Projeto de filtro digital



**Bruno Macabeus Mendes de Aquino**

Engenharia da Computação

# Sumário

[**Sumário**](#_rfslnw23d4qz) **1**

[**Introdução**](#_a6wv7gedm9n) **2**

[**Fundamentação teórica**](#_7db5fipppbq) **3**

[Sinais](#_6svim7pendva) 3

[Processamento digital de sinais](#_pgf5i5qsmryy) 3

[Filtros digitais](#_y41ezhtt7puy) 3

[Filtros FIR](#_i6l8jennqb2y) 3

[Filtros IIR](#_8nsh2atq3j5x) 4

[Tipos de filtros](#_ceivw0mqmak1) 5

[Passa alta](#_optmtfnyj06e) 5

[Passa baixa](#_5s2sg6idot2k) 5

[Passa faixa](#_vgdoxc9974m) 6

[Notch](#_pbf1wn1psqau) 6

[Python](#_7w2lkdrn7opj) 7

[**Metodologia**](#_njn950wvfr5l) **8**

[Analise preliminar do áudio](#_shaf8jiaqzc1) 8

[Recolher informações do WAV](#_nw9176py96sc) 8

[Gráfico no tempo e frequência antes dos filtros](#_ey7irin9j78f) 9

[Projeto com filtro IIR](#_hjdku59avobo) 11

[Projeto com filtro FIR](#_wk5dchcoo1jc) 17

[**Resultados**](#_8pqqzg9764mp) **24**

[**Conclusão**](#_xr23kd38acnv) **25**

[**Anexos**](#_590yddphcb81) **26**

[Anexo A](#_1m9wqfy9hycy) 26

[Anexo B](#_xse52mcbgzec) 30

[**Bibliografia**](#_90qz1j2zs7ji) **32**

# Introdução

Este trabalho apresenta um projeto para filtrar um áudio em arquivo WAV. No áudio há uma pessoa falando em um ambiente com muita interferência e tem-se como objetivo reduzir significamente essa interferência sobre o som da voz. Para a realização deste projeto optou-se pela linguagem de programação Python, devido a boa documentação e bibliotecas para a linguagem, como as bibliotecas científicas Numpy, Matplotlib e Scipy.

Um requisito adicional para a filtragem do áudio, para fins de aprendizado, é que precisa haver duas versões: uma só com filtros FIR e outra só com filtros IIR. Desse modo, pode-se comparar as vantagens e desvantagem de cada um, tanto a nível de facilidade em programar, como no resultado obtido na filtragem.

A primeira seção deste trabalho aborda a fundamentação teórica sobre a área de processamento digital de sinais. A segunda seção aborda sobre a metodologia e processo para filtrar o áudio. Em seguida, é comentado os resultados e conclusões do resultado obtido na filtragem do áudio. Por fim, na seção de Anexos está à íntegra os códigos usados no desenvolvimento dos filtros.

# Fundamentação teórica

## Sinais

Um sinal, de um modo em geral, é a descrição de um fenômeno ocorrido em um determinado tempo e espaço.[[1]](https://paperpile.com/c/cTtsKG/7XLV) Um simples exemplo é a variação da temperatura ambiente: podemos convencionar em nosso modelo que a variação da temperatura é um sinal e assim registrar a evolução da temperatura ao longo de um tempo.[[2]](https://paperpile.com/c/cTtsKG/bfj3)

## Processamento digital de sinais

A área de processamento digital de sinais caracteriza-se em capturar um sinal do mundo real e trazer para o mundo digital, a fim de efetuar alguma computação.[[3], [4]](https://paperpile.com/c/cTtsKG/MbmQ+L88T) Diversos tipos de sinais podem ser capturados, como sinal de vídeo, temperatura, áudio, assim como diversas computações podem ser realizadas, como soma, subtração, média.[[3]](https://paperpile.com/c/cTtsKG/MbmQ)

Apesar dos sinais do mundo real poderem ser tratados e processados ainda no mundo real, assim sem precisar ter o trabalho de converter para o digital, os meios digitais provém maior velocidade e precisão.[[3]](https://paperpile.com/c/cTtsKG/MbmQ) Outra vantagem é a flexibilidade, da qual pode-se implementar diferentes operações simplesmente fazendo modificações no software.[[5]](https://paperpile.com/c/cTtsKG/CvZ0)

Uma operação comum a se realizar é filtrar o sinal. Essa operação caracteriza-se em reduzir ou melhorar determinada característica de um sinal. Como estamos trabalhando em sinais digitais, essa nossa operação é realizada por um filtro digital, da qual trabalha no tempo discreto com uma amostragem de um determinado sinal.

## Filtros digitais

Os filtros digitais podem ser classificados em Filtros FIR e Filtros IIR. A seguir, veremos mais detalhadamente as principais características de cada grupo.

### Filtros FIR

* São filtros com resposta ao impulso finita
* Possuem memória finita
* São BIBO estáveis
* Não apresentam recursividade
* Podem implementar resposta em fase linear
* A função de transferência é da forma

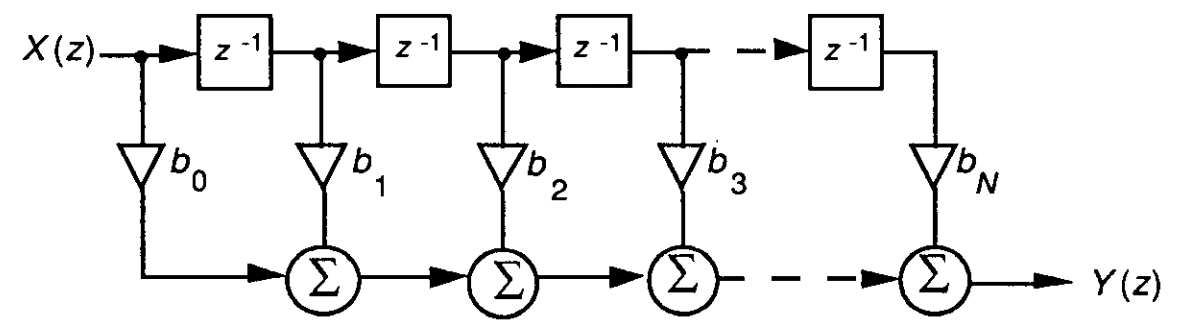


Figura 1. Estrutura para implementar um filtro FIR.[[5]](https://paperpile.com/c/cTtsKG/CvZ0)

### Filtros IIR

* São filtros com resposta ao impulso infinita
* Possuem memória infinita
* A estabilidade depende da posição dos polos
* Possuem resposta em fase não linear
* Para um dado erro de aproximação, necessitam de uma ordem menor se comparado a um filtro FIR
* Possuem recursividade
* A função de transferência é da forma

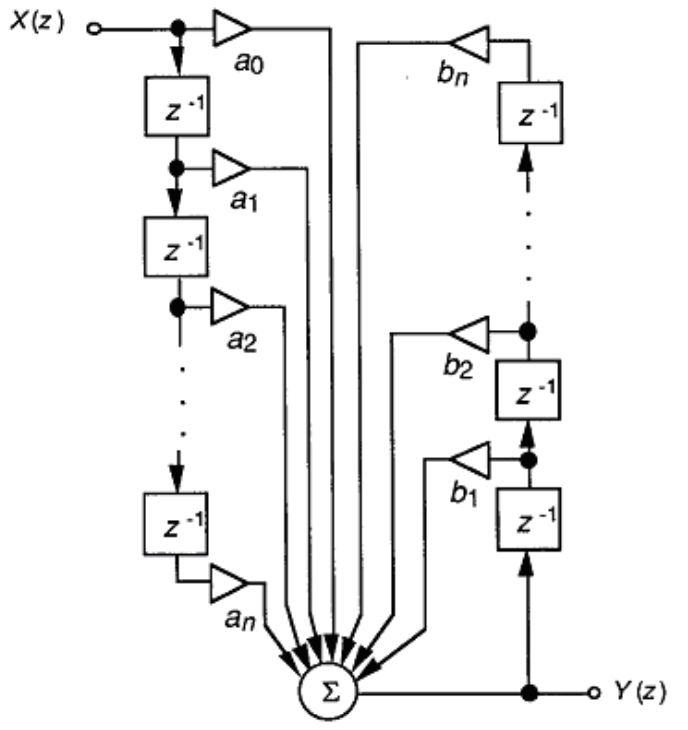


Figura 2. Estrutura para implementar um filtro IIR.[[5]](https://paperpile.com/c/cTtsKG/CvZ0)

### Tipos de filtros

A seguir apresentamos brevemente os filtros mais comuns.

#### Passa alta

Um filtro passa alta é caracterizado por deixar passar sinais nas frequências maiores que a frequência de corte, e atenuar sinais na frequência menor que a de corte.

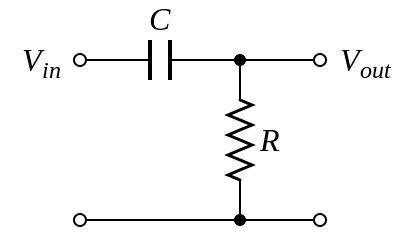


Figura 3. Circuito básico para a implementação de um passa alta.[[6]](https://paperpile.com/c/cTtsKG/yT7W)

#### Passa baixa

É complementar ao passa alta. No caso do passa baixa, ele deixa passar sinais nas frequências menores que a frequência de corte e atenuar sinais na frequência maior que a de corte.

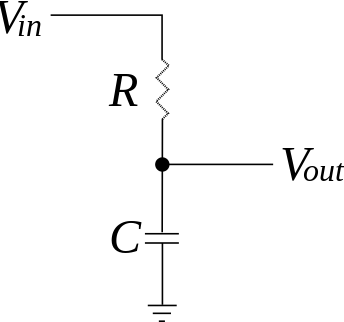


Figura 4. Circuito básico para a implementação de um passa baixa.[[7]](https://paperpile.com/c/cTtsKG/Ht2Z)

#### Passa faixa

Um filtro passa faixa é caracterizado por deixar passar frequências numa determinada faixa, enquanto atenua as frequências fora dessa faixa.

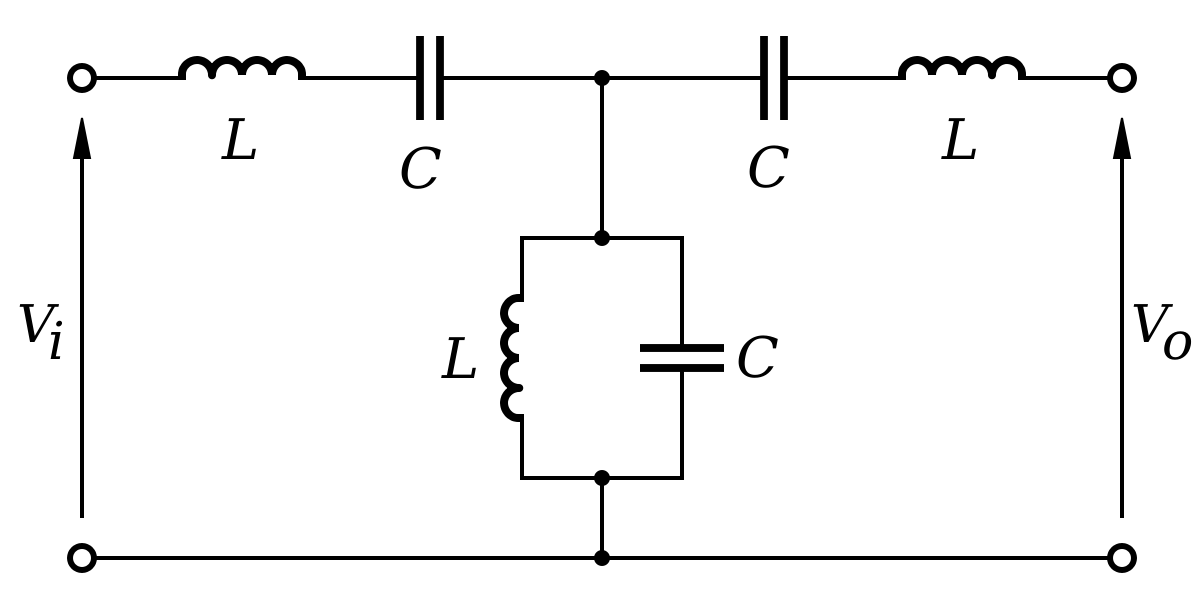


Figura 5. Exemplo de circuito para implementar um passa faixa.[[8]](https://paperpile.com/c/cTtsKG/i45V)

#### Notch

Um filtro notch é caracterizado por rejeitar uma frequência específica.

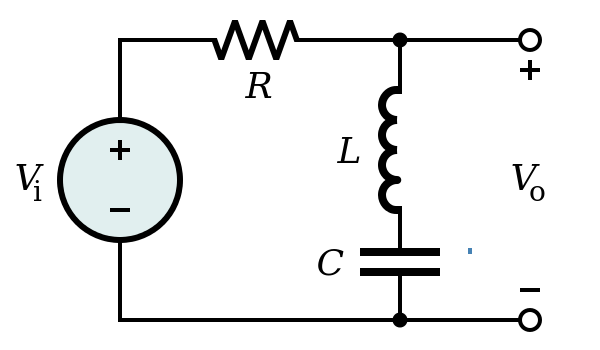


Figura 6. Exemplo de um circuito para implementar um filtro notch.[[9]](https://paperpile.com/c/cTtsKG/avbr)

## Python

Para a realização da filtragem do sinal, optou-se pela linguagem de programação Python, devido a boas bibliotecas científicas e facilidade no desenvolvimento. As bibliotecas científicas usadas foram:

* matplotlib: biblioteca para exibir gráficos
* scipy: biblioteca para operações científicas
* numpy: biblioteca matemática

# Metodologia

Nesta seção é apresentado o passo a passo de como foi feita a filtragem do áudio. O código completo encontra-se completo no Anexo A.

## Analise preliminar do áudio

Antes de começar a desenvolver o filtro, é importante escutar o áudio, para ter uma noção da frequência da voz e das interferências a serem eliminadas. Desse modo, ao escutar o som, de 4 segundos, observa-se que a voz é contínua em todo o áudio e há duas interferências a serem eliminadas: uma oscilante na alta frequência, similar a um som de alarme, enquanto a outra é contínua e de baixa frequência.

## Recolher informações do WAV

O primeiro passo é abrir o som no arquivo WAV, obter a taxa de amostragem nele e normalizá-lo. A normalização do sinal ajuda na análise do sinal e no desenvolvimento do filtro.

Para efetuar essa tarefa, usou-se a biblioteca wave além das já citadas e foi desenvolvida a função presente na Tabela 1.

|  |
| --- |
| Código para abrir um arquivo wav |
| *# abre um arquivo de audio do tipo wav e retorna a frequencia de amostragem dele e o audio normalizado*  *# parametro file\_path: caminho do arquivo a ser aberto*  *# retorno: tupla com (fs, audio normalizado)*  **def** open\_wav(file\_path):  *# open file*  wave\_file = wave.open(file\_path, **'rb'**)  *# get frames*  wave\_audio = wave\_file.readframes(-1)  wave\_audio = np.fromstring(wave\_audio, np.int16).astype(float)  fs = wave\_file.getframerate()  *# normalize*  wave\_audio /= np.max(np.abs(wave\_audio))  *#*  **return** (fs, wave\_audio)  fs, wave\_audio = open\_wav(**'/Users/macabeus/Desktop/pds/fala\_sirene\_tm0.wav'**) |

Tabela 1. Código para abrir um arquivo wav

Executando o código da Tabela 1, se obtem que a taxa de amostragem do arquivo é cerca de 22050.

## Gráfico no tempo e frequência antes dos filtros

Após abrir o arquivo, podemos plotar o gráfico do som no tempo e na frequência. Isso é importante para termos uma noção de como estar o áudio, de qual frequência precisaremos filtrar e termos uma comparação após aplicar os filtros.

|  |
| --- |
| Código plotar o áudio no tempo |
| *# plota grafico do audio no tempo*  *# parametro wave\_audio: audio a ser plotado*  *# parametro fs: frequencia de amostragem*  *# retorno: nenhum*  **def** plot\_audio\_time(wave\_audio, fs, title=**''**):  time = np.linspace(0, len(wave\_audio) / fs, num=len(wave\_audio))  plt.title(title)  plt.plot(time, wave\_audio)  plt.show()  plot\_audio\_time(wave\_audio, fs, title=**'Audio original no tempo'**) |

Tabela 2. Código plotar o áudio no tempo

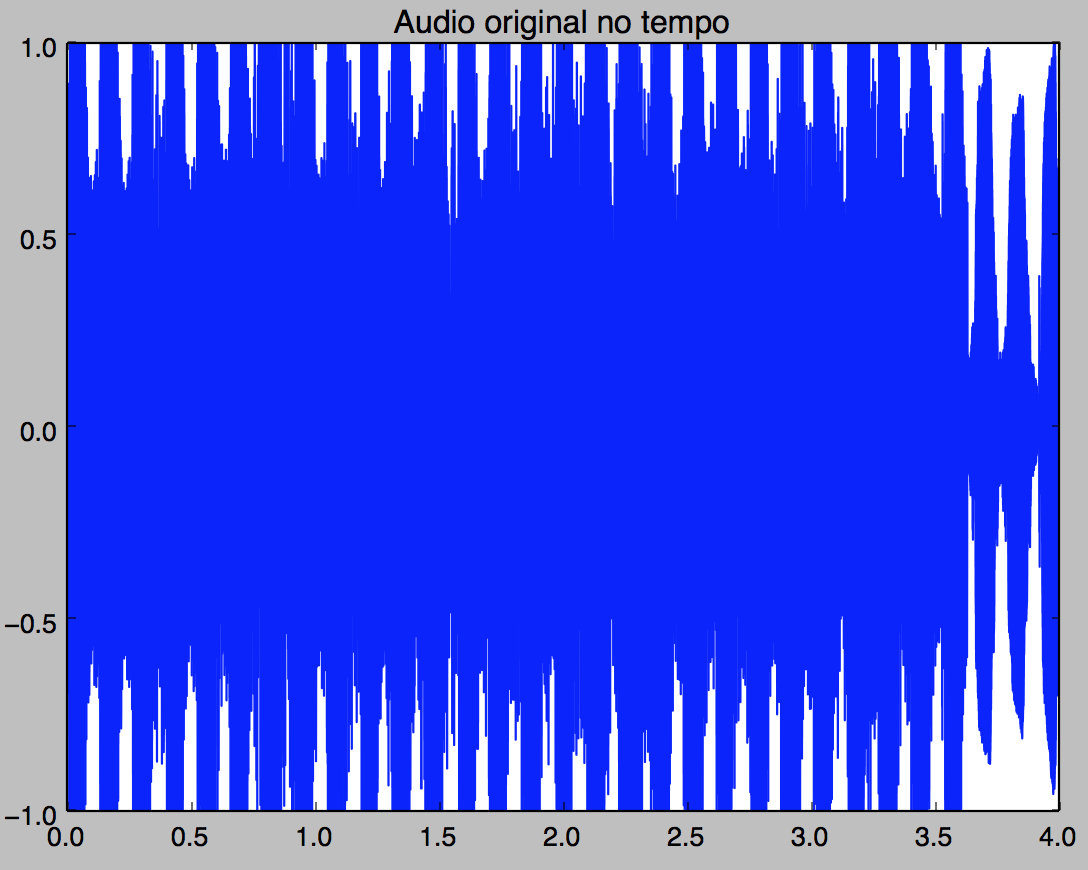


Figura 7. Áudio no domínio do tempo antes da aplicação dos filtros

Como pode-se perceber através do gráfico da Figura 7, o arquivo de som estar com muitas interferências.

|  |
| --- |
| Código para plotar o áudio na frequência |
| **def** normalize\_frequency\_axis(target, fs):  c = fft(target)  d = len(c)  f = np.arange(0, len(c)).astype(float)  f /= d  f \*= fs  **return** f  *# plota o grafico do audio na frequencia*  *# parametro target: audio ou filtro a ser plotado*  *# parametro fs: frequencia de amostragem*  *# retorno: nenhum*  **def** plot\_frequency(target, fs, frequency\_axis=None, title=**''**):  c = fft(target)  d = len(c)  *# normalizando eixo de frequencia, se precisar*  **if** frequency\_axis **is** None:  frequency\_axis = np.arange(0, len(c)).astype(float)  frequency\_axis /= d  frequency\_axis \*= fs  frequency\_axis = frequency\_axis[:len(frequency\_axis) / 2]  *#*  plt.title(title)  plt.plot(frequency\_axis, abs(c[:len(c) / 2]))  plt.show()  frequency\_axis = normalize\_frequency\_axis(wave\_audio, fs)  frequency\_axis = frequency\_axis[:len(frequency\_axis) / 2]  plot\_frequency(wave\_audio, fs, frequency\_axis=frequency\_axis, title=**'Audio original na frequencia'**) |

Tabela 3. Código para plotar o áudio na frequência.

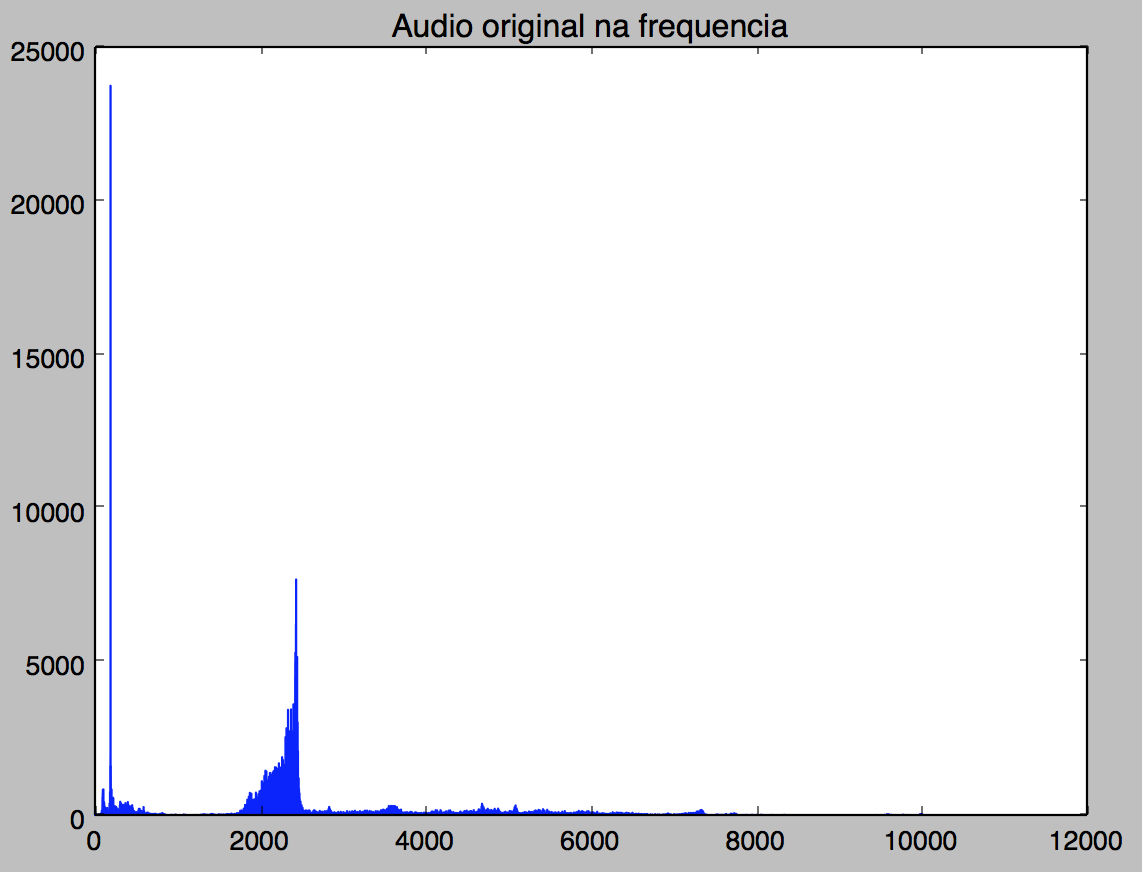


Figura 8. Áudio no domínio da frequência antes da aplicação dos filtros.

Como pode-se observar no gráfico da figura 8, há um grande pico na frequência de 200hz, uma pequena concentração em torno das frequências de 0 a 800hz e uma boa concentração nas frequências de 1500 a 3000hz. Nas frequências acima de 3000hz ainda há uma pequena concentração.

A partir desse gráfico, começou-se a questionar: em quais regiões estavam as interferências, e então precisavam ser filtradas? Qual região estava a voz, e então precisava ser preservada?

## Projeto com filtro IIR

Ao obter o gráfico da Figura 8, se decidiu supor que uma das interferências, no caso, a interferência de alta frequência que é similar a um alarme, estaria na região de 1500 a 3000hz, enquanto a voz estaria na região de 0 a 800hz.

Desse modo, para testar essa hipótese, foi desenvolvido um filtro IIR passa faixa butterworth para eliminar essa região. A escolha do butterworth se deu por duas razões: a região que deseja-se preservar estar longe da região a ser eliminada. Além disso, no butterworth não há oscilações na banda de rejeição nem na banda de passagem. O filtro criado é de quinta ordem e deixa passar as frequências de 200hz a 1000hz.

|  |
| --- |
| Código do filtro IIR passa faixa |
| *# criar filtro*  filter\_order = 5  b, a = signal.iirfilter(filter\_order, [200.0/(fs/2), 1000.0/(fs/2)], ftype=**'butter'**, output=**'ba'**)  *# exibir graficos do filtro*  plot\_filter\_frequency(b, a, fs, title=**'Filtro IIR passa faixa na frequencia'**)  plot\_filter\_iir\_time(b, a, 300, title=**'Filtro IIR passa faixa no tempo'**)  *# aplicar filtro*  wave\_audio\_filtered = signal.lfilter(b, a, wave\_audio)  *# exibir grafico do audio apos a aplicacao do filtro*  plot\_frequency(wave\_audio\_filtered, fs, frequency\_axis=frequency\_axis, title=**'Audio apos aplicacao do filtro passa baixa'**) |

Tabela 4. Código do filtro IIR passa faixa

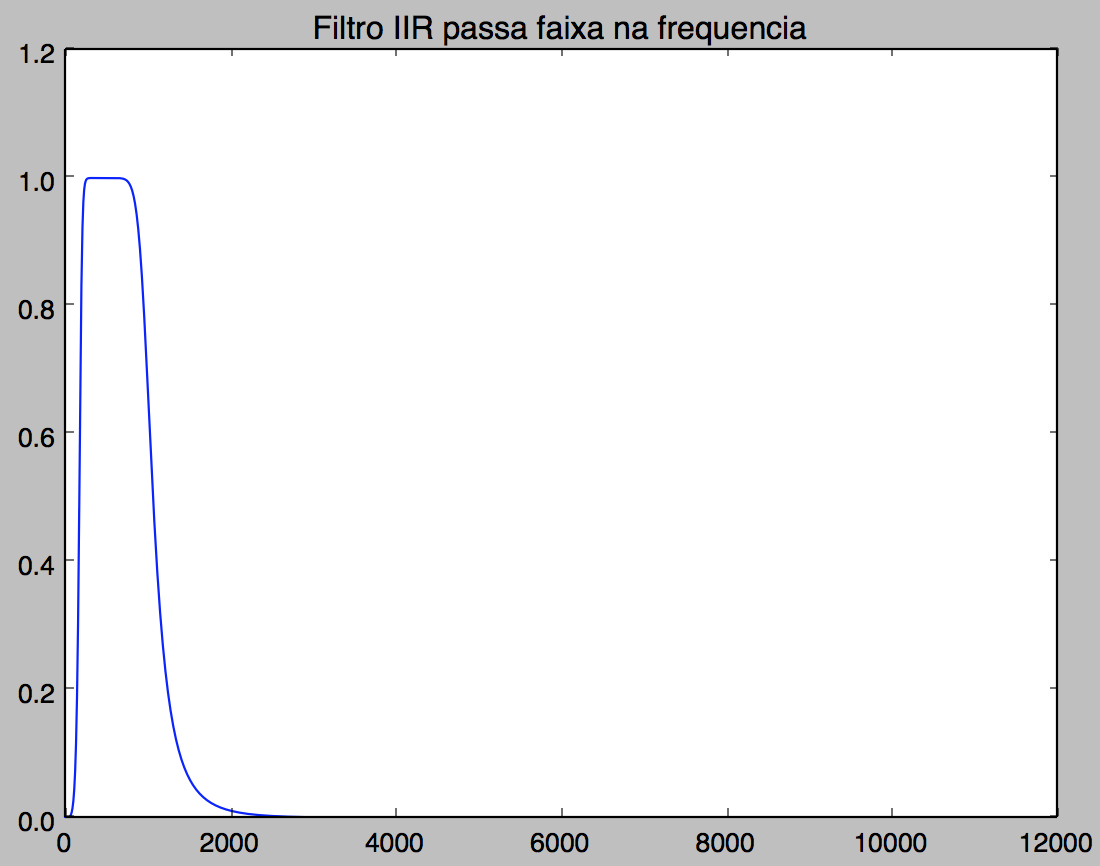


Figura 9: Gráfico na frequência do filtro IIR passa baixa butterworth

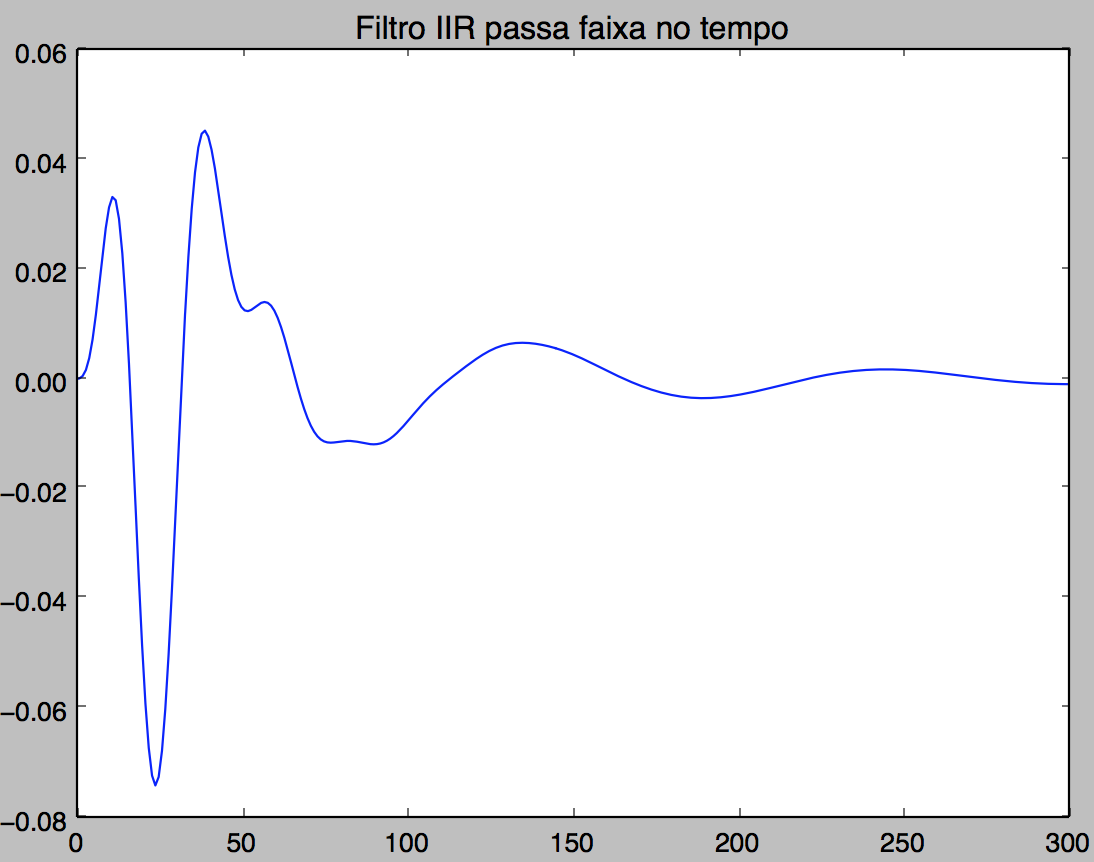


Figura 10. Gráfico de resposta ao impulso do filtro IIR passa faixa butterworth

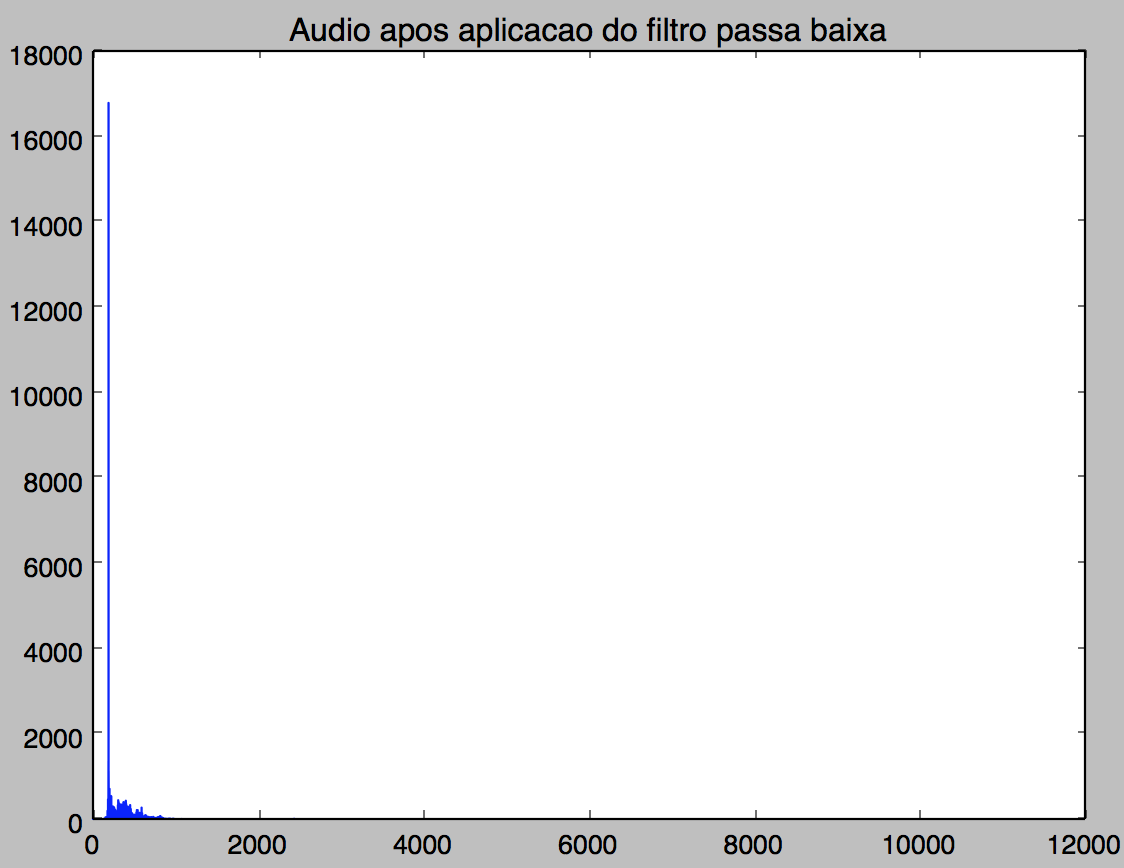


Figura 11: Gráfico do áudio na frequência após a aplicação do filtro butterworth

Tal como pode-se ver na Figura 11, a frequência da região de 1500 a 3000hz foi bem atenuada, enquanto a região de 0 a 800hz sofreu poucas alterações. Ao escutar o áudio após a aplicação do filtro, observa-se que a voz foi bem preservada, enquanto as interferências, principalmente a do "alarme" foi drasticamente reduzida.

Ainda era preciso atenuar a segunda interferência no sinal, a de baixa frequência e com som constante. Como o som é quase constante, uma boa hipótese seria buscar atenuar a frequência mais presente no áudio, no caso, a de 200hz. Como deseja-se filtrar uma frequência específica, uma boa solução seria fazer um filtro notch, e assim foi feito, conforme o código abaixo.

|  |
| --- |
| Código do filtro notch |
| *# criar filtro notch*  f0 = 200.0 *# frequencia a ser removida no sinal*  q = 20.0 *# fator de qualidade*  w0 = f0/(fs/2) *# normalizar a frequencia*  b, a = signal.iirnotch(w0, q)  *# exibir grafico do filtro no dominio da frequencia*  plot\_filter\_frequency(b, a, fs, title=**'Filtro IIR notch na frequencia'**)  plot\_filter\_iir\_time(b, a, 100, title=**'Filtro IIR notch no tempo'**)  *# aplicar filtro*  wave\_audio\_filtered = signal.lfilter(b, a, wave\_audio\_filtered)  *# exibir grafico do audio apos a aplicacao do filtro*  plot\_frequency(wave\_audio\_filtered, fs, frequency\_axis=frequency\_axis, title=**'Audio apos aplicacao do filtro notch'**) |

Tabela 5. Código do filtro notch

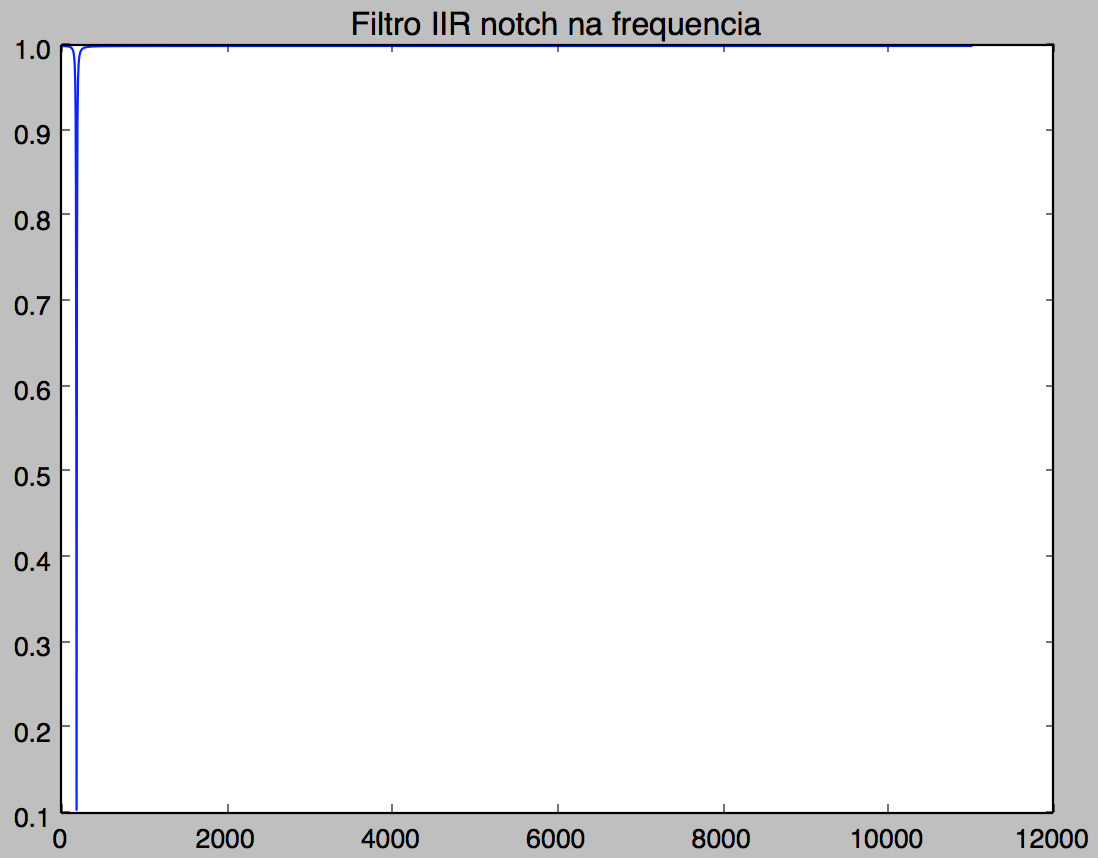


Figura 12. Gráfico na frequência do filtro IIR notch

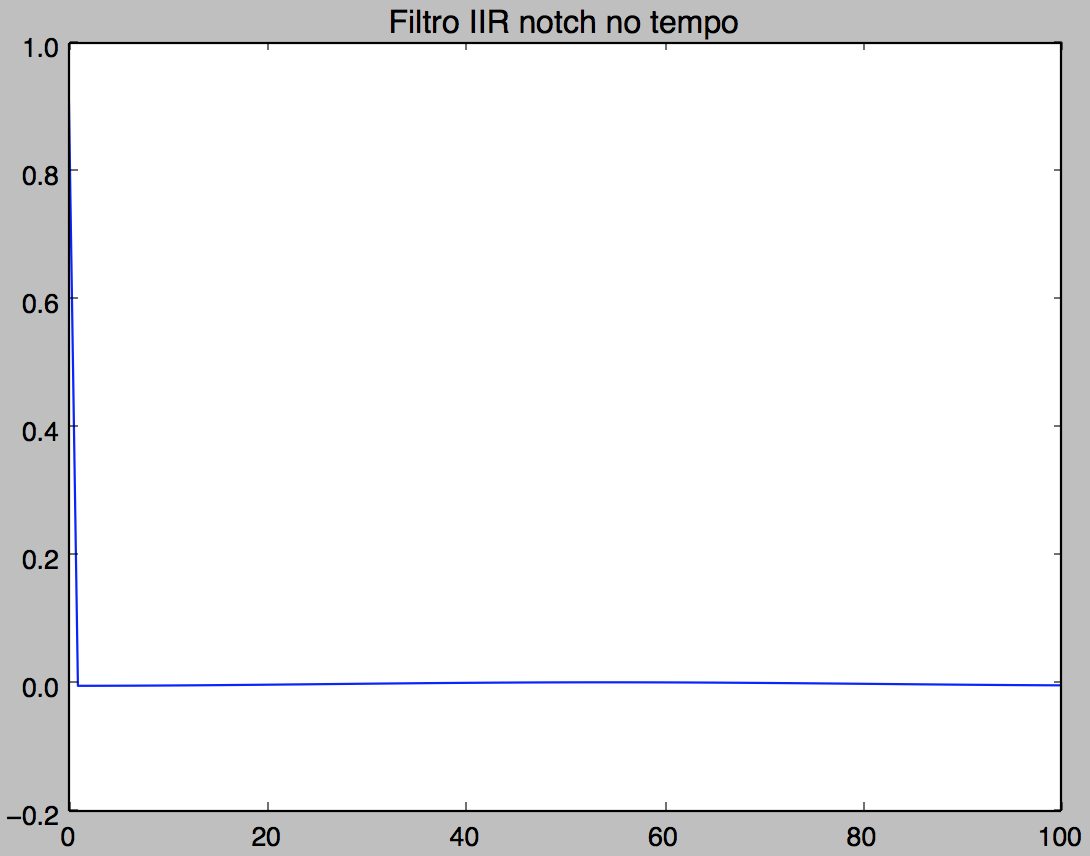


Figura 13. Gráfico de resposta ao impulso do filtro IIR notch

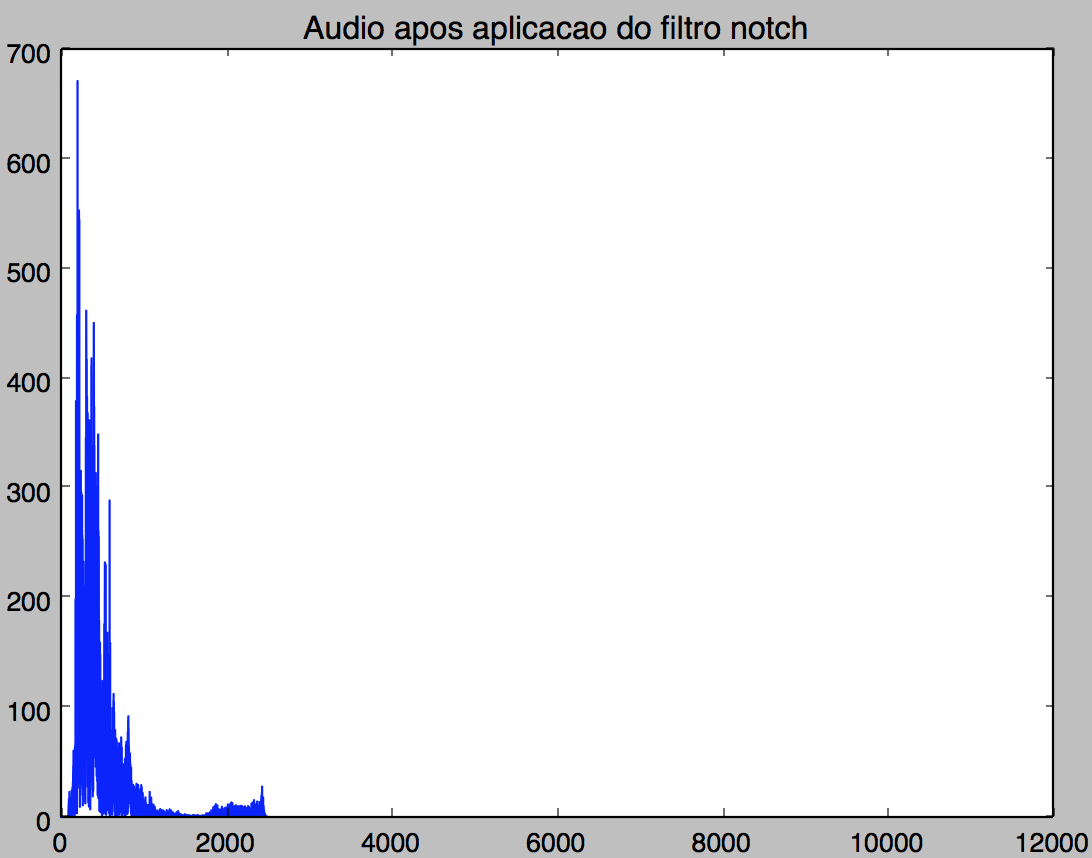


Figura 14. Gráfico do áudio na frequência após a aplicação do filtro notch

Como pode-se ver na Figura 14, a aplicação do filtro notch atenuou drasticamente a frequência de 200hz e, após escutar novamente o áudio, a interferência de baixa frequência foi reduzida drasticamente.

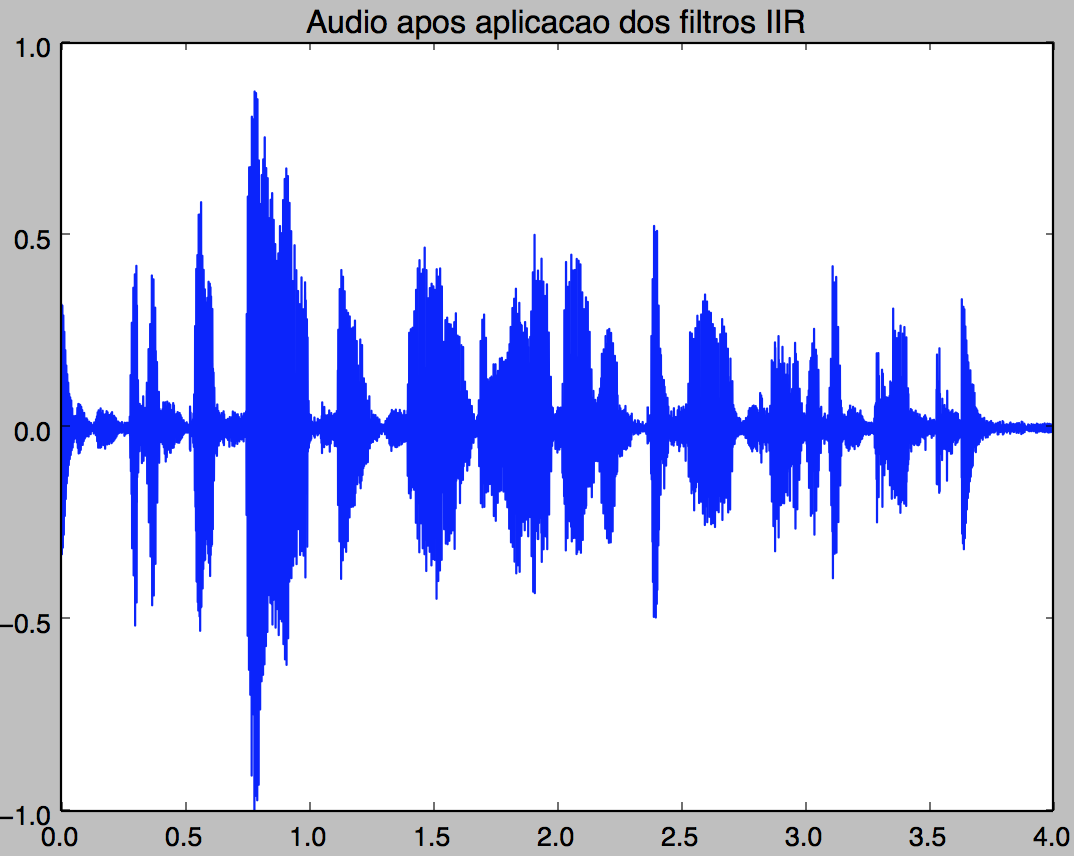


Figura 15. Áudio no domínio do tempo após a aplicação dos filtros IIR.

Na Figura 15 pode-se ver o gráfico do áudio resultado do áudio no domínio do tempo e, comparado-se com o da Figura 7, ou seja, antes da aplicação do filtro, está muito melhor, pois na Figura 9 pode-se ver um gráfico característico de uma pessoa falando. Após a aplicação dos dois filtros, um passa baixa butterworth e um notch, ficou bem mais fácil entender o que a pessoa está falando.

## Projeto com filtro FIR

O desenvolvimento foi, de certo modo, mais fácil do que no filtro IIR, pois agora sabemos quais eram as frequências das interferências. Então, precisávamos buscar resultados parecidos dos filtros IIR com filtros FIR.

No caso do filtro IIR passa baixa butterworth, foi desenvolvido um passa faixa usando a janela blackmanharris. A escolha dessa janela deveu-se por razões parecidas a do filtro IIR butterworth: boa distância entre a frequência a ser preservada e a frequência a ser cortada, e não haver oscilações na banda de passagem nem na banda de rejeição. Infelizmente, a biblioteca Scipy não oferece uma forma prática para se desenvolver um filtro passa faixa, assim precisando criar dois filtros com a mesma ordem e janela, mas com frequência de corte diferentes e depois substituir para de um com a parte do outro.[[10]](https://paperpile.com/c/cTtsKG/TAeH)

|  |
| --- |
| Código para o filtro passa faixa FIR |
| *# criar filtro*  n = 401  a = signal.firwin(n, cutoff=200, window=**'blackmanharris'**, nyq=fs)  b = - signal.firwin(n, cutoff=3100, window=**'blackmanharris'**, nyq=fs)  b[n/2] = b[n/2] + 1  df = - (a+b)  df[n/2] = df[n/2] + 1  *# exibir graficos do filtro*  plot\_frequency(df, fs, title=**'Filtro FIR passa faixa na frequencia'**)  plot\_filter\_fir\_time(df, title=**'Filtro FIR passa faixa no tempo'**)  *# aplicar filtro*  wave\_audio\_filtered = signal.lfilter(df, [1], wave\_audio)  *# exibir grafico do audio apos a aplicacao do filtro*  plot\_frequency(wave\_audio\_filtered, fs, frequency\_axis=frequency\_axis, title=**'Audio apos aplicacao do filtro passa faixa'**) |

Tabela 6. Código para o filtro passa faixa FIR

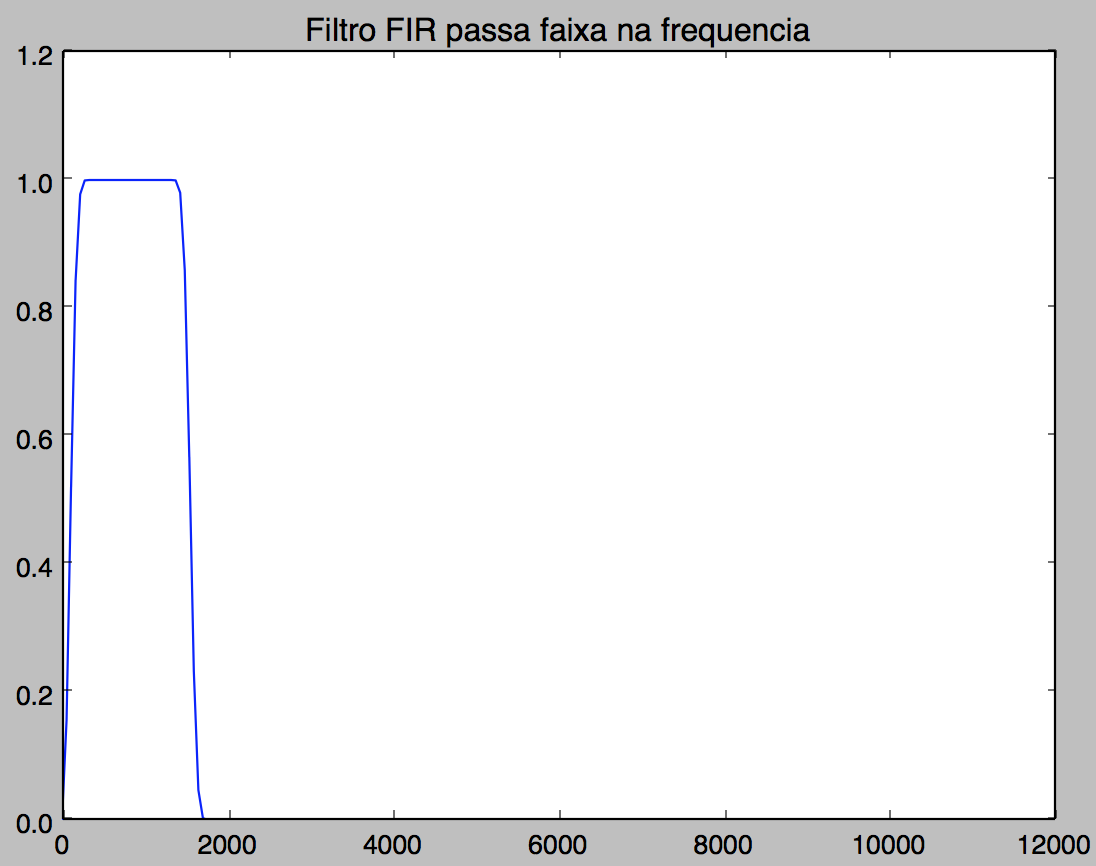


Figura 14. Gráfico na frequência do filtro FIR passa faixa

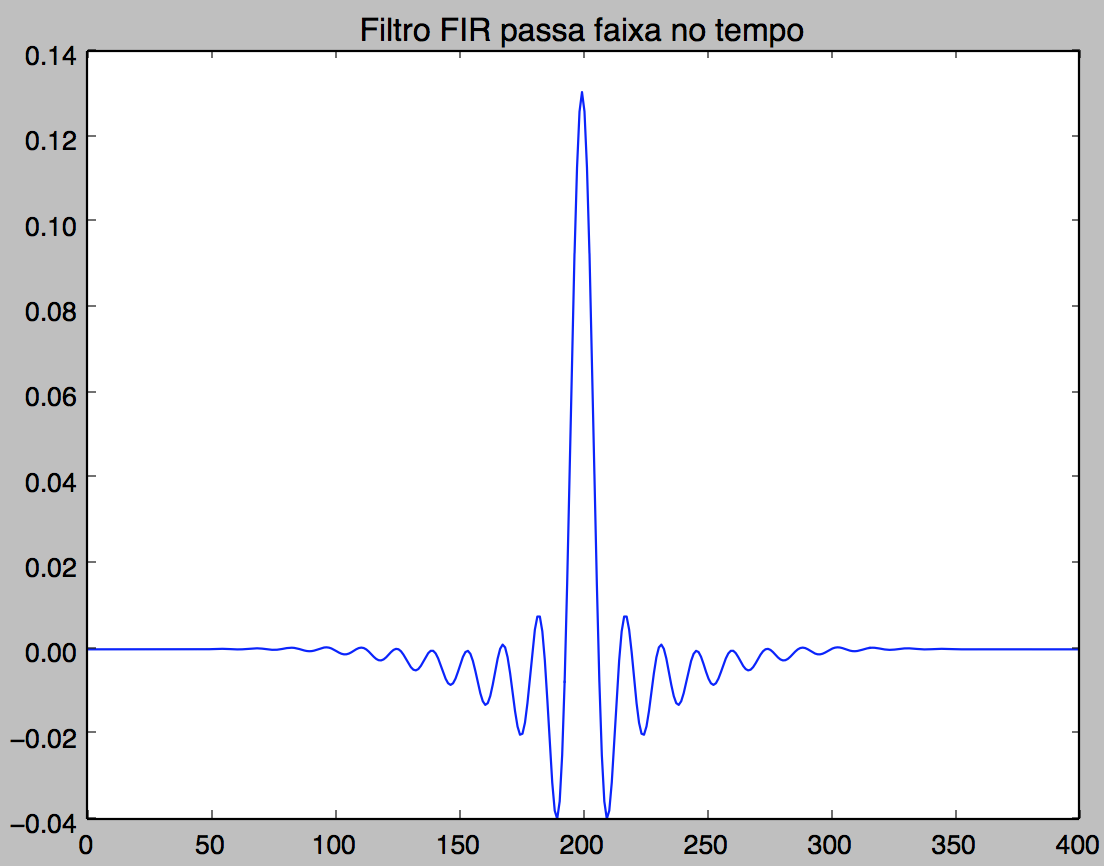


Figura 15. Gráfico de resposta ao impulso do filtro FIR passa faixa

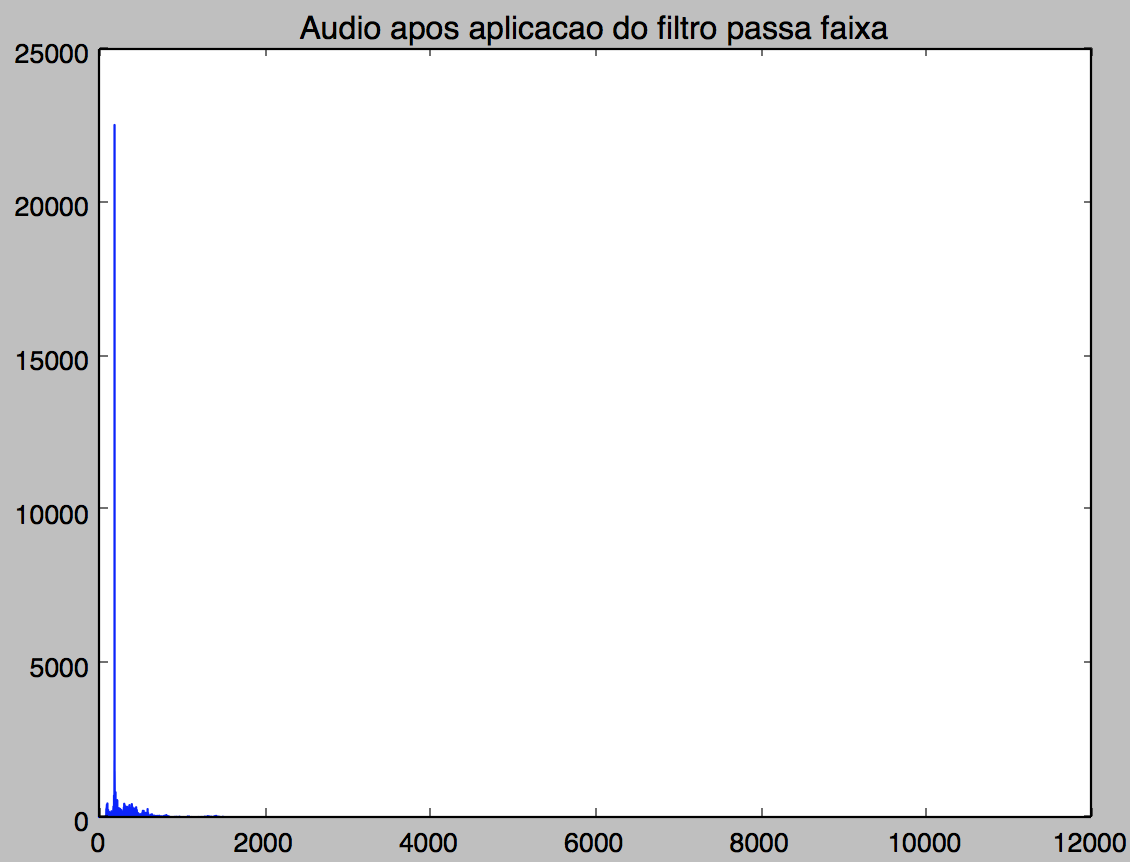


Figura 16. Gráfico do áudio na frequência após a aplicação do filtro passa faixa

Em seguida, era preciso obter um filtro notch FIR para eliminar a frequência de 200hz. Infelizmente, a biblioteca Scipy também não oferece uma forma prática de se obter um filtro notch FIR. A solução encontrada foi calcular previamente os coeficientes para se aplicar no filtro notch FIR, funcionando assim similar a uma janela retangular. O código usado para esse cálculo foi escrito em Scilab e encontra-se no Anexo B.

|  |
| --- |
| Código do filtro FIR notch |
| *# criar filtro*  notch\_b = [  0.0070497, 0.0075856, 0.0081005, 0.0085928, 0.0090606, 0.0095021, 0.0099157, 0.0103000, 0.0106533, 0.0109744,  0.0112619, 0.0115146, 0.0117314, 0.0119115, 0.0120538, 0.0121577, 0.0122225, 0.0122477, 0.0122331, 0.0121782,  0.0120830, 0.0119475 , 0.0117718, 0.0115563, 0.0113013, 0.0110074, 0.0106752, 0.0103056, 0.0098995, 0.0094580,  0.0089821, 0.0084733, 0.0079330 , 0.0073627, 0.0067640, 0.0061387, 0.0054887, 0.0048158, 0.0041222, 0.0034099,  0.0026812, 0.0019383, 0.0011835, 0.0004193 , - 0.0003520, - 0.0011279, - 0.0019058, - 0.0026832, - 0.0034577,  - 0.0042265, - 0.0049872, - 0.0057372, - 0.0064739, - 0.0071949, - 0.0078977, - 0.0085798, - 0.0092388,  - 0.0098725, - 0.0104785, - 0.0110547, - 0.0115989, - 0.0121092, - 0.0125836, - 0.0130202, - 0.0134174,  - 0.0137736, - 0.0140873, - 0.0143572, - 0.0145821, - 0.0147608, 0.9925815, - 0.0147608, - 0.0145821,  - 0.0143572, - 0.0140873, - 0.0137736, - 0.0134174, - 0.0130202, - 0.0125836, - 0.0121092, - 0.0115989,  - 0.0110547, - 0.0104785, - 0.0098725, - 0.0092388, - 0.0085798, - 0.0078977, - 0.0071949, - 0.0064739,  - 0.0057372, - 0.0049872, - 0.0042265, - 0.0034577, - 0.0026832, - 0.0019058, - 0.0011279, - 0.0003520,  0.0004193 , 0.0011835, 0.0019383, 0.0026812, 0.0034099, 0.0041222, 0.0048158, 0.0054887, 0.0061387,  0.0067640, 0.0073627, 0.0079330 , 0.0084733, 0.0089821, 0.0094580, 0.0098995, 0.0103056, 0.0106752, 0.0110074,  0.0113013, 0.0115563, 0.0117718, 0.0119475 , 0.0120830, 0.0121782, 0.0122331, 0.0122477, 0.0122225, 0.0121577,  0.0120538, 0.0119115, 0.0117314, 0.0115146, 0.0112619, 0.0109744, 0.0106533, 0.0103000, 0.0099157, 0.0095021,  0.0090606, 0.0085928, 0.0081005, 0.0075856, 0.0070497  ]  *# exibir graficos do filtro*  plot\_filter\_frequency(notch\_b, [1], fs, title=**'Filtro FIR notch na frequencia'**)  plot\_filter\_fir\_time(notch\_b, title=**'Filtro FIR notch no tempo'**)  *# aplicar filtro*  wave\_audio\_filtered = signal.lfilter(notch\_b, [1], wave\_audio\_filtered)  *# exibir grafico do audio apos a aplicacao do filtro*  plot\_frequency(wave\_audio\_filtered, fs, frequency\_axis=frequency\_axis, title=**'Audio apos aplicacao do filtro notch'**) |

Tabela 7. Código do filtro notch

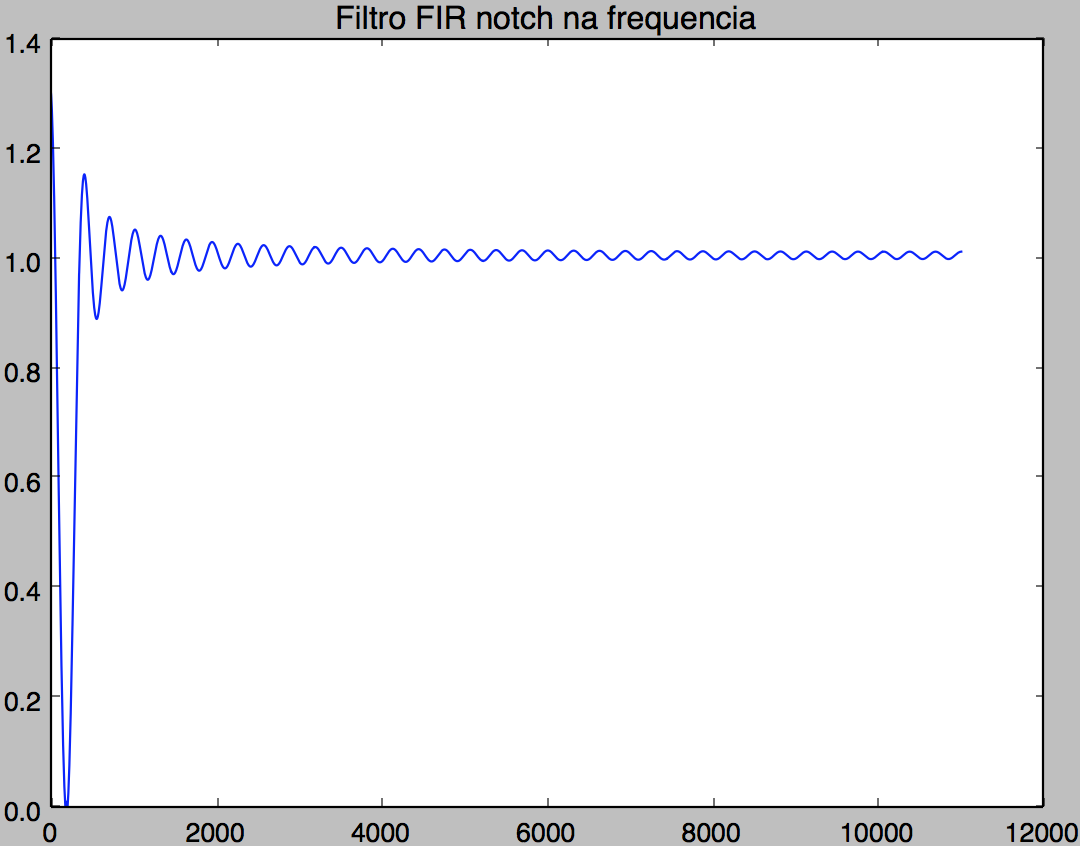


Figura 17. Gráfico na frequência do filtro FIR notch

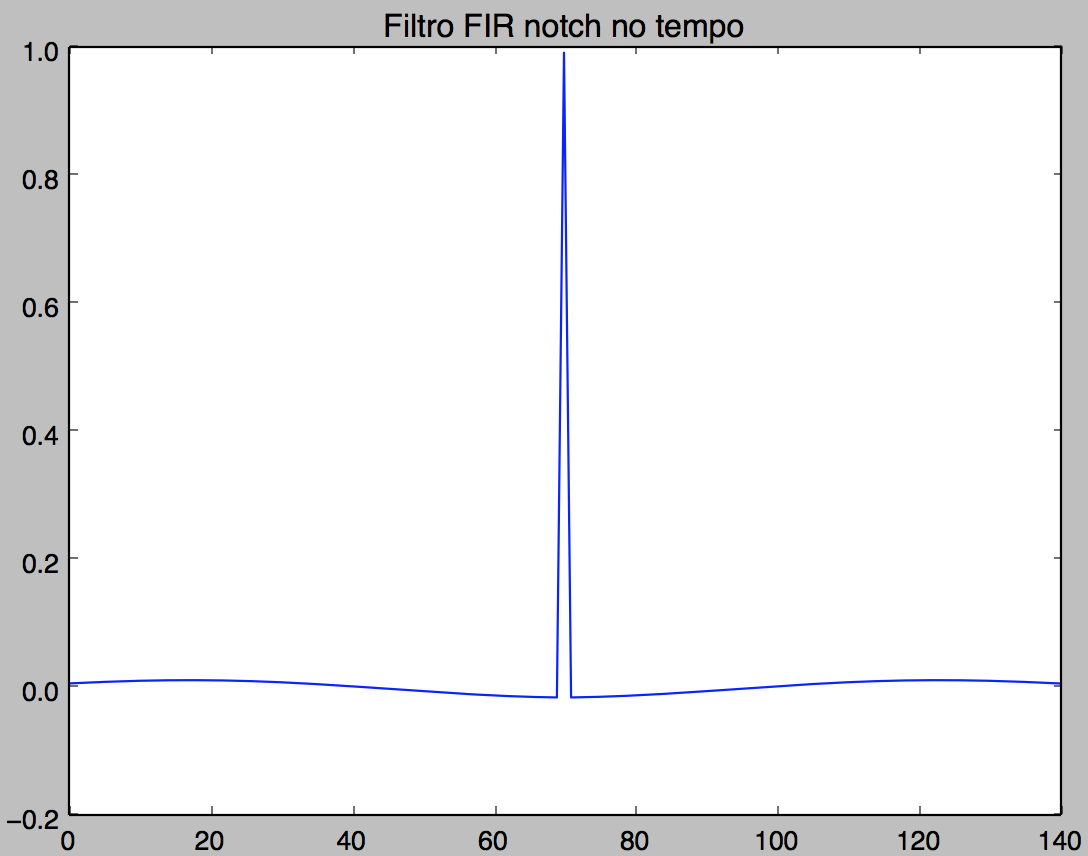


Figura 18. Gráfico de resposta ao impulso do filtro FIR notch

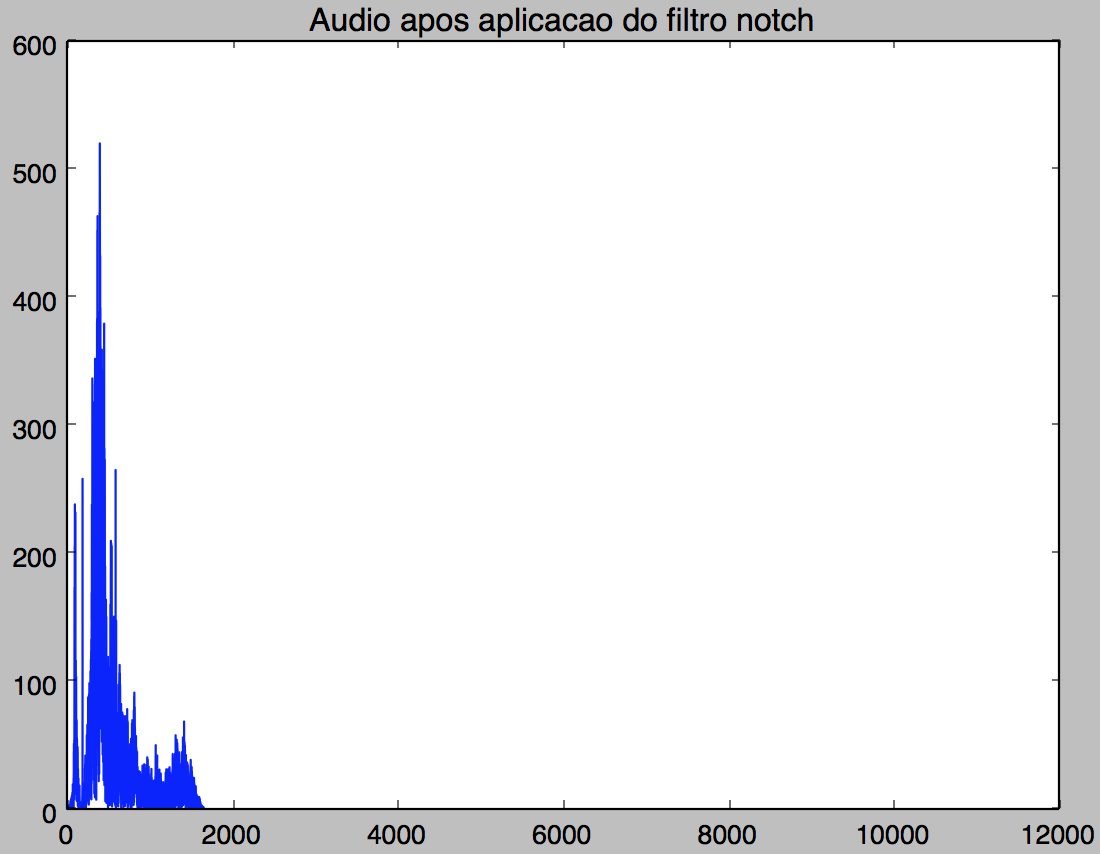


Figura 19. Gráfico do áudio na frequência após a aplicação do filtro notch FIR

Como pode-se observar na Figura 19, o filtro notch FIR desenvolvido apresenta várias oscilações na banda de passagem e o afunilamento para a banda de rejeição é muito mais devagar que a versão equivalente em IIR. De qualquer modo, a Figura 19 mostra que conseguiu atenuar drasticamente a frequência de 200hz.

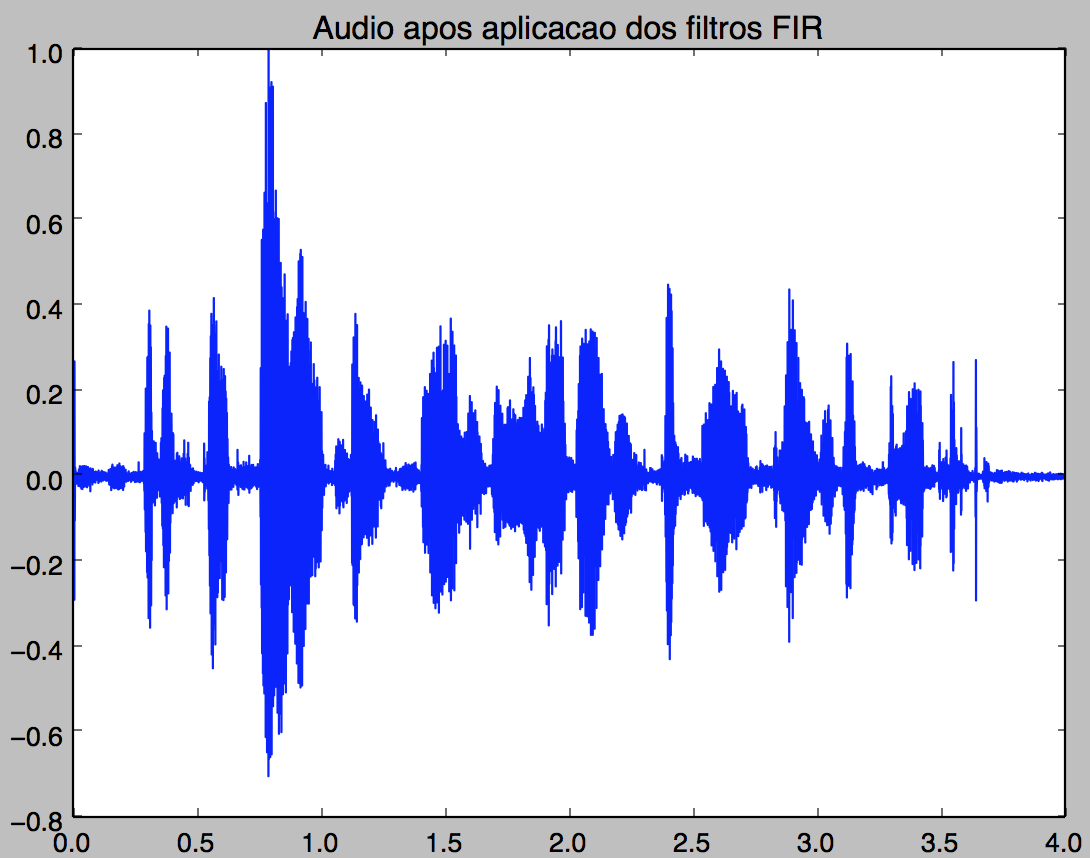


Figura 20. Áudio no domínio do tempo após a aplicação dos filtros FIR.

Na Figura 20 pode-se ver que conseguiu-se o efeito pretendido: as interferências foram bem reduzidas. O gráfico é característico de uma pessoa falando.

# 

# 

# Resultados

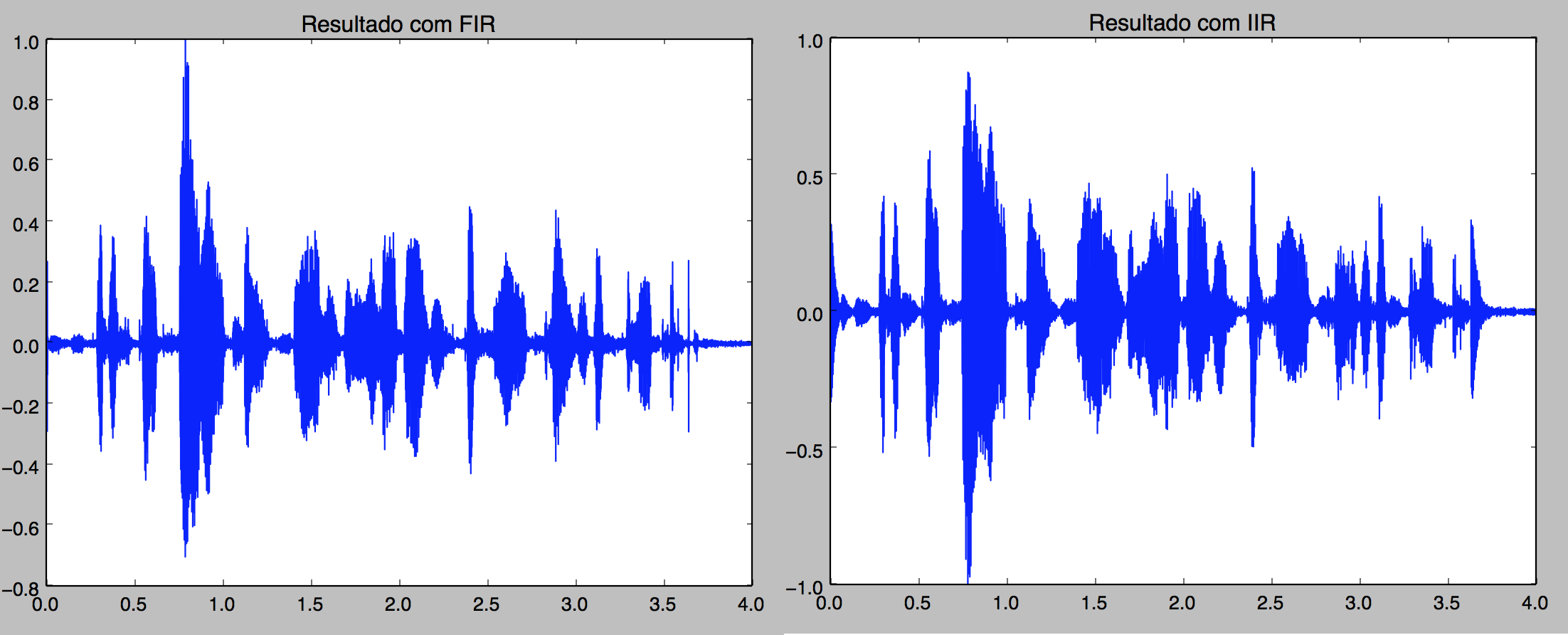


Figura 19. Comparativo no domínio do tempo com o resultado obtido entre filtros FIR (esquerda) e filtros IIR (direita)

Na Figura 19 pode-se comparar os resultados obtidos com as diferentes famílias de filtros. Apesar de algumas diferenças, em boa parte os áudio ficaram parecidos. Escutando, a versão filtrada com FIR apresenta menos interferência na voz, porém, em ambos é possível escutar e entender o que a pessoa diz.

A nível de código, para esse problema em específico, o desenvolvimento dos filtros IIR no Python usando a biblioteca Scipy foi muito mais fácil do que desenvolver os equivalentes em FIR. O primeiro filtro IIR precisou, basicamente, de apenas uma chamada de função, enquanto o equivalente FIR foram necessárias duas chamadas e ainda fazer um truque mudando os valores da array. No caso do filtro notch, para IIR bastou chamar uma função na biblioteca, enquanto no caso do FIR foi necessário calcular previamente os coeficientes e usá-los de forma literal no código.

# Conclusão

Obteve-se sucesso na proposta do trabalho, uma vez que foi desenvolvida as duas versões do código, uma apenas com filtros IIR, e outra apenas com filtros FIR, e em ambos casos a voz do áudio filtrado passou a ser compreensível.

# Anexos

## Anexo A

|  |
| --- |
| misc.py |
| **import** matplotlib.pyplot **as** plt  **from** scipy.fftpack **import** fft  **import** numpy **as** np  **import** wave  **import** scipy.signal **as** signal  *# abre um arquivo de audio do tipo wav e retorna a frequencia de amostragem dele e o audio normalizado*  *# parametro file\_path: caminho do arquivo a ser aberto*  *# retorno: tupla com (fs, audio normalizado)*  **def** open\_wav(file\_path):  *# open file*  wave\_file = wave.open(file\_path, **'rb'**)  *# get frames*  wave\_audio = wave\_file.readframes(-1)  wave\_audio = np.fromstring(wave\_audio, np.int16).astype(float)  fs = wave\_file.getframerate()  *# normalize*  wave\_audio /= np.max(np.abs(wave\_audio))  *#*  **return** (fs, wave\_audio)  *# plota grafico do audio no tempo*  *# parametro wave\_audio: audio a ser plotado*  *# parametro fs: frequencia de amostragem*  *# retorno: nenhum*  **def** plot\_audio\_time(wave\_audio, fs, title=**''**):  time = np.linspace(0, len(wave\_audio) / fs, num=len(wave\_audio))  plt.title(title)  plt.plot(time, wave\_audio)  plt.show()  *# plotar filtro IIR no dominio da frequencia*  *# parametro a: numerador do filtro*  *# parametro b: denominador do filtro*  *# parametro fs: frequencia de amostragem*  *# retorno: nenhum*  **def** plot\_filter\_frequency(b, a, fs, title=**''**):  w, h = signal.freqz(b, a, worN=1024 \* 8)  freq = w \* fs / (2 \* np.pi)  plt.title(title)  plt.plot(freq, abs(h))  plt.show()  *# plota o grafico do audio na frequencia*  *# parametro target: audio ou filtro a ser plotado*  *# parametro fs: frequencia de amostragem*  *# retorno: nenhum*  **def** plot\_frequency(target, fs, frequency\_axis=None, title=**''**):  c = fft(target)  d = len(c)  *# normalizando eixo de frequencia, se precisar*  **if** frequency\_axis **is** None:  frequency\_axis = np.arange(0, len(c)).astype(float)  frequency\_axis /= d  frequency\_axis \*= fs  frequency\_axis = frequency\_axis[:len(frequency\_axis) / 2]  *#*  plt.title(title)  plt.plot(frequency\_axis, abs(c[:len(c) / 2]))  plt.show()  *# plotar o filtro FIR no tempo*  *# parametro filter: filtro a ser plotado*  *# retorno: nenhum*  **def** plot\_filter\_fir\_time(filter, title=**''**):  plt.title(title)  plt.plot(filter)  plt.show()  *# plotar o filtro FIR no tempo*  *# parametro filter: filtro a ser plotado*  *# retorno: nenhum*  **def** plot\_filter\_iir\_time(b, a, sampling\_length, title=**''**):  filter = signal.lfilter(b, a, [1] + [0 **for** i **in** range(sampling\_length)])  plt.title(title)  plt.plot(filter)  plt.show()  **def** normalize\_frequency\_axis(target, fs):  c = fft(target)  d = len(c)  f = np.arange(0, len(c)).astype(float)  f /= d  f \*= fs  **return** f |

|  |
| --- |
| filter\_iir.py |
| **from** misc **import** \*  **import** numpy **as** np  **import** scipy.signal **as** signal  **from** scipy.io.wavfile **import** write  fs, wave\_audio = open\_wav(**'/Users/macabeus/Desktop/pds/fala\_sirene\_tm0.wav'**)  plot\_audio\_time(wave\_audio, fs, title=**'Audio original no tempo'**)  frequency\_axis = normalize\_frequency\_axis(wave\_audio, fs)  frequency\_axis = frequency\_axis[:len(frequency\_axis) / 2]  plot\_frequency(wave\_audio, fs, frequency\_axis=frequency\_axis, title=**'Audio original na frequencia'**)  *###*  *# filtrar com passa baixa*  *# criar filtro*  filter\_order = 5  b, a = signal.iirfilter(filter\_order, [200.0/(fs/2), 1000.0/(fs/2)], ftype=**'butter'**, output=**'ba'**)  *# exibir graficos do filtro*  plot\_filter\_frequency(b, a, fs, title=**'Filtro IIR passa faixa na frequencia'**)  plot\_filter\_iir\_time(b, a, 300, title=**'Filtro IIR passa faixa no tempo'**)  *# aplicar filtro*  wave\_audio\_filtered = signal.lfilter(b, a, wave\_audio)  *# exibir grafico do audio apos a aplicacao do filtro*  plot\_frequency(wave\_audio\_filtered, fs, frequency\_axis=frequency\_axis, title=**'Audio apos aplicacao do filtro passa baixa'**)  *###*  *# filtrar com notch*  *# criar filtro notch*  f0 = 200.0 *# frequencia a ser removida no sinal*  q = 20.0 *# fator de qualidade*  w0 = f0/(fs/2) *# normalizar a frequencia*  b, a = signal.iirnotch(w0, q)  *# exibir grafico do filtro no dominio da frequencia*  plot\_filter\_frequency(b, a, fs, title=**'Filtro IIR notch na frequencia'**)  plot\_filter\_iir\_time(b, a, 100, title=**'Filtro IIR notch no tempo'**)  *# aplicar filtro*  wave\_audio\_filtered = signal.lfilter(b, a, wave\_audio\_filtered)  *# exibir grafico do audio apos a aplicacao do filtro*  plot\_frequency(wave\_audio\_filtered, fs, frequency\_axis=frequency\_axis, title=**'Audio apos aplicacao do filtro notch'**)  *###*  *# resultado*  *# salvar*  wave\_audio\_filtered /= np.max(np.abs(wave\_audio\_filtered))  write(**'filtered\_iir.wav'**, fs, wave\_audio\_filtered)  *# grafico no tempo*  plot\_audio\_time(wave\_audio\_filtered, fs, title=**'Audio apos aplicacao dos filtros IIR'**) |

|  |
| --- |
| filter\_fir.py |
| **from** scipy.io.wavfile **import** write  **from** misc **import** \*  fs, wave\_audio = open\_wav(**'/Users/macabeus/Desktop/pds/fala\_sirene\_tm0.wav'**)  plot\_audio\_time(wave\_audio, fs, title=**'Audio original no tempo'**)  frequency\_axis = normalize\_frequency\_axis(wave\_audio, fs)  frequency\_axis = frequency\_axis[:len(frequency\_axis) / 2]  plot\_frequency(wave\_audio, fs, frequency\_axis=frequency\_axis, title=**'Audio original na frequencia'**)  *###*  *# filtrar com passa baixa*  *# criar filtro*  n = 401  a = signal.firwin(n, cutoff=200, window=**'blackmanharris'**, nyq=fs)  b = - signal.firwin(n, cutoff=3100, window=**'blackmanharris'**, nyq=fs)  b[n/2] = b[n/2] + 1  df = - (a+b)  df[n/2] = df[n/2] + 1  *# exibir graficos do filtro*  plot\_frequency(df, fs, title=**'Filtro FIR passa faixa na frequencia'**)  plot\_filter\_fir\_time(df, title=**'Filtro FIR passa faixa no tempo'**)  *# aplicar filtro*  wave\_audio\_filtered = signal.lfilter(df, [1], wave\_audio)  *# exibir grafico do audio apos a aplicacao do filtro*  plot\_frequency(wave\_audio\_filtered, fs, frequency\_axis=frequency\_axis, title=**'Audio apos aplicacao do filtro passa faixa'**)  *###*  *# filtrar com notch*  *# criar filtro*  notch\_b = [  0.0070497, 0.0075856, 0.0081005, 0.0085928, 0.0090606, 0.0095021, 0.0099157, 0.0103000, 0.0106533, 0.0109744,  0.0112619, 0.0115146, 0.0117314, 0.0119115, 0.0120538, 0.0121577, 0.0122225, 0.0122477, 0.0122331, 0.0121782,  0.0120830, 0.0119475 , 0.0117718, 0.0115563, 0.0113013, 0.0110074, 0.0106752, 0.0103056, 0.0098995, 0.0094580,  0.0089821, 0.0084733, 0.0079330 , 0.0073627, 0.0067640, 0.0061387, 0.0054887, 0.0048158, 0.0041222, 0.0034099,  0.0026812, 0.0019383, 0.0011835, 0.0004193 , - 0.0003520, - 0.0011279, - 0.0019058, - 0.0026832, - 0.0034577,  - 0.0042265, - 0.0049872, - 0.0057372, - 0.0064739, - 0.0071949, - 0.0078977, - 0.0085798, - 0.0092388,  - 0.0098725, - 0.0104785, - 0.0110547, - 0.0115989, - 0.0121092, - 0.0125836, - 0.0130202, - 0.0134174,  - 0.0137736, - 0.0140873, - 0.0143572, - 0.0145821, - 0.0147608, 0.9925815, - 0.0147608, - 0.0145821,  - 0.0143572, - 0.0140873, - 0.0137736, - 0.0134174, - 0.0130202, - 0.0125836, - 0.0121092, - 0.0115989,  - 0.0110547, - 0.0104785, - 0.0098725, - 0.0092388, - 0.0085798, - 0.0078977, - 0.0071949, - 0.0064739,  - 0.0057372, - 0.0049872, - 0.0042265, - 0.0034577, - 0.0026832, - 0.0019058, - 0.0011279, - 0.0003520,  0.0004193 , 0.0011835, 0.0019383, 0.0026812, 0.0034099, 0.0041222, 0.0048158, 0.0054887, 0.0061387,  0.0067640, 0.0073627, 0.0079330 , 0.0084733, 0.0089821, 0.0094580, 0.0098995, 0.0103056, 0.0106752, 0.0110074,  0.0113013, 0.0115563, 0.0117718, 0.0119475 , 0.0120830, 0.0121782, 0.0122331, 0.0122477, 0.0122225, 0.0121577,  0.0120538, 0.0119115, 0.0117314, 0.0115146, 0.0112619, 0.0109744, 0.0106533, 0.0103000, 0.0099157, 0.0095021,  0.0090606, 0.0085928, 0.0081005, 0.0075856, 0.0070497  ]  *# exibir graficos do filtro*  plot\_filter\_frequency(notch\_b, [1], fs, title=**'Filtro FIR notch na frequencia'**)  plot\_filter\_fir\_time(notch\_b, title=**'Filtro FIR notch no tempo'**)  *# aplicar filtro*  wave\_audio\_filtered = signal.lfilter(notch\_b, [1], wave\_audio\_filtered)  *# exibir grafico do audio apos a aplicacao do filtro*  plot\_frequency(wave\_audio\_filtered, fs, frequency\_axis=frequency\_axis, title=**'Audio apos aplicacao do filtro notch'**)  *###*  *# resultado*  *# salvar*  wave\_audio\_filtered /= np.max(np.abs(wave\_audio\_filtered))  write(**'filtered\_fir.wav'**, fs, wave\_audio\_filtered)  *# grafico no tempo*  plot\_audio\_time(wave\_audio\_filtered, fs, title=**'Audio apos aplicacao dos filtros FIR'**) |

## Anexo B

|  |
| --- |
| coefficient.sci |
| fa = 22050; *// frequência de amostragem* fa2 = 22050/2;  */// Projeto de filtro IIR usando o comando iir do Scilab:* hz=iir(4,'sb','butt',[190 210]/fa,[.08 .03]); *// filtro de ordem 4, rejeita faixa* q=poly(0,'q'); *// coeficientes em termos de função de 'q'* hzd=horner(hz,1/q); *// função no formato H(q) = N(q)/D(q)* disp(hzd); *// mostrando H(q)*  *// Coeficientes obtidos de hzd - filtro IIR:* N = [0.9999432, -2.0041426, 1.007474]; D = [1.0074167, -2.00414, 1];  *// Resposta ao impulso:* x = zeros(1,125); x(1) = 1; *// impulso unitário* h = filter(N,D,x); *// resposta ao impulso* figure; plot(h); *// gráfico da resposta ao impulso*  *//Gráfico da resposta em frequência do filtro FIR:* hn = [h(71:-1:2), h(1), h(2:71)]; *// filtro simétrico* disp(hn'); *// mostrando os coeficientes do filtro no tempo* [hzm,fr]=frmag(hn,1,1024); *// calculo da resp. em freq.* figure; plot2d(fa\*fr',hzm'); *// gráfico da resp. em freq.* |

# Bibliografia

[1] [M. V. Paolo Prandoni, *Signal Processing For Communications*. .](http://paperpile.com/b/cTtsKG/7XLV)

[2] [M. V. Paolo Prandoni, *Signal Processing For Communications*. 2008.](http://paperpile.com/b/cTtsKG/bfj3)

[3] [A. Device, “A Beginner’s Guide to Digital Signal Processing (DSP)”, *Analog Device*. [Online]. Disponível em:](http://paperpile.com/b/cTtsKG/MbmQ) <http://www.analog.com/en/design-center/landing-pages/001/beginners-guide-to-dsp.html>[. [Acessado: 10-jul-2017]](http://paperpile.com/b/cTtsKG/MbmQ)

[4] [“A Beginner’s Guide to Digital Signal Processing (DSP) | Design Center | Analog Devices”. [Online]. Disponível em:](http://paperpile.com/b/cTtsKG/L88T) <http://www.analog.com/en/design-center/landing-pages/001/beginners-guide-to-dsp.html>[. [Acessado: 10-jul-2017]](http://paperpile.com/b/cTtsKG/L88T)

[5] [E. R. Barbosa, “Filtros digitais reconfiguráveis”, 2006 [Online]. Disponível em:](http://paperpile.com/b/cTtsKG/CvZ0) <http://www.up.edu.br/blogs/engenharia-da-computacao/wp-content/uploads/sites/6/2015/06/2006.8.pdf>[. [Acessado: 10-jul-2017]](http://paperpile.com/b/cTtsKG/CvZ0)

[6] [“High pass filter - Wikimedia Commons”. [Online]. Disponível em:](http://paperpile.com/b/cTtsKG/yT7W) <https://commons.wikimedia.org/wiki/File:High_pass_filter.svg>[. [Acessado: 14-jul-2017]](http://paperpile.com/b/cTtsKG/yT7W)

[7] [“RC Divider from Wikimedia Commons”. [Online]. Disponível em:](http://paperpile.com/b/cTtsKG/Ht2Z) <https://commons.wikimedia.org/wiki/File:RC_Divider.svg>[. [Acessado: 13-jul-2017]](http://paperpile.com/b/cTtsKG/Ht2Z)

[8] [“Bandpass Filter - Wikipedia”. [Online]. Disponível em:](http://paperpile.com/b/cTtsKG/i45V) <https://en.wikipedia.org/wiki/File:Bandpass_Filter.svg>[. [Acessado: 14-jul-2017]](http://paperpile.com/b/cTtsKG/i45V)

[9] [“Band-Reject Filter - Wikimedia Commons”. [Online]. Disponível em:](http://paperpile.com/b/cTtsKG/avbr) <https://commons.wikimedia.org/wiki/File:Band-Reject_Filter.svg>[. [Acessado: 14-jul-2017]](http://paperpile.com/b/cTtsKG/avbr)

[10] [M. Pastell, “FIR filter design with Python and SciPy”. [Online]. Disponível em:](http://paperpile.com/b/cTtsKG/TAeH) <http://mpastell.com/2010/01/18/fir-with-scipy/>[. [Acessado: 14-jul-2017]](http://paperpile.com/b/cTtsKG/TAeH)