UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

SEMESTRE 2023.2

Projeto Computacional de Filtro – Transformada de Fourier de Tempo Discreto

ALUNOS: João Vitor de Oliveira Fraga e Abraão de Carvalho Albuquerque

MATRÍCULA: 537377 e 538286

CURSO: Engenharia de Telecomunicações

PROFESSOR: Igor Moaco Guerreiro

DISCIPLINA: Sinais e Sistemas

RESOLUÇÃO

Foi dado um áudio .wav da música tema de “Star Wars”com 1 minuto de duração (pessoalmente adorei a música).

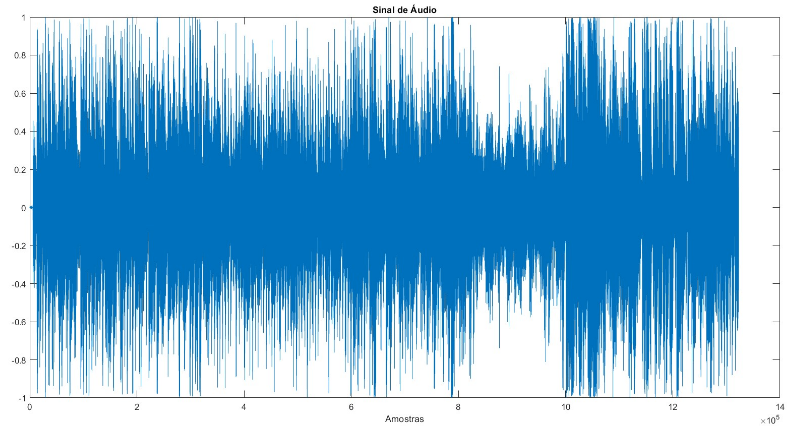
Para solucionarmos o problema, utilizamos dois ambientes diferentes para resolução da questão, iremos apresentar primeiro a resolução feita no MATLAB e após isso a resolução em Python.

Optamos por fazer isso para que tivessemos mais opções após a resolução do problema.

MATLAB

Após carregar o áudio e plotarmos o sinal temos:

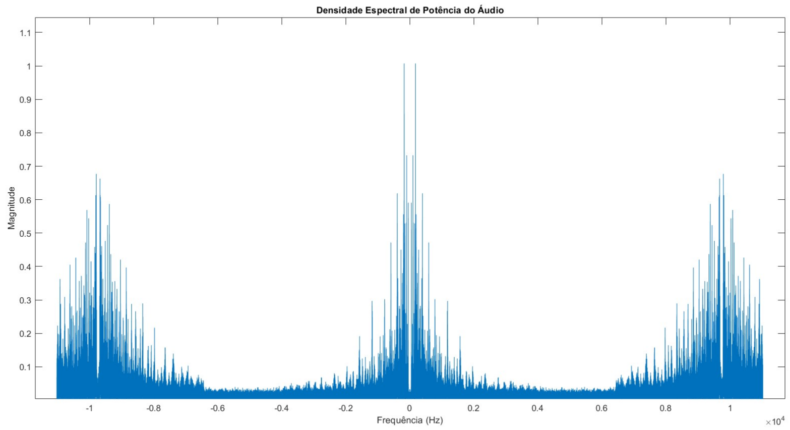
Figura 1: Sinal $x[n]$



Fonte: Elaborado pelo Autor.

Depois de plotarmos o sinal em função do tempo, calculamos sua Transformada de Fourier para fazer o sinal no domínio do tempo $x[n]$ ir para o domínio da frequência $X(e^{j\omega})$. Após isso plotamos a Densidade Espectral de Potência (DEP) do sinal, obtendo a seguinte figura:

Figura 2: Sinal $X(e^{j\omega})$



Fonte: Elaborado pelo Autor.

Quando fazemos um análise da DEP, conseguimos inferir que a partir de $4000Hz$ temos apenas ruído, então precisamos de um filtro passa-baixa de 4000Hz.

Para conseguir fazer isso utilizando de um sistema LIT usamos os conhecimentos que obtemos na cadeira de Sinais e Sistemas. A primeira propriedade é saber que quando temos a multiplicação em um domínio, quando passarmos para outro teremos uma convolução e vice-versa. Ex:

$y[n] = x[n] * h[n]$ quando fazemos a transfromada $Y(e^{j\omega}) = X(e^{j\omega})H(e^{j\omega})$

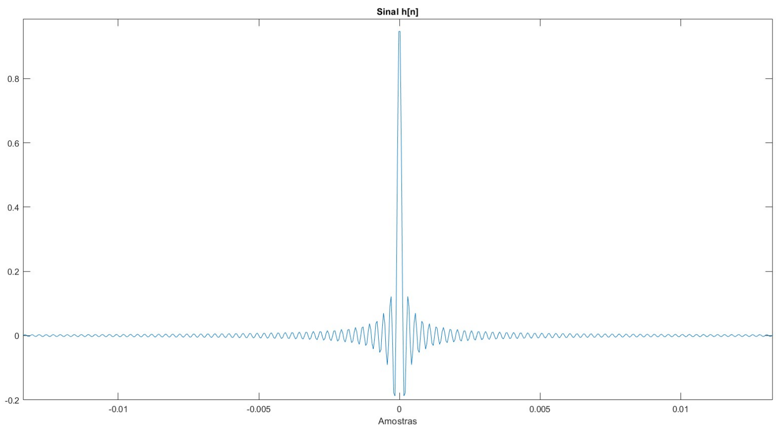
$Y(e^{j\omega}) = X(e^{j\omega}) * H(e^{j\omega})$ quando fazemos a transformada inversa $y[n] = x[n]h[n]$

Com esse conhecimento e sabendo que quando temos uma função sinc, ou seja $h[n] = \frac{sen\omega_c n}{\pi n}$ quando sofre a Transformada de Fourrier, teremos um quadrado unitário que vai de $-\omega_c$ até ω_c, logo:

$$H(j\omega) = \begin{cases} 1 & \text{se } |\omega| \leq \omega_c \\ 0 & \text{c. c} \end{cases}$$

Com essas informações, primeiro definimos uma função $h[n]$ que seja uma sinc com ω_c(Frequência de Corte) igual a $4000Hz$, que foi a escolhida depois de fazer a análise da DEP de $x[n]$, quando definimos e plotamos o sinal $h[n]$ obtemos a seguinte imagem:

Figura 3: Sinal $h[n]$



Fonte: Elaborado pelo Autor.

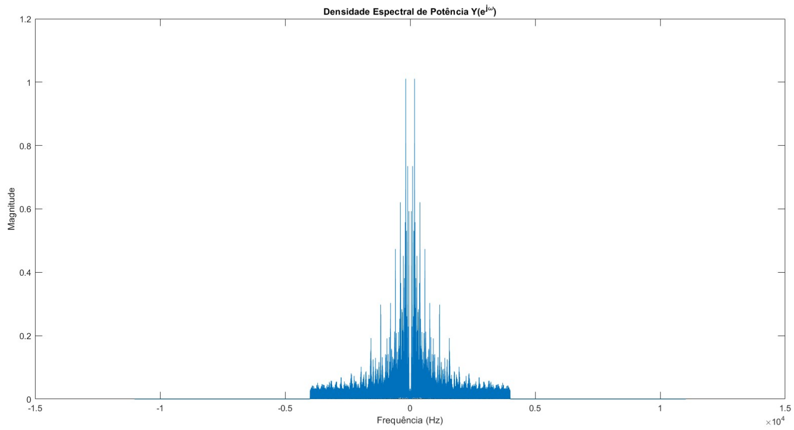
Com isso, temos tanto o sinal $x[n]$ quanto $h[n]$, se colocarmos eles no domínio da frequência teremos que para valores maiores do que a frequência de corte o sinal $Y(e^j\omega)$ será 0, fazendo assim a filtragem passa-baixa, logo se fizermos $Y(e^{j\omega}) = X(e^{j\omega})H(e^{j\omega})$ iremos obter um sinal filtrado, logo o áudio sem ruído usando o método da multiplicação na frequência.

Tal ação implica que conseguimos obter o mesmo áudio sem ruído usando do método da convolução no tempo.

- Método da Multiplicação na Frequência

Para realizar esse método usamos os sinais $X(e^{j\omega})$ e $H(e^{j\omega})$ obtidos anteriormente e os multiplicamos, quando fazemos isso conseguimos a seguinte DEP:

Figura 4: DEP $Y(e^{j\omega})$

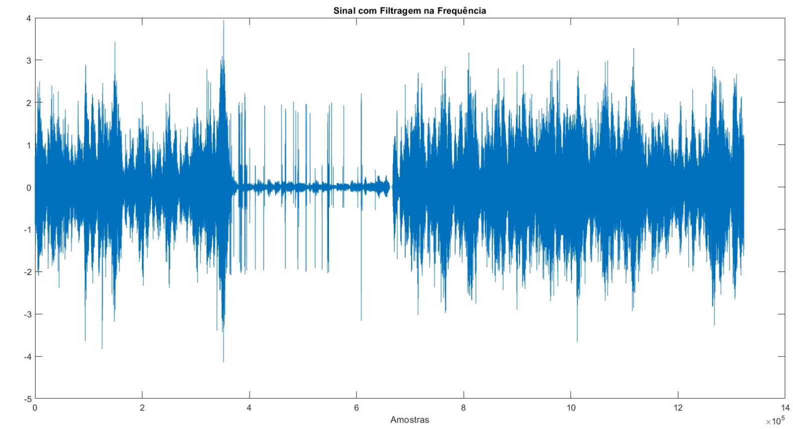


Fonte: Elaborado pelo Autor.

Observamos que o filtro passa-baixa foi um sucesso, já que toda a informação que tinha depois da frequência de corte foi para 0.

Após fazer a multiplicação, basta usarmos a “ifft”que faz a Transformada Inversa de Fourier, trazendo o sinal de volta para o domínio do tempo, quando fazemos isso obtemos um sinal quase que satisfátorio.

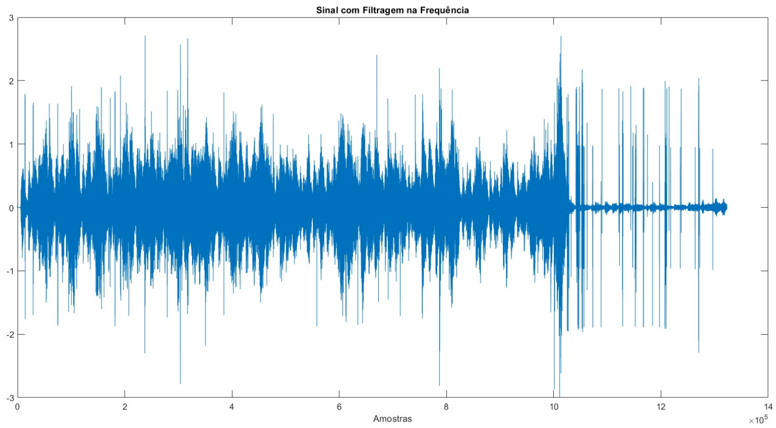
Figura 5: Sinal $y[n_1]$ errado



Fonte: Elaborado pelo Autor.

Nesse sinal nós temos o sinal filtrado, no entanto os primeiros 30 segundos do áudio são na realidade a parte final, enquanto os últimos 30 segundos do áudio são a parte inicial, não consigo explicar o motivo disso acontecer tendo em vista que o filtro foi um sucesso. Contudo eu mudei o sinal $y[n_1]$ para que tivessemos o áudio correto, ficando então com o seguinte áudio filtrado: No arquivo .zip enviado esse arquivo está salvo como "Filtro Multiplicacao"para que

Figura 6: Sinal $y[n_1]$ correto



Fonte: Elaborado pelo Autor.

consiga vê o sucesso da filtragem.

- Método de Convolução no Tempo

Nesse método usamos os valores já conhecidos de $x[n]$ e $h[n]$ e aplicamos uma convolução nesse sinal, sem a necessidade de realizar a Transformada de Fourrier. Pode dar a impressão que esse método é mais rápido, já que não é preciso mudar o domínio do sinal, contudo, a convolução exige uma capacidade computacional muito maior.

Observamos isso por conta que quando usamos o Método da Multiplicação na Frequência os códigos rodam em menos de 1 segundo, já na convolução demora mais para obtermos os resultados.

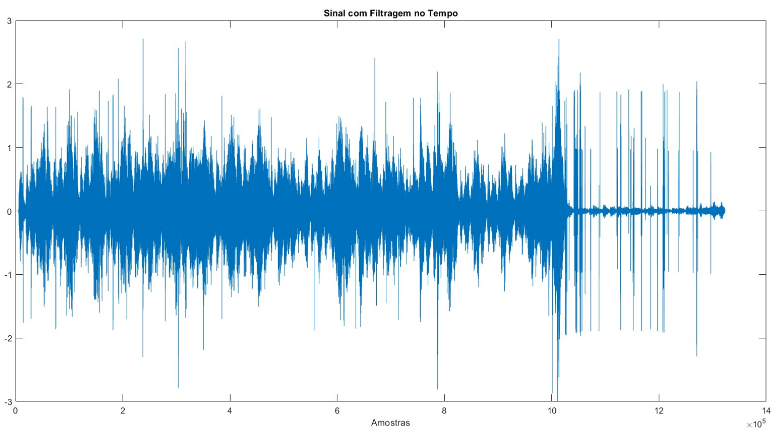
Indo para os resultados, quando é feita a convolução obtemos um sinal filtrado, sem "troca"no tempo, como visto no método anterior, contudo temos outro problema.

Quando convolui dois sinais de mesmo tamanho, o sinal resultante é maior porque estamos "deslizando"um sinal sobre o outro e somando os valores que se sobrepõem.

Com isso em mente e para obter o sinal filtrado correto, dividi o áudio para que começasse apenas no momento que tem informação e acabasse quando acabasse as informações diferente de zero.

No final das contas, o áudio filtrado $y[n_2]$ é obtido a partir da convolução de $x[n]$ com $h[n]$, obtendo o seguinte sinal:

Figura 7: Sinal $y[n_2]$

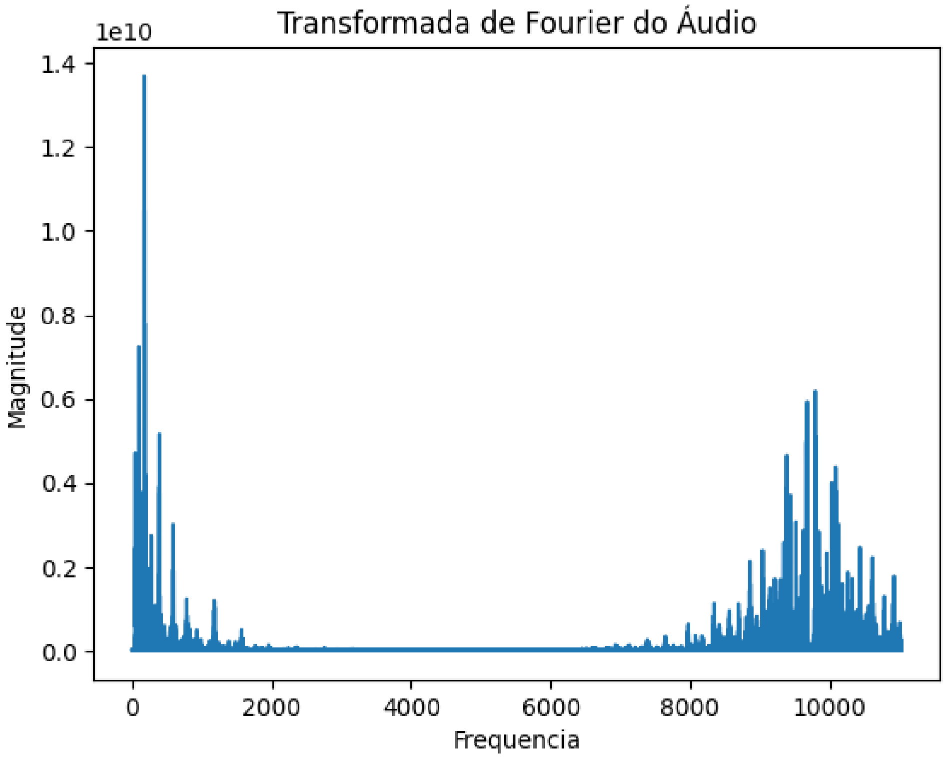


Fonte: Elaborado pelo Autor.

Python

Utilizando um método de separação que não envolve a função sinc e implementando-o em Python, podemos empregar a Transformada de Fourrier para obter a magnitude em frequência, de maneira análoga ao realizado no ambiente MATLAB. Os resultados da FFT em Python são apresentados no Gráfico a seguir.

Figura 8: Sinal na Frequência

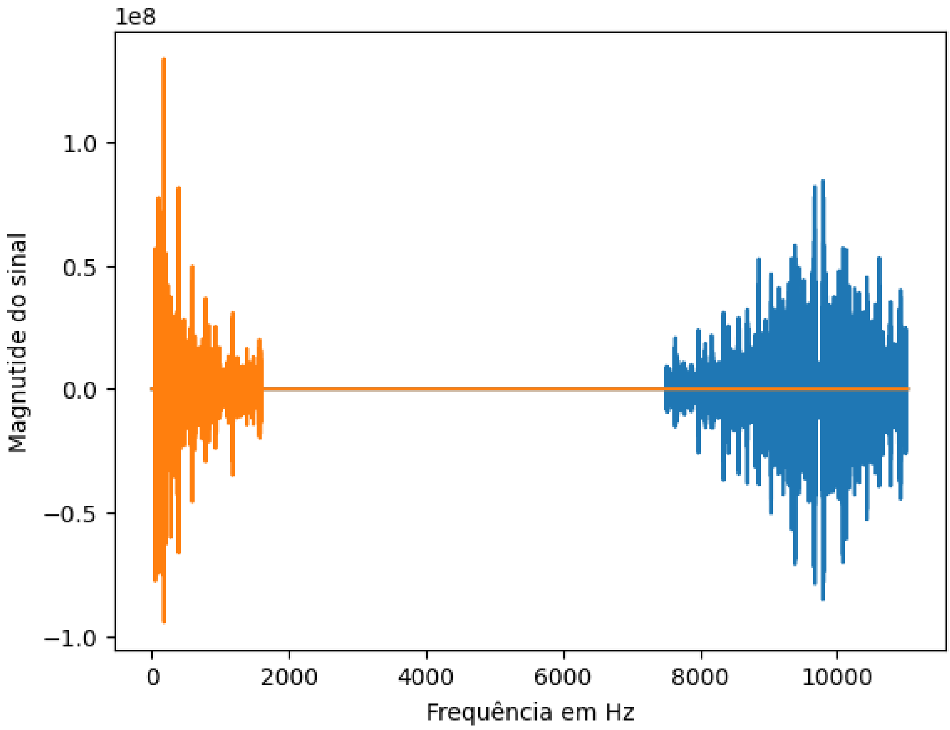


Fonte: Elaborado pelo Autor.

Conforme previsto, o gráfico exibe resultados idênticos aos obtidos no MATLAB. Procedendo com a etapa de separação dos áudios, realizamos a cópia do vetor e zeramos as componentes em que a frequência está fora do intervalo entre 40 e 1600 Hz, obtendo assim o áudio principal de Star Wars. O segundo vetor, por sua vez, também tem os valores zerados fora das frequências entre 8000 e 10500 Hz, correspondentes ao áudio sobreposto à música. Ao realizar a transformada reversa e salvar o áudio, é possível ouvir os áudios separados com qualidade, sem interferências mútuas.

A seguir, apresentamos o gráfico que exibe as frequências dos novos áudios.

Figura 9: Frequência dos Novos Áudios



Fonte: Elaborado pelo Autor.

Conclusão

Para comparar a filtragem feita usando diferentes métodos em Python e MatLab eu utilizei a própria função do MatLab de filtro passa-baixa, para comparar os áudios de saída. No arquivo .zip eu chamei esse áudio de “Filtro FuncaoPassaBaixa”.

Minha análise foi que o filtro feito por nós teve um desempenho bem semelhante.

Apesar de termos dois problemas no código em MatLab, um para cada tipo de filtragem, foram problemas fáceis de contornar e de resolver.

Além do mais utilizamos o Python para implementar um outro método de filtragem, mostrando assim que conseguimos separar o áudio de diversas maneiras, e a melhor de ser usada depende simplesmente do problema que será apresentado.

Vale lembrar que os códigos feitos em MatLab estão na pasta com mesmo nome e os códigos feitos em Python estão na pasta de mesmo nome.

Concluimos então que o trabalho de filtragem foi um sucesso.