT do sinal.

```
# Filtro passa faixa


# Configuration.
fS = 44100  # Sampling rate.
fL = 400  # Cutoff frequency.
fH = 3000  # Cutoff frequency.
NL = 461  # Filter length for roll-off at fL, must be odd.
NH = 203  # Filter length for roll-off at fH, must be odd.

# Compute a low-pass filter with cutoff frequency fH.
hlpf = np.sinc(2 * fH / fS * (np.arange(NH) - (NH - 1) / 2.))
hlpf *= np.blackman(NH)
hlpf /= np.sum(hlpf)

# Compute a high-pass filter with cutoff frequency fL.
hhpf = np.sinc(2 * fL / fS * (np.arange(NL) - (NL - 1) / 2.))
hhpf *= np.blackman(NL)
hhpf /= np.sum(hhpf)
hhpf = -hhpf
hhpf[(NL - 1) // 2] += 1

# Convolve both filters.
h = np.convolve(hlpf, hhpf)


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

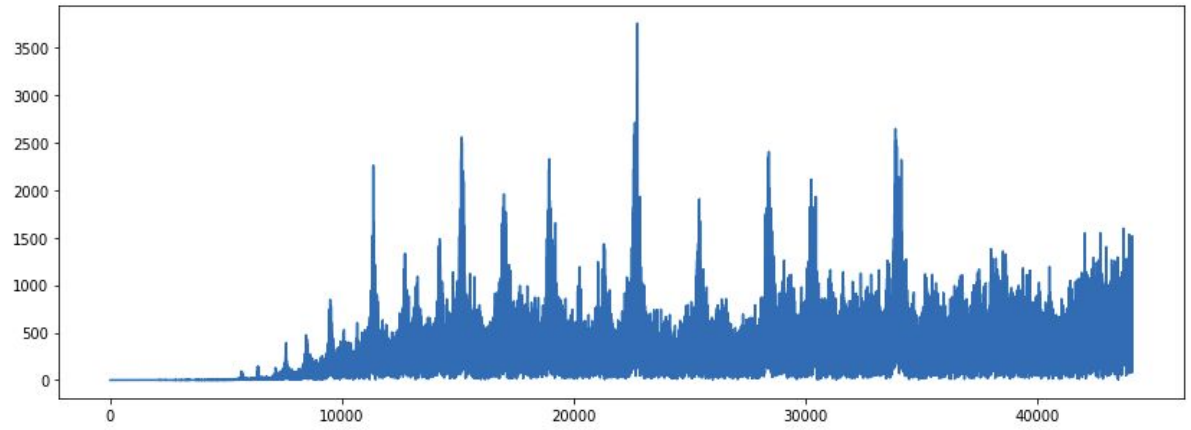



```
[<matplotlib.lines.Line2D at 0x7f1fc3b04128>]
```



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

```
Y = overlapsave(x,h)
# Y = np.convolve(x,h)
plt.figure(figsize=(14, 5))
plt.plot(abs(np.fft.rfft(Y))[:sr]) # sinal de saída filtrado
```

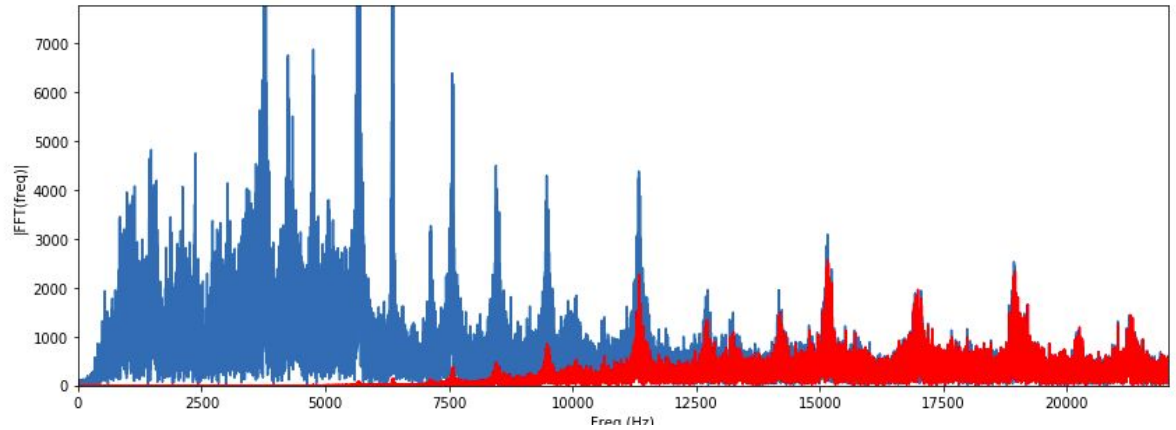


Ainda, verificou-se a FFT do sinal de entrada, e do sinal de saída. O próximo passo tem como função verificar como o sinal de entrada se modificou.

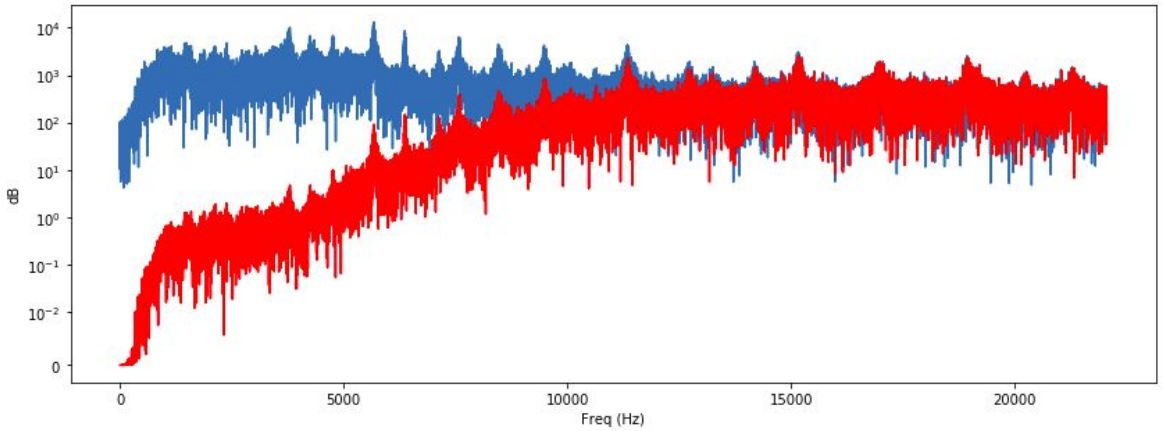
```
# entrada
ipd.Audio(x, rate=sr)
```


```
# saida
ipd.Audio(Y, rate=sr)
```

Uma comparação entre o sinal de entrada e o sinal de saída pode ser vista pela imagem a seguir.



Aplicando a escala dB na imagem, podemos ter uma melhor ideia do funcionamento do filtro.



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

- https://github.com/da3mons/Python/blob/master/filter2.ipynb

Na segunda etapa do trabalho, é efetuada a carga do áudio em que há o canto dos pássaros, *audio/birdssnr10.wav* que também encontra-se na pasta atual do projeto, novamente utilizando o librosa.

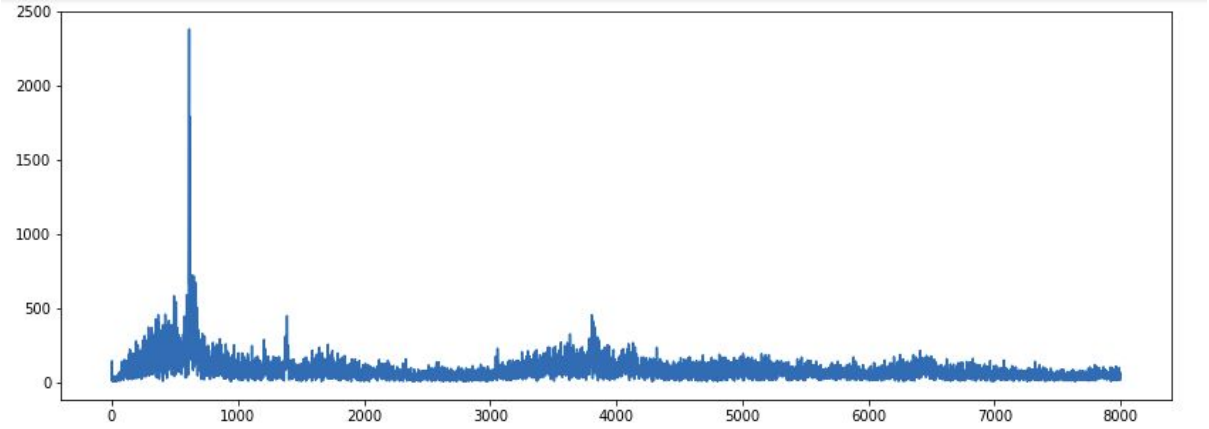
```
x, sr = librosa.load('audio/birdssnr10.wav', sr=16000) # SNR = -10dB
print(x.shape, sr)
```

Após a carga do áudio, foi gerado um filtro *passa-baixa* pelo site FIIIR!, e foi inserido ao código, atribuído a variável h. Essa filtragem tem como objetivo permitir a passagem de baixas frequências sem dificuldades e atenua (ou reduz) a amplitude das frequências maiores que a frequência de corte.

```
# low pass
# Configuration.
fS = 16000  # Sampling rate.
fL = 2500  # Cutoff frequency.
N = 149  # Filter length, must be odd.

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

# Apply window.
h *= np.blackman(N)

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

Para esta segunda etapa, ao invés do algoritmo OverLap-Save utilizou-se a função np.convolve, que retorna a convolução linear discreta de duas seqüências unidimensionais.

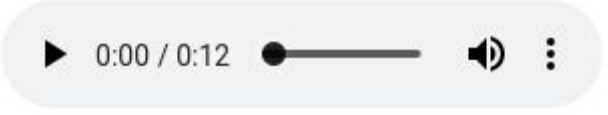
```
Y = np.convolve(x,h)
plt.figure(figsize=(14, 5))
plt.plot(abs(np.fft.rfft(Y))[:sr//2]) # sinal de saida filtrado
```



Verificou-se o sinal de entrada e a saída, após aplicar a convolução com o filtro passa-baixa, para saber como o sinal de entrada se modificou.

```
ipd.Audio(x, rate=sr)
```


```
ipd.Audio(Y, rate=sr)
```



Por fim, adicionou-se então um filtro *passa-altas*, que permite a passagem das frequências altas com facilidade, porém atenua (ou reduz) a amplitude das frequências abaixo da frequência de corte, para melhorar então o áudio já filtrado (sem o som dos pássaros).

```
# hi-pass
# Configuration.
fS = 16000  # Sampling rate.
fH = 400  # Cutoff frequency.
N = 295  # Filter length, must be odd.

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

# Apply window.
h *= np.blackman(N)

# 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
```

Aplicou-se novamente a convolução com o filtro *passa-altas* através da função np.convolve.

```
Y2 = np.convolve(Y,h)
plt.figure(figsize=(14, 5))
plt.plot(abs(np.fft.rfft(x))[:sr])
plt.plot(abs(np.fft.rfft(Y))[:sr]) # sinal de saida filtrado
```

Por fim, verificou-se a saída do áudio filtrado.

```
ipd.Audio(Y2, rate=sr)
```

3. Conclusão

Para este trabalho foram efetuados dois experimentos separadamente, no primeiro podemos notar que a utilização do filtro *passa-faixa* na música, ele atenuou sinais acima da frequência de corte desejada como o esperado.

No segundo experimento, ou segunda etapa do trabalho, efetuou-se a carga de um áudio com uma fala e com cantos de pássaros, impossibilitando assim a audição da fala. Utilizou-se um filtro *passa-baixas* para limpar o barulho feitos pelos pássaros, e posteriormente um filtro *passa-altas* para intensificar a fala.

Isso comprova que para ambas as etapas os projetos dos filtros, utilizando a Kaiser window, foi satisfatório em um primeiro momento de experimentação da filtragem FIR. Assim, o sistema de filtragem de áudio de fato conseguiu trazer os resultados desejados para ambos projetos.

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/.