**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL**
**ESCOLA POLITÉCNICA**
**ENGENHARIA DE COMPUTAÇÃO**

# EQUALIZADOR PARAMÉTRICO DIGITAL EM TEMPO REAL

**EDUARDO VIANA PEREIRA**

Monografia apresentada como requisito parcial à obtenção do grau de Engenheiro de Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Dênis Fernandes

**Porto Alegre**
**2021**

# EQUALIZADOR PARAMÉTRICO DIGITAL EM TEMPO REAL


## RESUMO

Este trabalho apresenta o desenvolvimento de um equalizador paramétrico em tempo real, implementado em python e embarcado em uma *Raspberry Pi 3*, que possui uma CPU Quad Core 1.2GHz Broadcom BCM2837 64bit. A diferença de um equalizador paramétrico para os outros tipos, é que o usuário pode ajustar o valor da frequência que deseja filtrar, não possuindo valores de frequências predefinidas. Nesse contexto, esse trabalho busca desenvolver um equalizador paramétrico de boa qualidade, com uma boa resposta em tempo real, e com um baixo valor de mercado, utilizando um "computador" de baixo custo para executar a aplicação.



**Palavras-Chave:** Equalizador, DSP, Áudio.

## AGRADECIMENTOS

Agradeço ao professor Dênis Fernandes, por toda ajuda e conhecimento que me forneceu para o desenvolvimento deste trabalho, sempre me apoiando e me direcionando para melhorar cada vez mais.

Agradeço aos professores Júlio Cesar e Renan Viero por participarem da banca e por toda ajuda disponibilizada tanto neste trabalho quanto em toda minha carreira acadêmica e profissional.

Agradeço aos meus amigos e ex-colegas, Eduardo Brum, Hêndrick Gonçalves, Clodoaldo Borba, Rebeca Gomes, e em especial ao meu amigo Matheus Ferronato por toda ajuda, parcerias em trabalhos e pelas melhores dicas de *Python*, o *Dream Team* virá.

Agradeço ao professor Marlon Moraes que me indicou para o tão sonhado emprego na DATACOM que estou hoje, e ao meu gestor Stein pela confiança e por acreditar no meu trabalho.

Agradeço a minha mãe Rosania Viana por toda ajuda e apoio durante esses 6 anos de curso, ao meu pai, Rogério Pereira que também é engenheiro e sempre me apoio e me ajudou durante toda a faculdade, e toda minha família por toda força e apoio.

Agradeço aos meus amigos, Luka de Lima, Pedro Rembold, Thailon Dorneles e Nicolas Coelho, por me tirarem da bolha de estudo quando eu precisava.

E por fim agradeço a minha namorada Alexandra Suchy que sempre me incentivou para realização dos meus trabalhos nesse final de curso.

“This is one **small step for a man**, one **giant leap for mankind.**”
(Neil Armstrong)

# SUMÁRIO

# LISTA DE FIGURAS

# LISTA DE SIGLAS

CPU – *Central Processing Unit*

DSP – *Digital Signal Processor*

FIR – *Finite Impulse Response*

IIR – *Infinite Impulse Response*

DB – *Decibéis*

AD – *Analógico-digital*

DA – *Digital-analógico*

USB – *Universal Serial Bus*

MIC – *Microfone*

RCA – *Radio Corporation of America (nome da empresa que desenvolveu o cabo)*

MIX – *Mistura*

RJ45 – *Registered jack 45*

HDMI – *High-Definition Multimedia Interface*

SD – *Secure Digital Card*

RAM – *Random Access Memory*

LAN – *Local Area Network*

GPIO – *General Purpose Input/Output*

C – *Linguagem de programação*

# 1. INTRODUÇÃO

Um equalizador é um sistema utilizado no ramo musical para modificar a resposta em frequência de um determinado som, ou seja, com um equalizador é possível que apenas um instrumento em específico de uma determinada música seja removido ou tenha o volume reduzido ou amplificado. Existem inúmeras categorias de equalizadores, os principais são os gráficos e os paramétricos. Os gráficos são equalizadores que já possuem frequências fixas para o usuário amplificar ou atenuar, assim sendo limitado o uso as frequências impostas por este equipamento. Os paramétricos que será o tema abordado neste trabalho, são equalizadores onde o usuário pode escolher qual a frequência que deseja atenuar ou amplificar, escolhendo o valor do ganho, negativo ou peositivo, e também a largura da banda da frequência escolhida, podendo ser um filtro de pico, filtrando apenas a frequência desejada ou algo como um rejeita ou passa banda.

Um equalizador pode ser analógico ou digital. Neste projeto o equalizador será digital, pois sistemas de filtros digitais tem inúmeras vantagens comparado com os analógicos, sendo principalmente pela facilidade em trabalhar com filtros de ordem mais altas, [Han].

## 1.1 Motivação

A motivação para a escolha do tema deste projeto foi primeiramente o amor pela música, e também por já ser músico e conhecer as principais dificuldades hoje encontradas neste ramo relacionado a este assunto. Uma parte da motivação também veio por conversar com pessoas que trabalham no meio musical, e ver que hoje em dia ainda se tem muita dificuldade para encontrar equipamentos de boa qualidade com um preço acessível, atendendo todas as demandas no meio, uma delas a resposta em tempo real, que foi o principal objetivo a ser alcançado neste trabalho.

## 1.2 Objetivos

O objetivo deste trabalho foi implementar um equalizador paramétrico com uma boa qualidade de áudio, com uma resposta em tempo real, e um baixo custo de produção. Para estes objetivos serem alcançados, foram usados filtros do tipo FIR, que possuem um melhor resultado de fase comparados com os filtros IIR, porém possuindo enorme complexidade, com ordens elevadas, assim dificultando a resposta em tempo real da aplicação. Para conseguir uma resposta em tempo real do filtro, foi escolhido um *Raspberry Pi 3*, que

nada mais é do que um computador de placa única, suportando inúmeros sistemas operacionais, sendo a maioria baseados em *Linux*, com um tamanho parecido a um cartão de crédito, podendo ser conectada a um monitor e conexão *internet*, possuindo inúmeras aplicações. O motivo da escolha deste dispositivo, primeiro o seu baixo custo, em relação as suas funcionalidades e também do seu processador de alta frequência, comparado a microcontroladores, que não conseguiriam entregar a resposta em tempo real utilizando filtros FIR com ordens mais altas.

# 2. TRABALHOS RELACIONADOS

Neste capítulo serão apresentados alguns trabalhos relacionados ao tema proposto.

Foram encontrados alguns trabalhos que se assemelham muito a proposta deste, porém foi encontrado uma grande dificuldade, a relação entre boa qualidade do filtro, e uma resposta em tempo real. Nos inúmeros trabalhos encontrados, foi adotado o uso de filtros IIR para realizar a equalização. A escolha se da por filtros IIR possuir uma ordem menor que filtros FIR com os mesmos parâmetros, assim podendo ser implementados em microcontroladores mais baratos conseguindo obter uma resposta em tempo real.

A seguir podemos ver alguns trabalhos relacionados.

### 2.0.1 Equalizador Paramétrico de Áudio Digital com cinco Bandas, Interface Gráfica *Touch Screen e Operação* em Tempo Real

Este trabalho foi desenvolvido no Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, pelo autor José Nicolau Varela. O trabalho consiste em um equalizador paramétrico com 5 bandas, implementado em um kit de desenvolvimento STM32F746G, com uma interface gráfica e *touch screen*. O processador do kit possui uma frequência de 216 MHZ, assim limitando a ordem do filtro, por este motivo o autor preferiu em usar filtros do tipo IIR, pois possuem uma ordem mais baixa.

Os resultados obtidos neste trabalho foram satisfatórios segundo a proposta, porém o autor relata que enfrentou difilcudade, pois o processamento da interface gráfica desenvolvida gerava ruídos no áudio de saída, assim comprometendo a qualidade e confiabilidade do equalizador, isso acontece pela baixa frequência que o *hardware* escolhido possui. Outra dificuldade que o autor comenta, e coloca como melhorias futuras, é o aumento da ordem dos filtros, pois como dito anteriormente, filtros IIR possuem uma ordem muito baixa, interferindo diretamente na qualidade da equalização, não sendo possível o aumento da ordem pois o microcontrolador não suporta um filtro com uma ordem maior.

A figura 2.1 mostra um filtro de *Shelving*, sendo um filtro usado em equalização que atua ou na região de baixa, ou alta frequência, ou seja, criando uma transição entre uma região extrema, implementado no equalizador do trabalho citado acima.

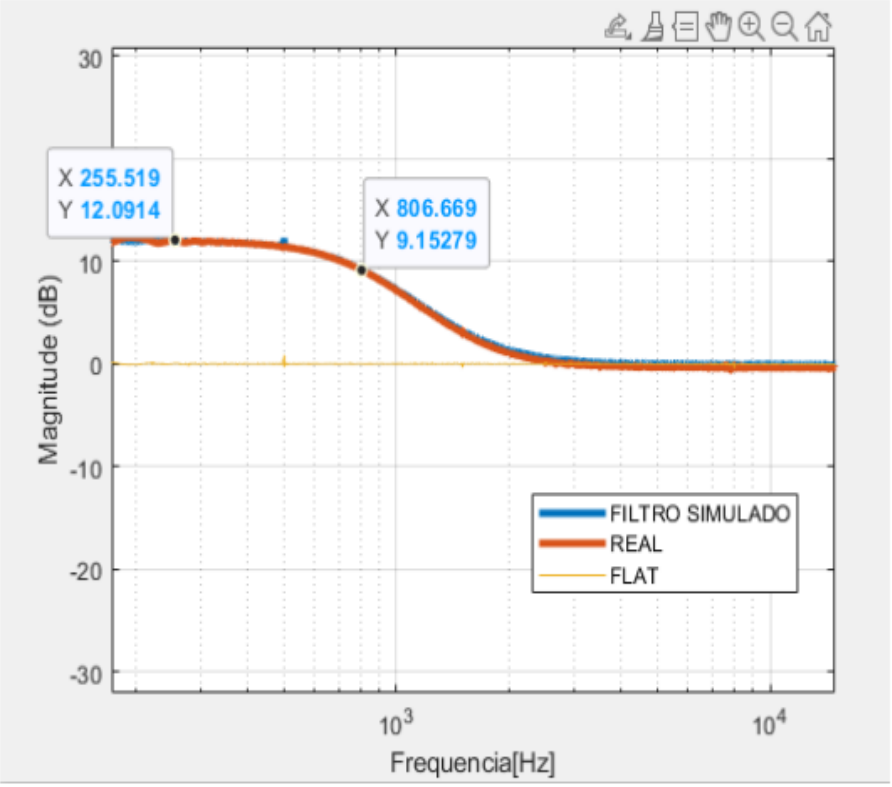



Figura 2.1: Resposta do *low shelving* equalizado e simulado, com G = 12 dB e fc = 800 Hz
Fonte: [VAR21]

### 2.0.2 Sistema para Equalização Paramétrica de Áudio em Tempo Real

O trabalho desenvolvido pelo Matheus Bauer na Universidade Tecnológica Federal do Paraná, consiste em equalizador paramétrico de 5 bandas utilizando filtros IIR, que também serão implementados em um kit de desenvolvimento STM32F746G *Discovery*. O autor relata que a categoria de filtro escolhido se deu devido à baixa ordem do tipo do filtro, em comparação ao FIR. Os resultados do trabalho mostram que o mesmo ficou bem limitado, pois se fez necessários, deixar frequências predefinidas, com os coeficientes pré calculados, pois a resposta em tempo real estava sendo afetada.

Na figura 2.2 podemos ver o equalizador do trabalho mencionado funcionando, com as configurações sendo frequência de corte 3 Khz e um ganho de 2dB. Observamos que a fase está bem atrasada, devido ao uso do filtro IIR. No trabalho proposto também tera uma fase atrasada, porém com uma fase linear, assim não distorcendo o sinal.

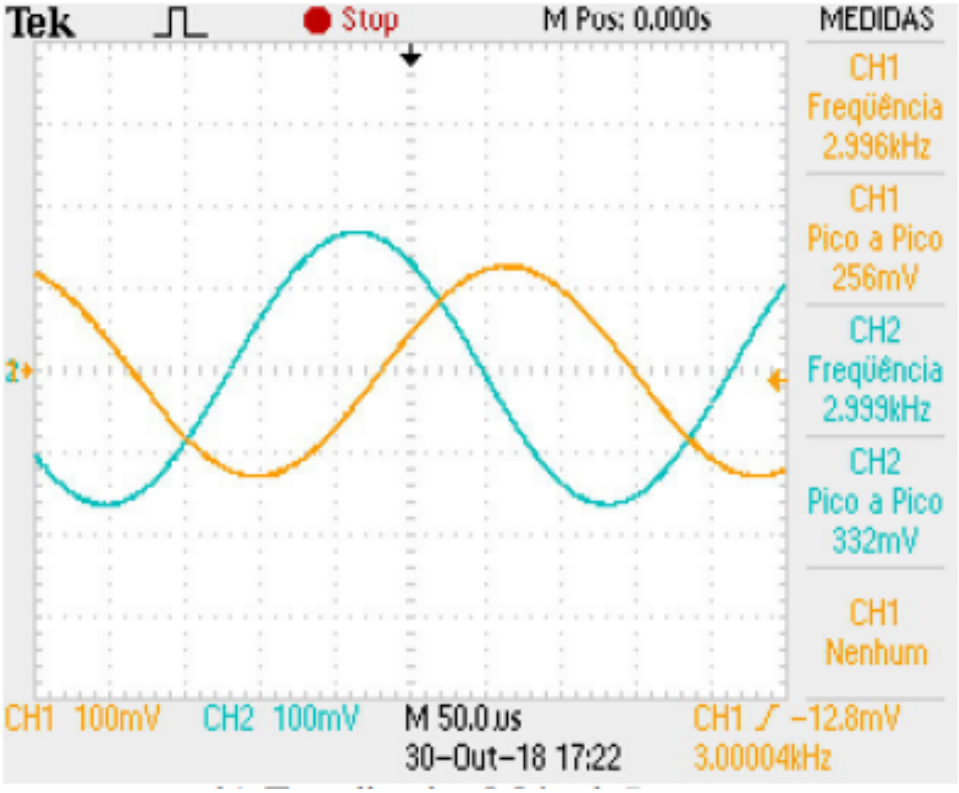



Figura 2.2: Resultado prático filtro *Peak* 3 kHz com ganho de 2dB
Fonte: [BAU18]

### 2.0.3 *DESIGN AND IMPLEMENTATION OF A PARAMETRIC EQUALIZER USING IIR AND FIR FILTERS*

O trabalho foi desenvolvido por Marco Antonio Martínez Ramírez na *University of Manchester*, sendo o trabalho que mais se assemelhou ao projeto em questão. Ele desenvolve um equalizador paramétrico utilizando filtros IIR e FIR a fim de comparar os resultados. O equalizador é implementado apenas no *software* MATLAB, não sendo colocado em nenhum *hardware* externo.

No projeto foi implementado um rejeita-banda utilizando FIR e IIR, e o resultado foi extremamente diferente entre às duas categorias de filtros. Uma das principais diferenças é que o IIR tem uma seletividade limitada, assim todas as bandas selecionadas afetam o espectro inteiro. Outro ponto positivo do FIR foi que ele apresentou uma resposta totalmente plana nas bandas, já no IIR não sendo totalmente plano, isso num equalizador de áudio é muito importante, pois acaba afetando outras bandas que não se tem interesse em mexer.

Observamos na figura 2.3 a diferença da resposta obtida com o filtro FIR, os gráficos da esquerda, em comparação com o IIR, gráficos da direita. Podemos observar que o filtro IIR acaba alterando bastante o espectro, mexendo bastante nas frequências vizinhas,

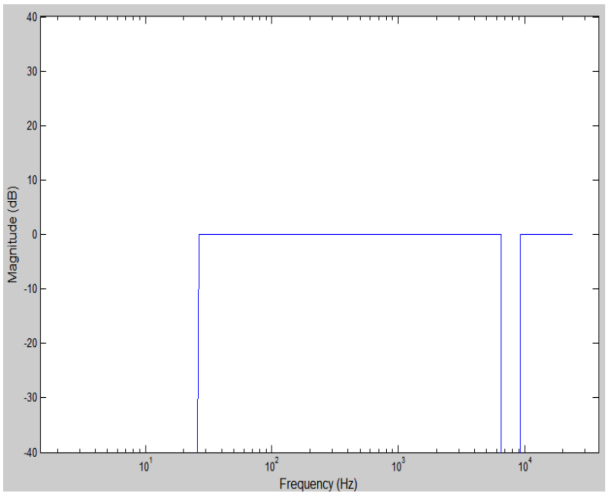
já no FIR vemos uma resposta completamente plano, alterando apenas as frequências desejadas, sem alterar o resto do espectro.

Descrição dos gráficos FIR, rejeita-banda:

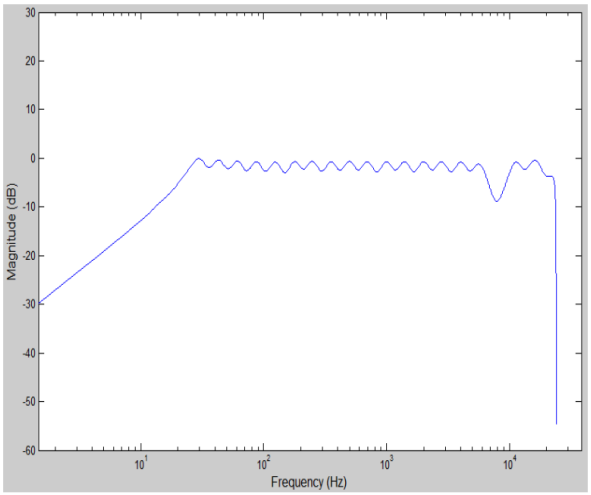
- a - resposta desejada gerada no MATLAB, com frequência central de 8kHz, Q=3 e ganho de 0dB;
- b - resposta desejada gerada no *software Signal Wizard*, utilizando a janela de *Hanning*;
- c - Resultado da FFT após filtrar um sinal de ruído branco;

Descrição dos gráficos IIR, rejeita-banda:

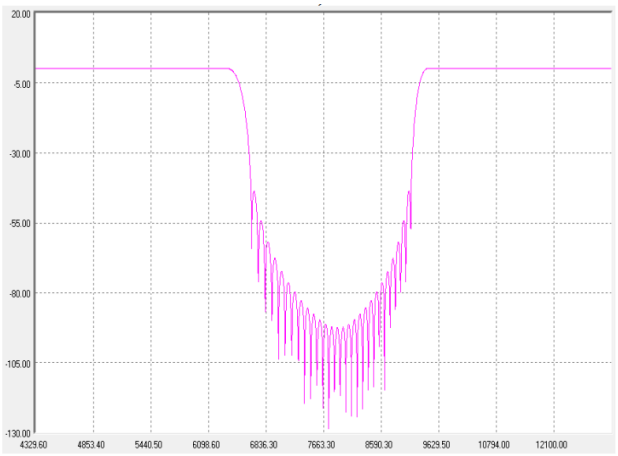
- d - resposta desejada gerada no MATLAB, com frequência central de 8kHz, Q=3 e ganho de 0dB;
- e - resposta desejada gerada no *software Signal Wizard*, utilizando a janela de *Hanning*;
- f - Resultado da FFT após filtrar um sinal de ruído branco;



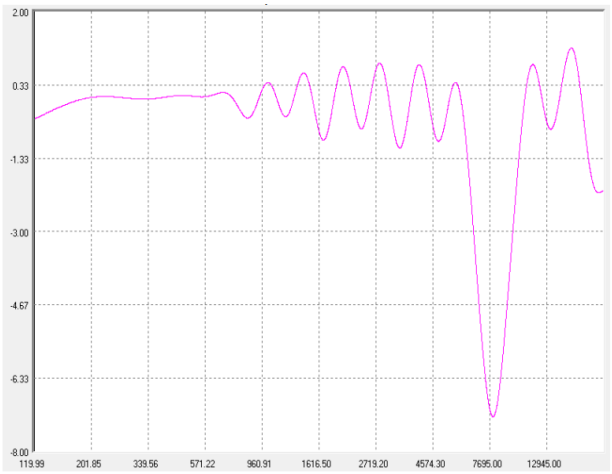

a)



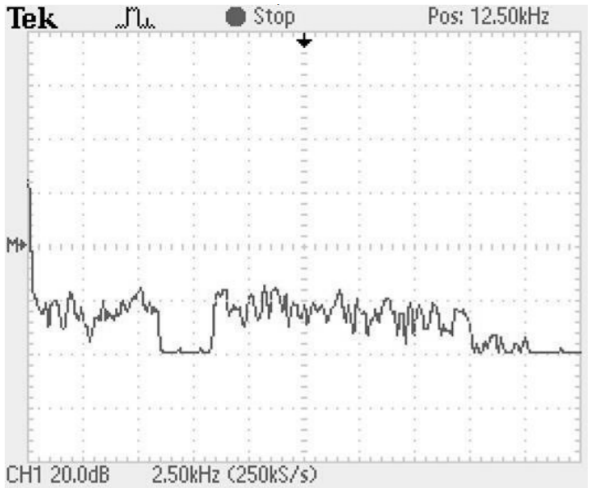

d)



b)



e)




c)


f)

Figura 2.3: Resposta de um rejeita-banda utilizando filtros FIR e IIR
Fonte: [Ram13]

Também foi efetuado um teste de um filtro de pico, utilizando como frequência de corte 6432Hz, onde o filtro FIR também obtém um desempenho maior do que o IIR, como

podemos ver na figura 2.4, onde o filtro IIR acaba prejudicando frequências vizinhas que não deveriam ser alteradas.

Descrição dos gráficos FIR, rejeita-banda:

- a - resposta desejada gerada no MATLAB, com frequência central de 6432Hz, Q=1000 e ganho de 10dB;
- b - resposta desejada gerada no *software Signal Wizard*, utilizando a janela de *Hanning*;
- c - Resultado da FFT após filtrar uma onda senoidal com um ruído branco;

Descrição dos gráficos IIR, rejeita-banda:

- d -resposta desejada gerada no MATLAB, com frequência central de 6432Hz, Q=1000 e ganho de 10dB;
- e - resposta desejada gerada no *software Signal Wizard*, utilizando a janela de *Hanning*;
- f - Resultado da FFT após filtrar uma onda senoidal com um ruído branco;

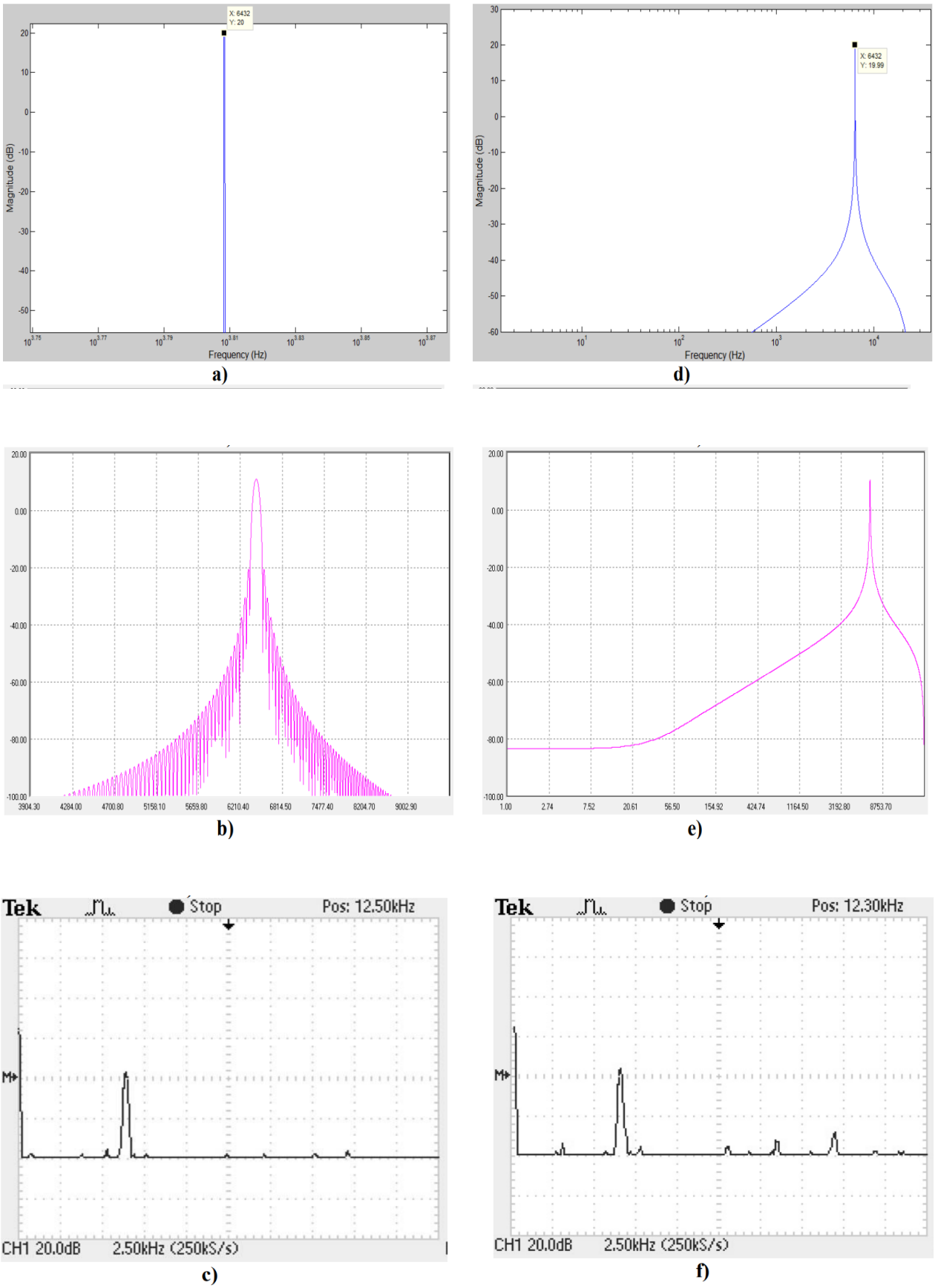



Figura 2.4: Resposta de um filtro de pico utilizando filtros FIR e IIR
Fonte: [Ram13]

# 3. IMPLEMENTAÇÃO DO PROJETO

Neste tópico serão abordadas as etapas do projeto assim como a descrição de algumas tecnologias que foram usadas no desenvolvimento do trabalho.

## 3.1 Etapas

O projeto foi desenvolvido da seguinte forma:

- Desenvolvimento do filtro FIR;
- Implementação do filtro no MATLAB;
- Testes com uma interface de áudio externa no *Raspberry Pi 3*;
- Desenvolvimento do filtro em uma linguagem suportada pela *Raspberry Pi 3*;
- Implementação do equalizador utilizando a interface de áudio com a placa;
- Testes e validação do equalizador paramétrico;

### 3.1.1 Desenvolvimento do Filtro FIR

No desenvolvimento do projeto do filtro, foi feito a soma um filtro rejeita-banda com um passa banda, sendo as frequências de corte iguais para ambos, assim o passa banda sendo multiplicado com o ganho desejado, onde um ganho menor do que 1 e maior que zero sendo uma atenuação, e um ganho maior que 1, uma amplificação. Podemos ver a representação deles, nas figuras 3.1 e 3.2, [Zol95].

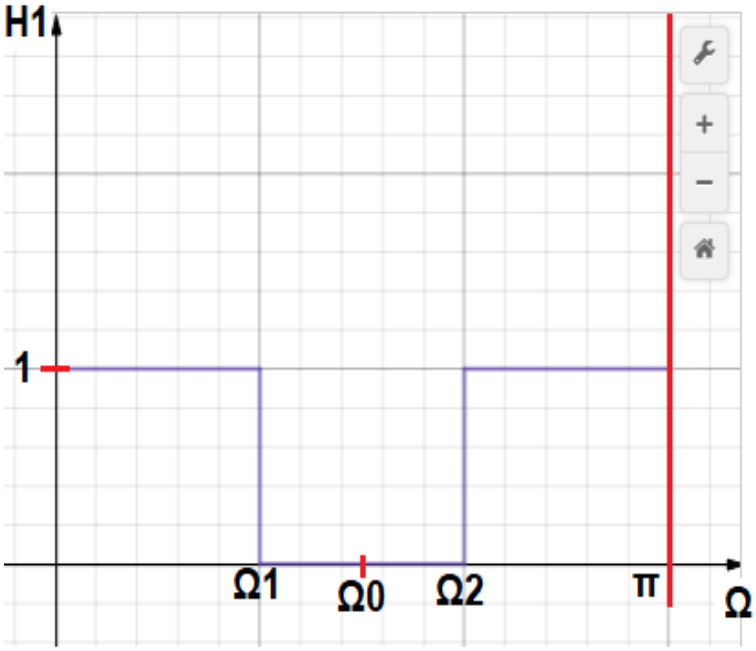



Figura 3.1: Filtro Rejeita Banda
Fonte: Autor

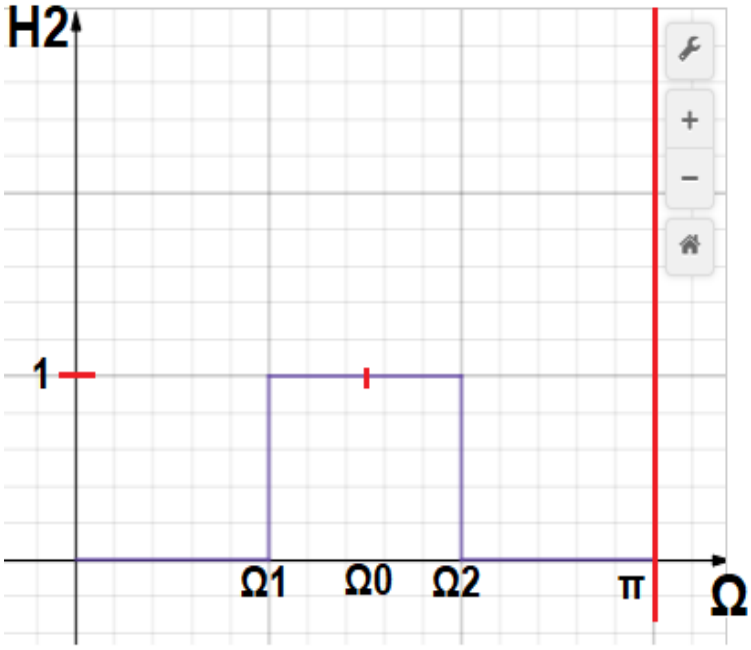



Figura 3.2: Filtro Passa Banda
Fonte: Autor

Assim o filtro resultante sendo:

$$H = H1 + G * H2 \tag{3.1}$$

Onde G é o ganho desejado, quando menor que 1 e maior que zero ele é uma atenuação e quando maior que 1 uma amplificação.

Primeiramente é criado um filtro passa-baixa para fazer o passa-banda, chamado de hlp, onde N é igual à ordem do filtro:

$$hlp = \frac{\sin(\omega c * (n + -\frac{N}{2})}{(n + -\frac{N}{2}) * \pi} \tag{3.2}$$

Com isso podemos calcular o passa-banda e o rejeita-banda, chamados de hbp(3.3) e hbr(3.4) respectivamente:

$$hbp = 2 * \cos(\omega 0 * (n - \frac{N}{2})) * hlp \tag{3.3}$$

$$hbr = \delta(n) - hbp \tag{3.4}$$

Após a criação dos filtros podemos somar os dois filtros e multiplicar o passa banda pelo ganho desejado do equalizador (3.5), onde gdB é o ganho informado pelo usuário:

$$hn = hbr + 10^{(gdB/20)} * hbp \tag{3.5}$$

### 3.1.2 Implementação do filtro no MATLAB

Nesta etapa o filtro desenvolvido foi implementado no *Software* de simulação MATLAB, [VKI12], apenas de uma forma gráfica, para observar como ficará as respostas em frequência conforme os parâmetros que serão informados pelo usuário, frequência central, largura da banda e ganho em *dB*, podendo ser negativo ou positivo. O objetivo desta etapa foi verificar a resposta do filtro e a ordem, tanto em frequências altas e baixas e em larguras de bandas estreitas e mais largas, para caso em alguma situação a ordem fique muito alta, possa ser feito ajustes no filtro para obter uma configuração com uma boa qualidade do filtro e uma ordem que não seja muito elevada para a resposta em tempo real poder ocorrer quando implementado no *hardware*.

Implementando o filtro desenvolvido no MATLAB, foram obtidos excelentes resultados, bem fiéis comparados com os parâmetros informados pelo usuário. Podemos ver na imagem abaixo um filtro com frequência de amostragem de 48kHz, frequência central igual a 1kHz, largura de banda de 500Hz, ganho de 1dB e largura de banda de transição de 500Hz, com esses parâmetros foi obtido uma ordem de 578:

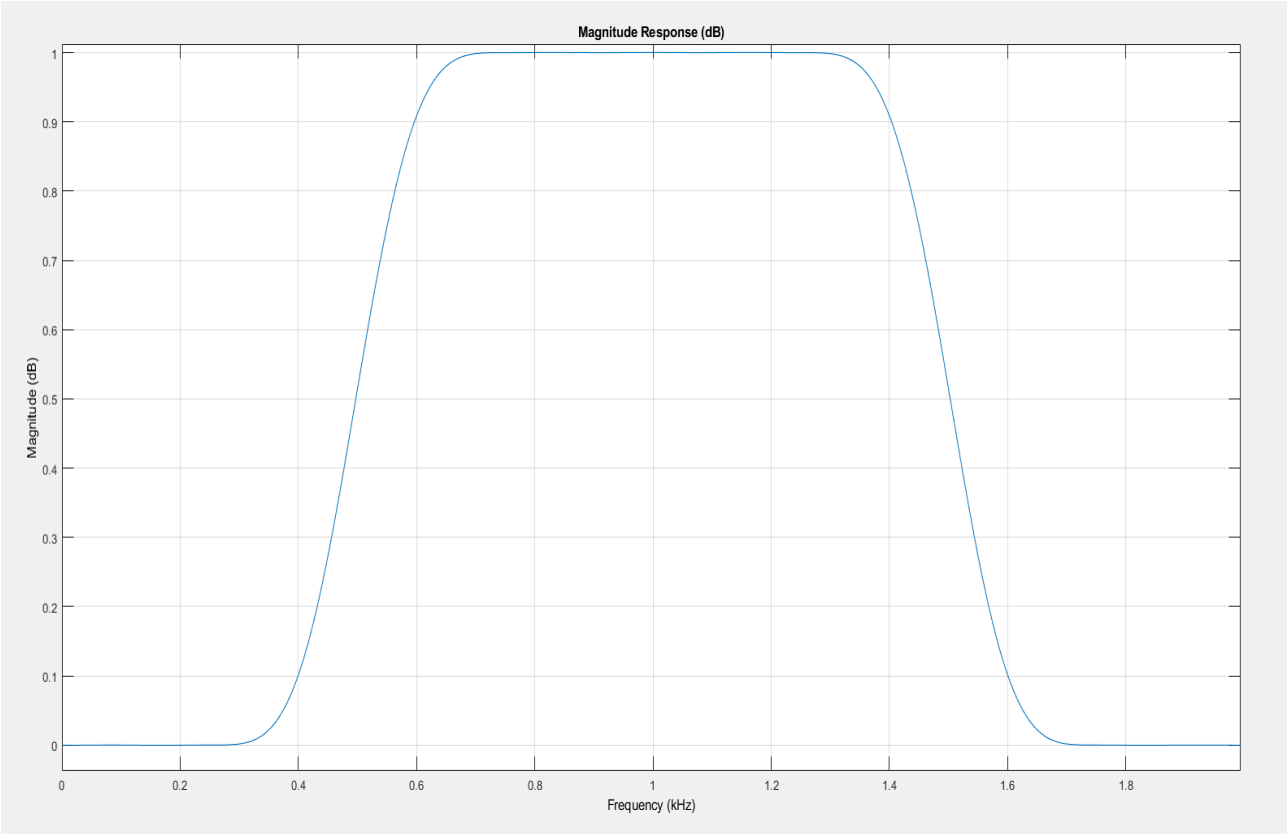



Figura 3.3: Teste do filtro no MATLAB
Fonte: Autor

Realizando outro teste com frequência de amostragem de 48kHz, frequência central de 500Hz, largura de banda de 100Hz, ganho de -5dB e largura de banda de transição de 500Hz, a ordem foi de 576:

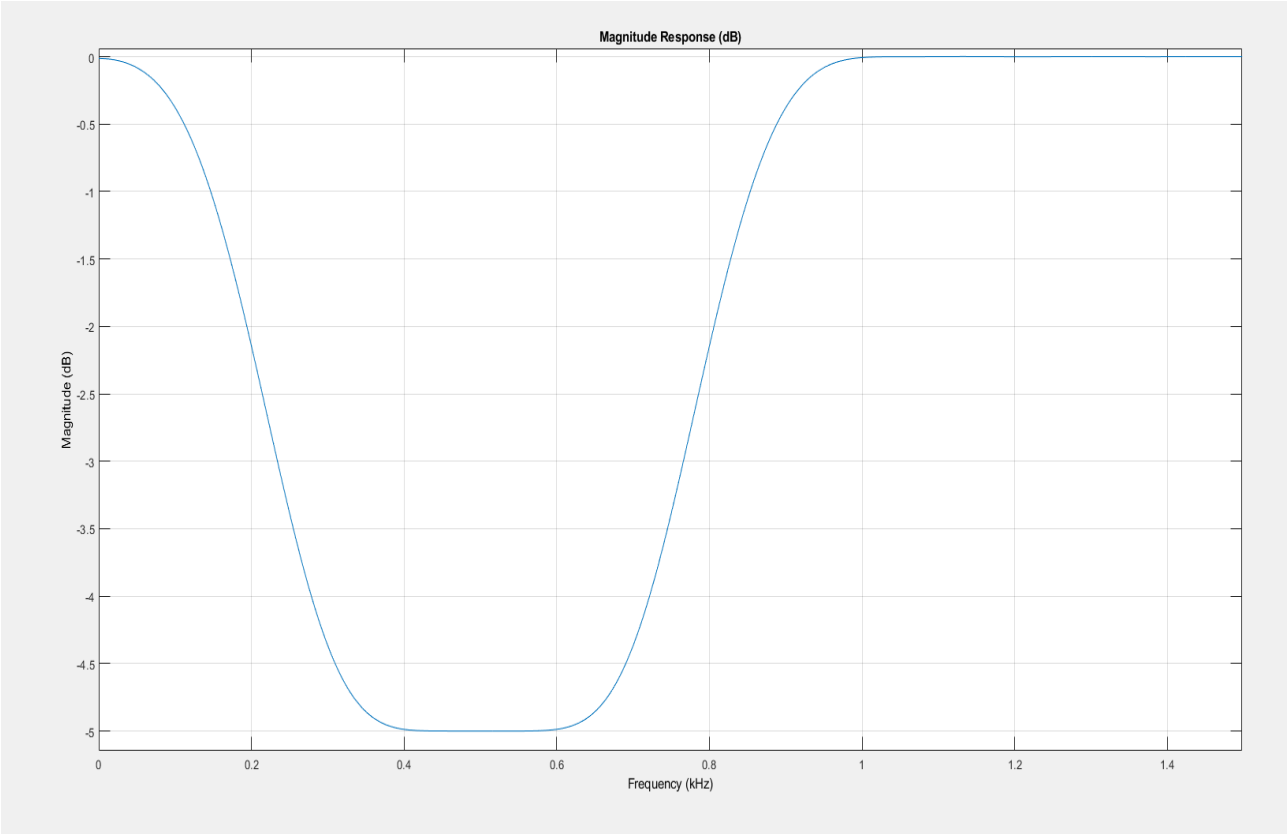



Figura 3.4: Teste do filtro no MATLAB
Fonte: Autor

Nesta etapa do projeto foi optado por deixar um parâmetro fixo, sendo a largura da banda de transição, pois quando ela é alterada para um valor menor que 300Hz a ordem ficaria muito grande para a execução na *Raspberry Pi 3*, pois quando ele se aproxima de um filtro de pico a ordem sobe bastante, um teste feito com os mesmos parâmetros da figura 3.5, onde a largura da banda de transição de 500Hz foi alterada para 100Hz assim o filtro possuindo uma nova ordem de 2882.

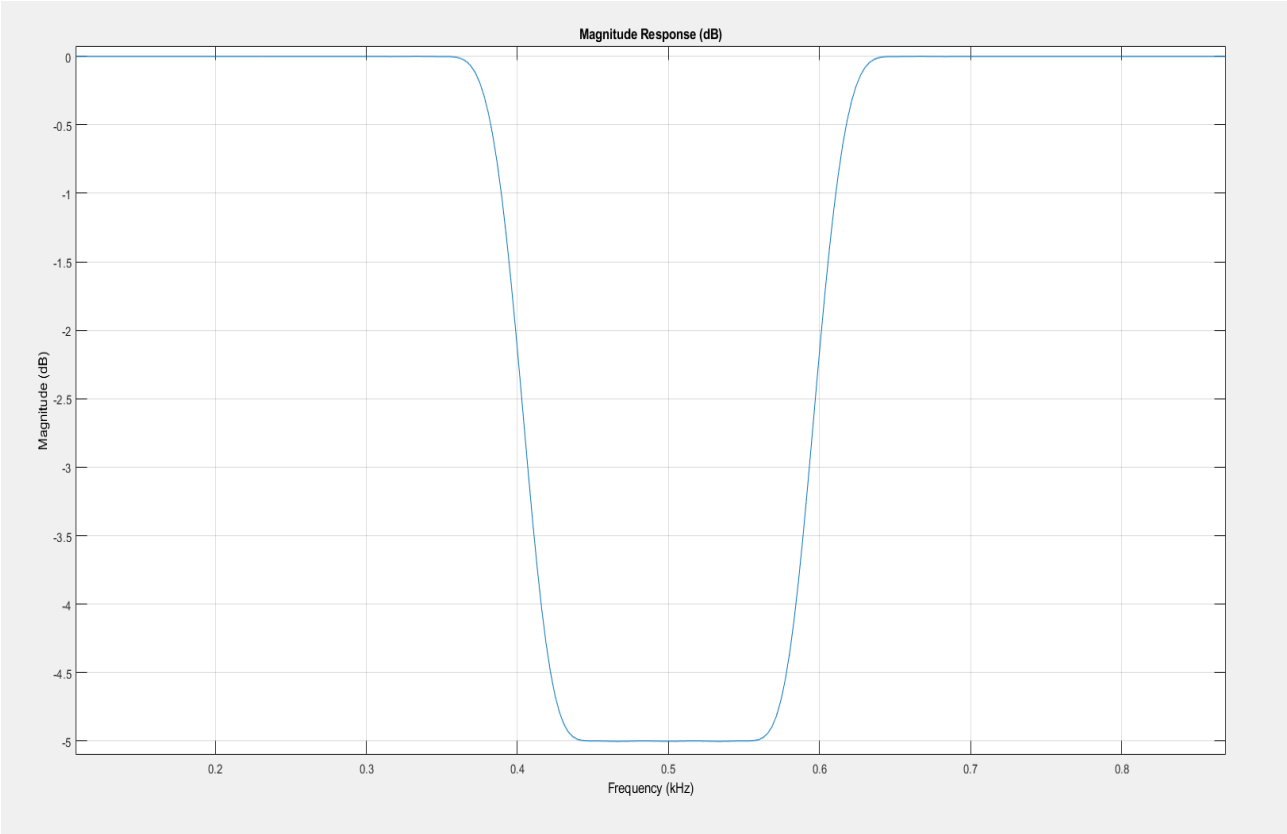



Figura 3.5: Teste do filtro no MATLAB
Fonte: Autor

Aqui também foram realizados testes utilizando o *pooling* no MATLAB, lendo as amostras de áudio e realizando a equalização no *software* retornando o áudio filtrado com os gráficos do oscilograma e espectro de frequência de entrada e de saída. Assim possuindo um protótipo do equalizador funcionado, com um recurso visual bem importante para verificar como o equalizador está se comportando em diferentes combinações de parâmetros.

Podemos ver nas figuras 3.6 e 3.7, dois testes sendo feitos, os dois com os mesmos parâmetros, frequência de amostragem de 48kHz, frequência central de 1kHz e ganho de 5dB, porém no primeiro teste o sinal de entrada foi uma senoide de 1kHz e o sinal de entrada do segundo uma senoide de 2kHz, onde as figuras da esquerda são do sinal de entrada e as figuras da direita o sinal de saída já equalizado.

É importante ressaltar que nos gráficos do espectro de frequência o ganho sempre é mostrado em um pois ele é normalizado.

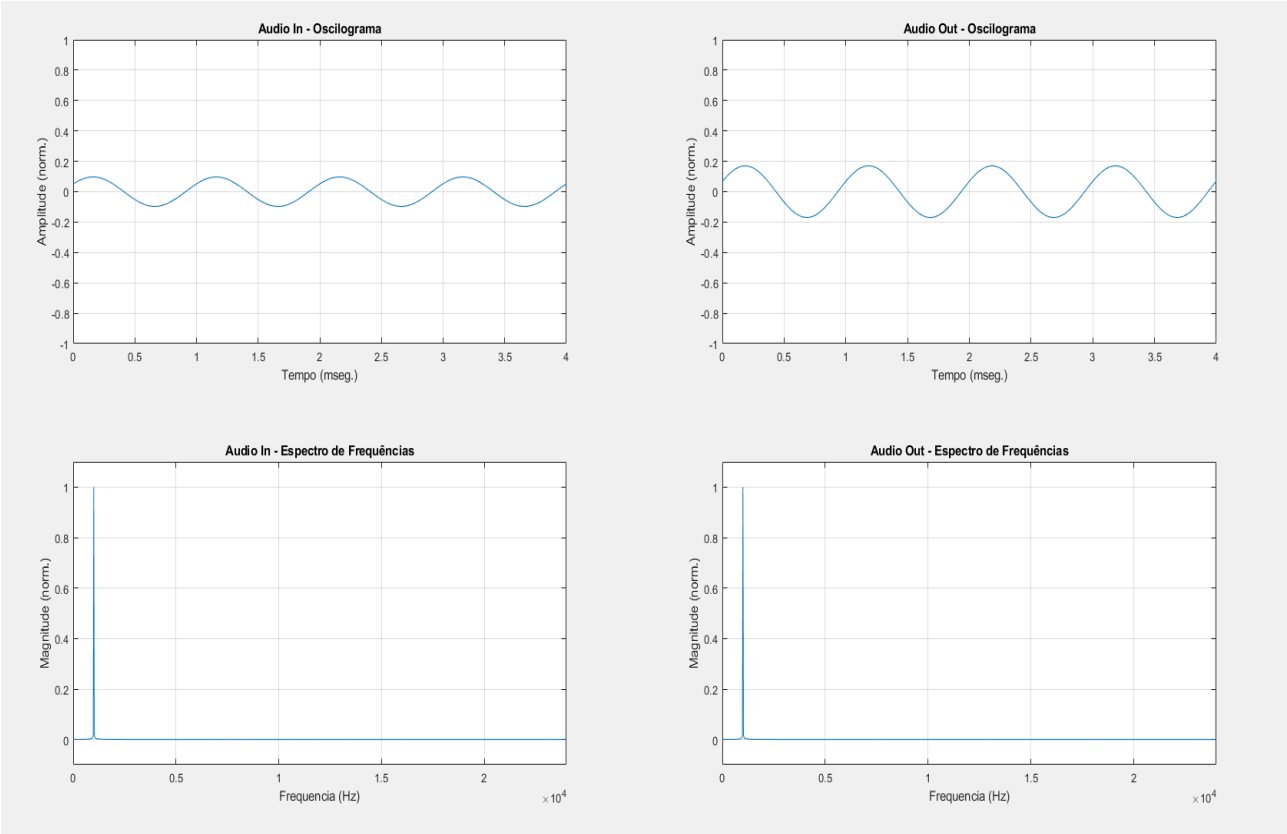



Figura 3.6: Teste do filtro no MATLAB
Fonte: Autor

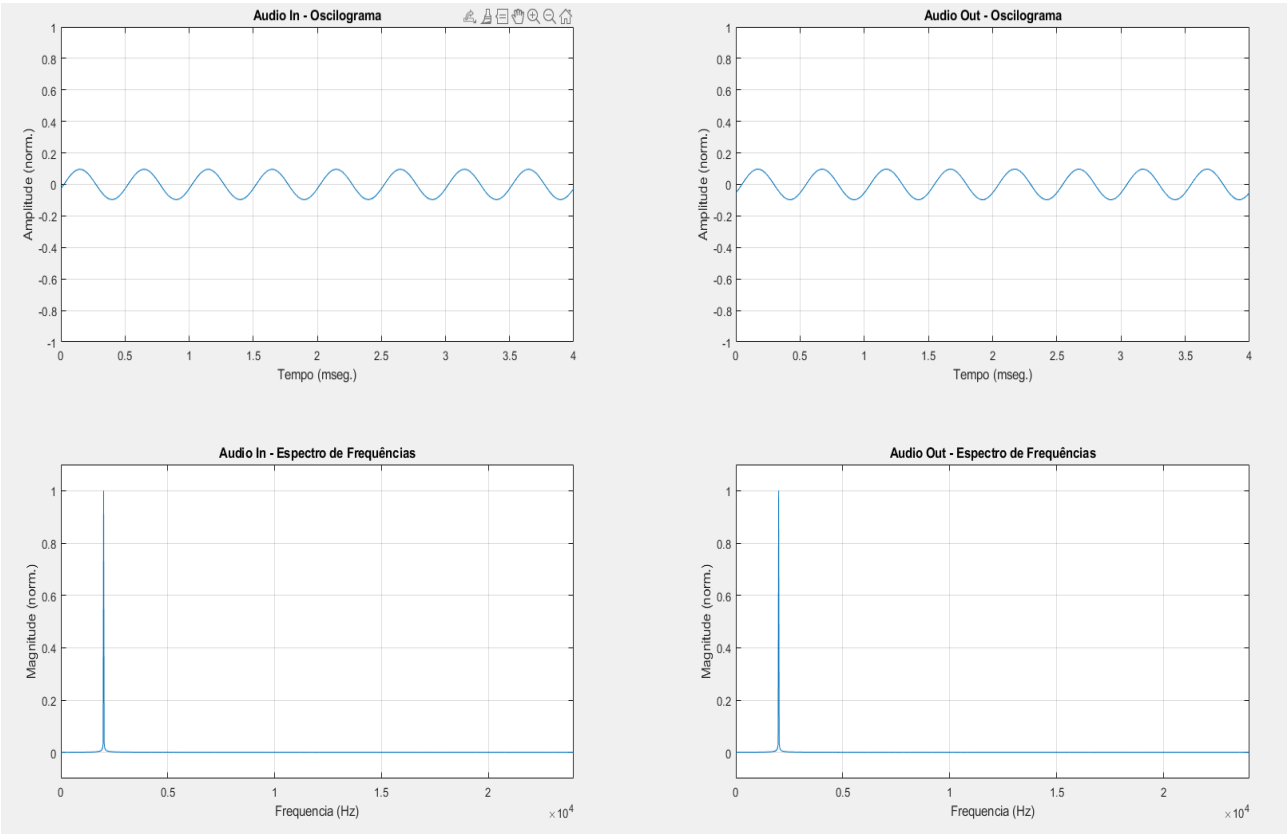



Figura 3.7: Teste do filtro no MATLAB
Fonte: Autor

Podemos observar na figura 3.6 que o sinal de saída recebe o ganho de 5dB comparado com o de entrada, pelo fato da entrada ser de 1kHz, já na figura 3.7 não é possível ver alteração na amplitude do sinal, pois o sinal de entrada está fora da faixa de ganho da frequência central.

Colocando uma senoide de 1,3KHz é visto que o ganho de 5dB é aplicado, pois estamos usando uma largura de banda de 500Hz, como mostra a figura 3.8

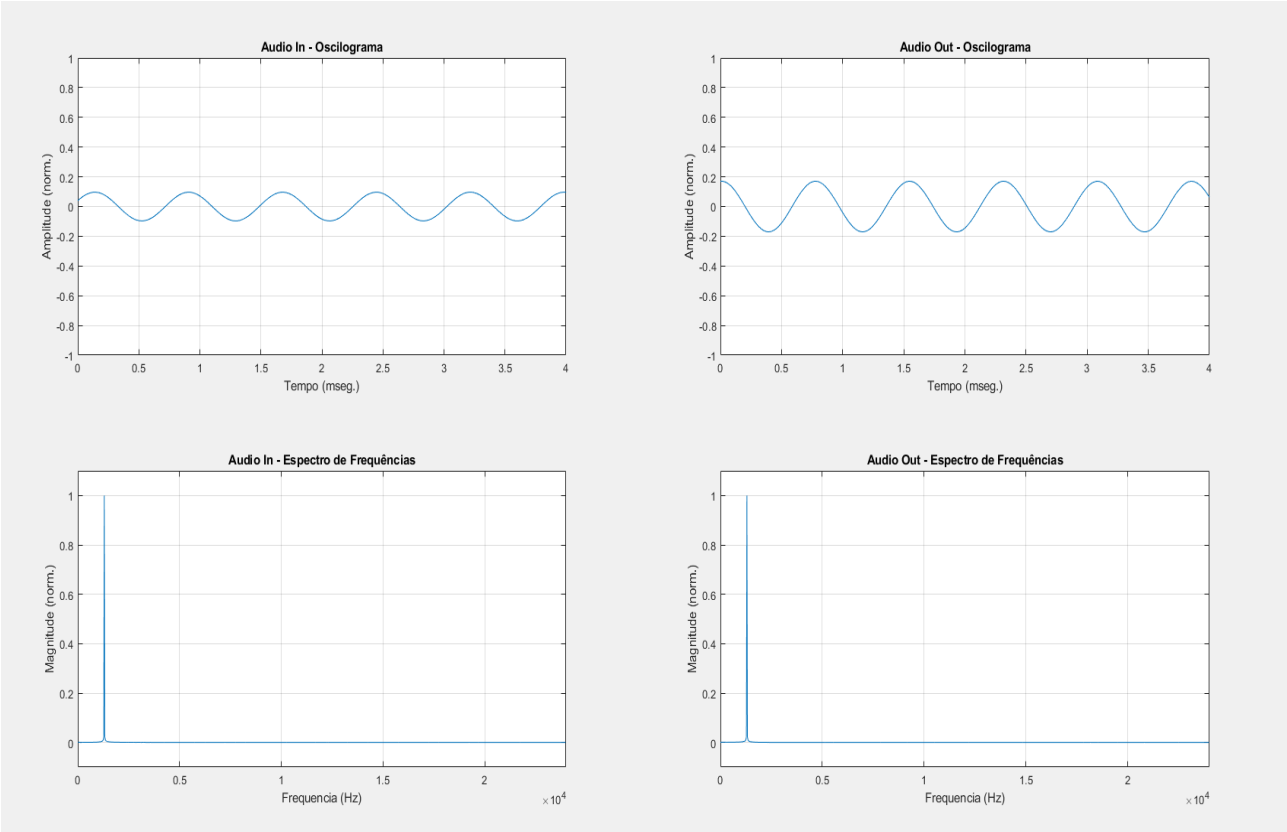



Figura 3.8: Teste do filtro no MATLAB
Fonte: Autor

Por fim realizando um teste mais prático foi colocado um sinal de entrada sendo um ruído branco somado com uma senoide de 1kHz, com os parâmetros do filtro sendo f0=1kHz, gdB=-25dB e largura de banda de 500Hz, podemos ver no lado esquerdo da figura 3.9, espectrograma o pico da senoide na faixa de 1kHz, já na imagem da direita o pico da senoide some, ficando apenas o ruído branco no espectrograma com uma atenuação na faixa de 1kHz.

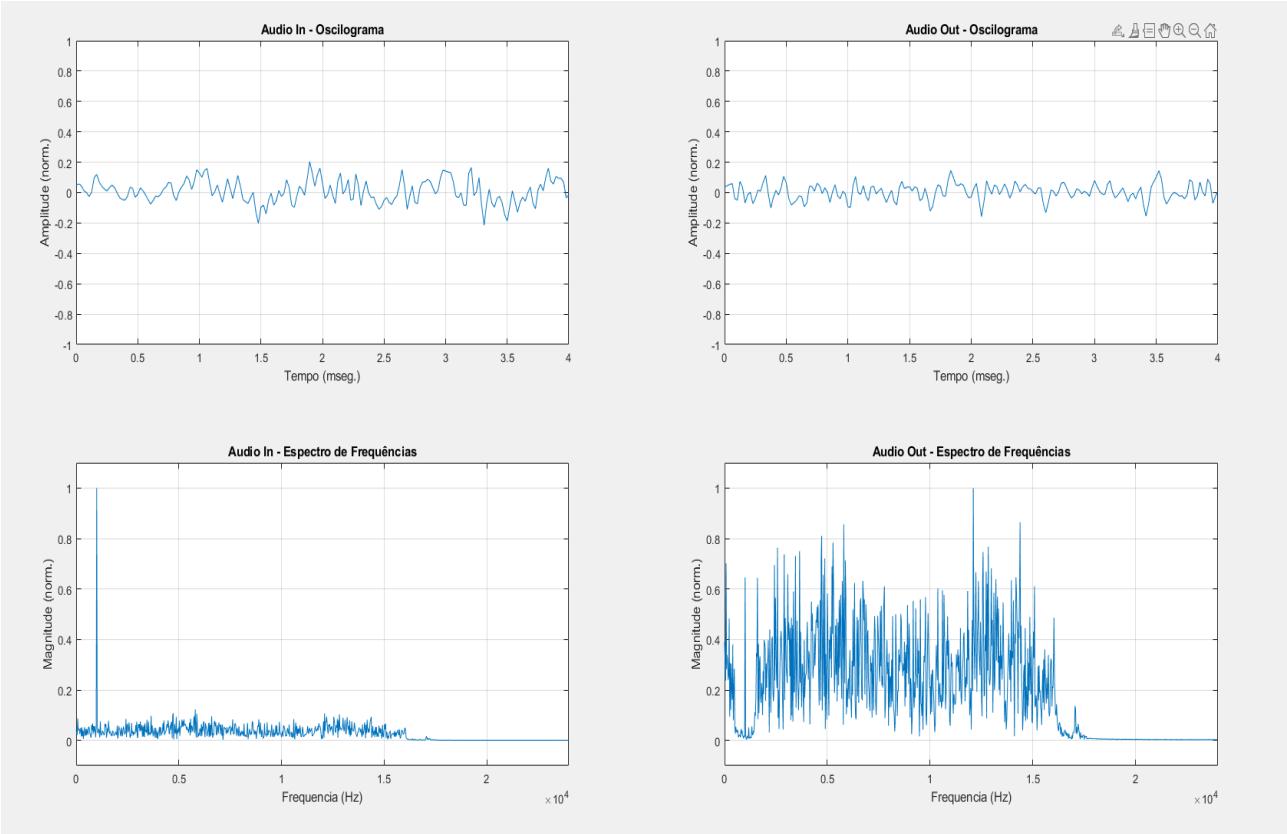



Figura 3.9: Teste do filtro no MATLAB
Fonte: Autor

### 3.1.3 Testes com uma interface de áudio externa no *Raspberry Pi 3*

Como o *Raspberry Pi 3* não possui conversor AD, foi usado uma interface de áudio da *Behringer* a *XENYX 302USB*, essa interface possui uma resolução de 16bit/48kHz, com as seguintes entradas e saídas:

- IN - Mic (XLR/P10);
- IN - Entrada de linha (RCA);
- IN - Track (RCA);
- OUT - Main MIX (RCA);
- IN - Mic (P2);
- OUT - Saída de áudio (P2);

A figura 3.10 mostra a interface externa de áudio utilizada para a aquisição do áudio equalizado.


Figura 3.10: Interface de áudio
Fonte: [Beh]

Então nesta etapa foram efetuados testes para ver como a interface de áudio se comporta com o sistema operacional que está rodando na *Raspberry*.

O sistema operacional usado foi o *Raspberry Pi OS Lite* disponível no site do fabricante.

Lembrando que esta interface externa será usada somente no projeto para testes e validação, podendo ser usado no futuro qualquer placa de áudio que possua conversor AD/DA.

A figura 3.11 mostra a *Raspberry Pi 3* que foi utilizada com a interface para fazer a equalização.

Figura 3.11: Raspberry Pi 3
Fonte: [Ras]

Observa-se que a placa possui 4 entradas USBs, uma entrada RJ45 para rede, uma saída HDMI, uma entrada micro SD, saída de áudio p2 e entrada para sua fonte de

alimentação micro USB. A placa possui um processador *Quad Core 1.2GHz Broadcom BCM2837 64bit CPU*, 1GB de memória RAM um chip *BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board* e 40 pinos de GPIO.

Os testes realizados com a interface com a *Raspberry Pi 3* foram um sucesso, não precisando instalar nenhum *drive* para o a *Raspberry Pi 3* reconhecer a placa de áudio, sendo *plug and play*.

Como a interface de áudio usada replica o som de entrada para a saída, foi optado por utilizar a saída *p2* da *Raspberry Pi 3* para a saída do sinal equalizado.

### 3.1.4 Desenvolvimento do filtro em uma linguagem suportada pela *Raspberry Pi 3*

Primeiramente a ideia era desenvolver o equalizador na linguagem C por ser uma linguagem de maior domínio do autor e mais otimizada, porém no decorrer do desenvolvimento foram enfrentados muitos problemas para a aquisição do sinal de entrada, pois como o sistema operacional da *Raspberry* é baseado em *Linux* teria que ser feito inúmeras chamadas de sistemas, assim tornando o código muito complexo. A partir deste problema foram pesquisadas outras abordagens para a aquisição do sinal, onde foi encontrada uma biblioteca em *python* que fazia a aquisição e mandava o sinal para o dispositivo de saída da máquina a *Sound Device*, [MIT21]. Esta biblioteca faz a aquisição do sinal de entrada conforme o dispositivo setado como principal, e disponibiliza uma lista do tamanho do número de amostras, contendo os valores do sinal de entrada, assim facilitando o tratamento do sinal e a equalização.

Com esta biblioteca é possível setar o número de canais, frequência de amostragem entre outros parâmetros, no caso deste trabalho foi apenas setado a frequência de amostragem para 48kHz e trocado o padrão de 2 canais para apenas 1.

O código começa setando os parâmetros da biblioteca, como dispositivo de entrada e saída, *SampleRate*, canais, etc. Após isso é feita a aquisição dos parâmetros, como frequência central, ganho, e largura de banda, para logo após poder ser feito o cálculo dos coeficientes filtro.

Para o cálculo do filtro é primeiramente feito o cálculo das frequências limites das bandas de passagem e rejeição tanto superior quanto inferior, em Hz, para logo após fazer a normalização dessas frequências. Para este projeto foi escolhido a janela de *Blackman*, para obter melhor qualidade no filtro. Após o cálculo da janela de *Blackman*, os filtros são calculados, passa-baixa, passa-banda e rejeita banda, e por fim feito o cálculo dos coeficientes como é mostrado na seção 3.1.1.

O próximo passo é calcular as condições iniciais do filtro, que neste caso, será um vetor com tamanho da ordem do filtro, com seus valores todos em zero.

Feito isso o programa chama a função de aquisição do sinal que é *loop* principal do programa, onde ele ficará todo o tempo de execução. Nesta função a biblioteca *Sound Device* nos disponibiliza os valores em formato de lista com as amostras de áudio, porém para o sinal ser filtrado com os coeficientes calculados foi utilizado uma função *lfilter* da biblioteca *Scipy*, [Org21], onde se é passado o vetor de coeficientes do filtro calculado, o sinal de entrada e as condições iniciais do filtro, devolvendo o sinal de saída filtrado, com as novas condições iniciais. Porém, esta função trabalha com vetores do tipo *np.array* e o sinal de entrada disponibilizado pela *Sound Device* é uma lista, por isso antes de a função *lfilter* ser chamada o sinal de entrada é transformado para np.array e logo após o cálculo o sinal de saída do tipo *np.array* é transformado novamente em lista para a biblioteca *Sound Device* mandar as amostras filtradas para o dispositivo de saída.

# 4. TESTES E RESULTADOS

Este capítulo apresenta os testes realizados e seus resultados, primeiro em um *Desktop* comum e por fim os testes realizados na *Raspberry Pi 3*.

## 4.1 Testes em um *Desktop* comum

Primeiramente foram realizados testes no *Desktop* utilizado para o desenvolvimento do equalizador na linguagem *Python*. O *hardware* utilizado foi um computador com um processador *Intel Core I7 9700KF* com frequência de 3.6GHz (4.9GHz *Max Turbo*) [Int] e 16GB de memória RAM.

Para o desenvolvimento neste computador foi adicionado um gráfico no equalizador para ajudar na visualização do sinal de saída. Mesmo com este auxilio gráfico o equalizador com uma ordem maior que a normal, entre 2000 e 5000, se comportou de maneira estável, possuindo um *delay* imperceptivel.

Pode se observar na figura 4.1 o equalizador recebendo uma senóide de 5kHz, com seus parâmetros ajustados para apenas dar um ganho de 1dB na frequência de 5kHz, apenas para uma primeira visualização. Para a entrada do sinal foi utilizado um aplicativo de gerador de sinais para celular, ligado na mesma interface de áudio usado na *Raspberry Pi 3*.

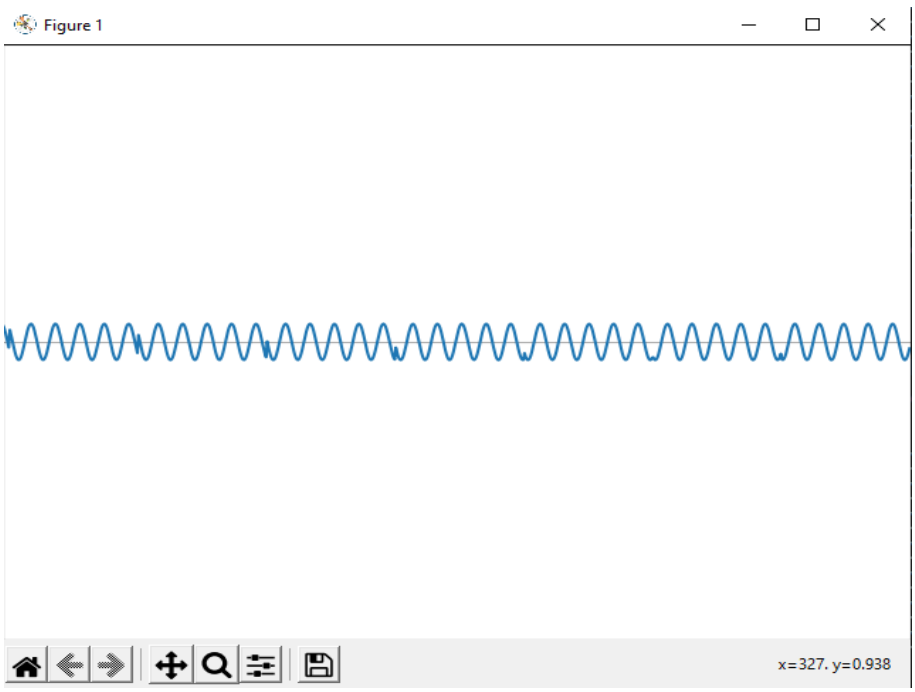



Figura 4.1: Teste do filtro em um Desktop, ordem de 578
Fonte: Autor

Observa-se um pequeno ruído no sinal, isso se dá a uma pequena interferência causada pelo cabo utilizado.

Realizando o mesmo teste anterior, porém com um ganho de 10dB, temos o seguinte resultado:

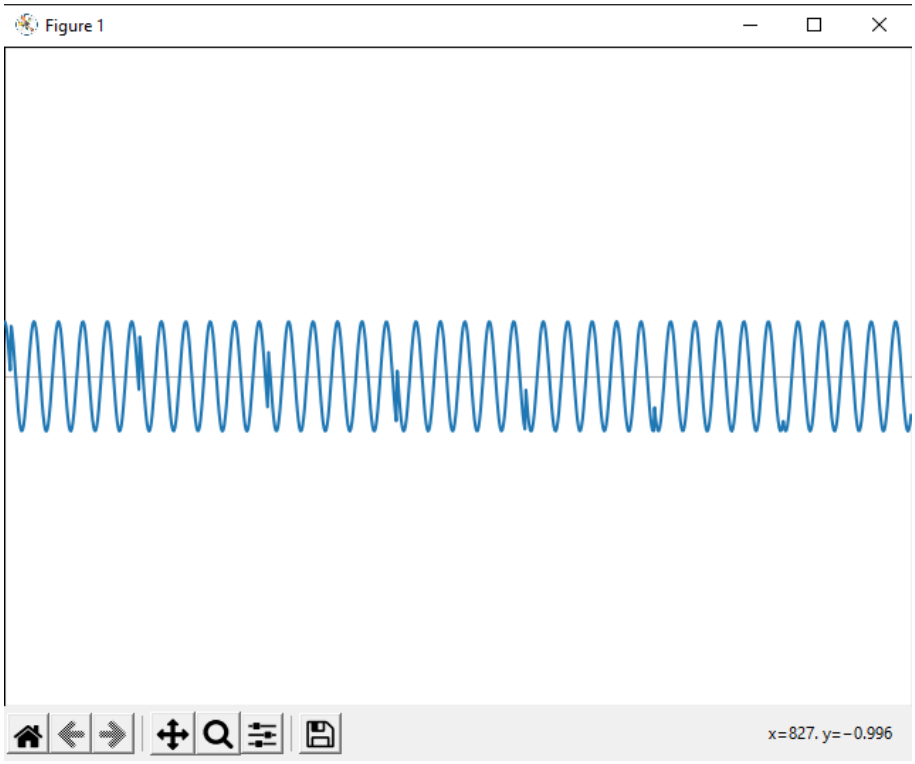



Figura 4.2: Teste do filtro em um Desktop, ordem de 578
Fonte: Autor

Para testar a atenuação foi colocado um ganho de -10 dBs, visto na figura 4.3, lembrando que esses 3 testes foram feitos com uma largura de banda de 500Hz e uma largura de banda de transição de 500Hz.


Figura 4.3: Teste do filtro em um Desktop, ordem de 578
Fonte: Autor

Para fim de testes foi diminuída a largura de banda para 50Hz, o equalizador se manteve estável e sua ordem baixou de 578 para 576, mostrando que este parâmetro pode ser disponibilizado sem restrições para o usuário modicar de acordo com sua necessidade. Porém, alterando a largura da banda de transição para 50, a ordem subiu para 5762, assim aumentando um pouco o *delay* no computador não sendo muito perceptível, mas com esta ordem o *delay* na *Raspberry Pi 3* ficaria mais de 5 segundos.

## 4.2 Testes na *Raspberry Pi 3*

Após a validação do equalizador no PC, foram iniciados os testes na *Raspberry Pi 3*, foi primeiramente utilizado o mesmo teste do *desktop*, porém com o uso do auxílio gráfico no *python* o atraso na *Raspberry* ficou muito grande, dificultando muito o uso e os testes. Para contornar este problema foi retirado o auxílio gráfico e apenas usado o áudio de saída para a validação do equalizador no *hardware*.

Primeiramente foi usado um segundo celular ligado na saída do *Raspberry* com um aplicativo de osciloscópio, para validar os ganhos.

Os primeiros testes foram feitos com senoides de entrada com uma frequência mais baixa.

Figura 4.4:

- Frequência central, f0 = 200;
- Largura de banda, bf = 500;
- Frequência de amostragem, fa = 48kHz;
- Ganho em dB, GdB = 1;
- Senóide de entrada 200Hz;
- Ordem 576;

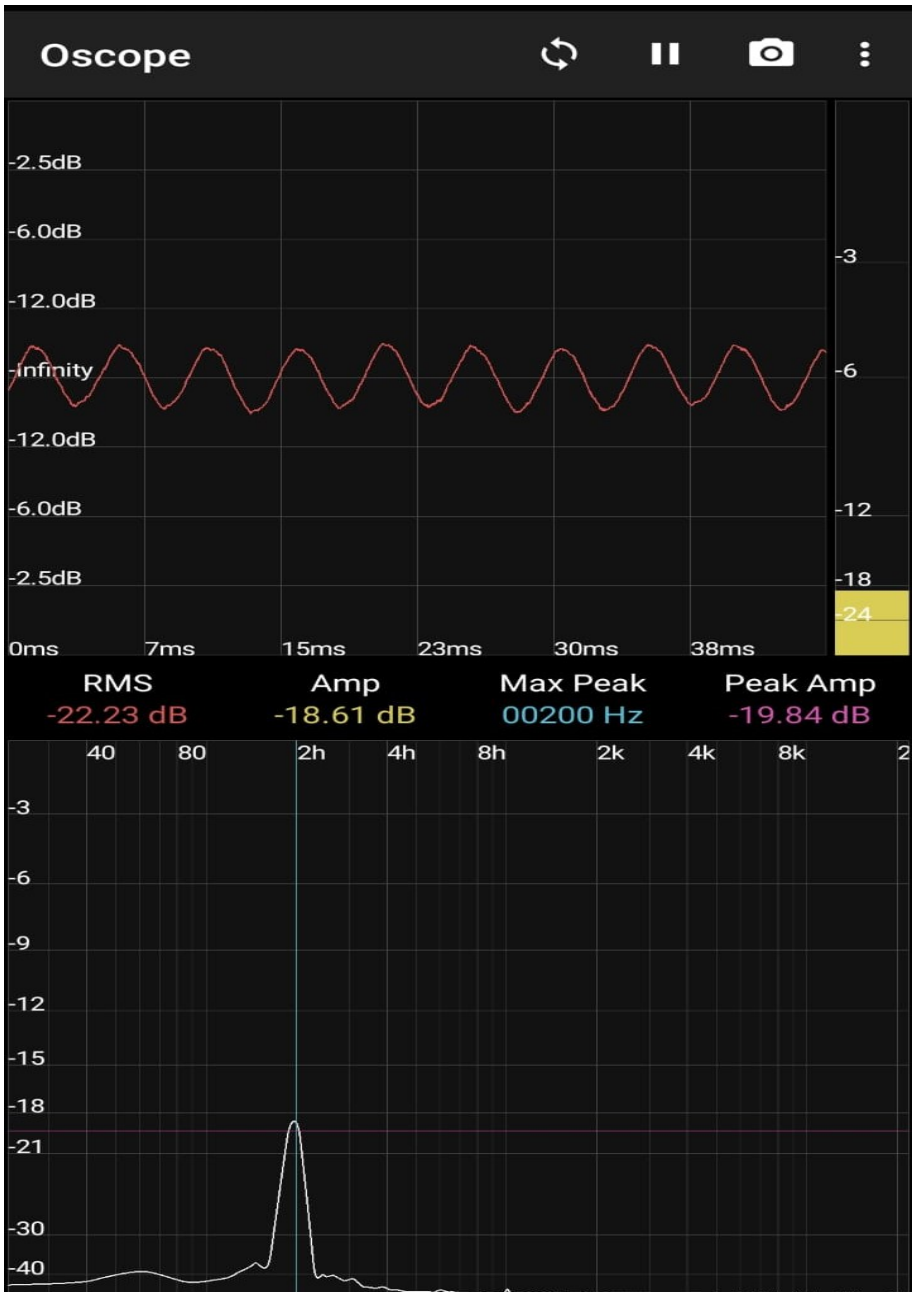



Figura 4.4: Teste do filtro no *Raspberry*
Fonte: Autor

Figura 4.5:

- Frequência central, f0 = 200;
- Largura de banda, bf = 500;
- Frequência de amostragem, fa = 48kHz;
- Ganho em dB, GdB = -10;
- Senóide de entrada 200Hz;
- Ordem 576;

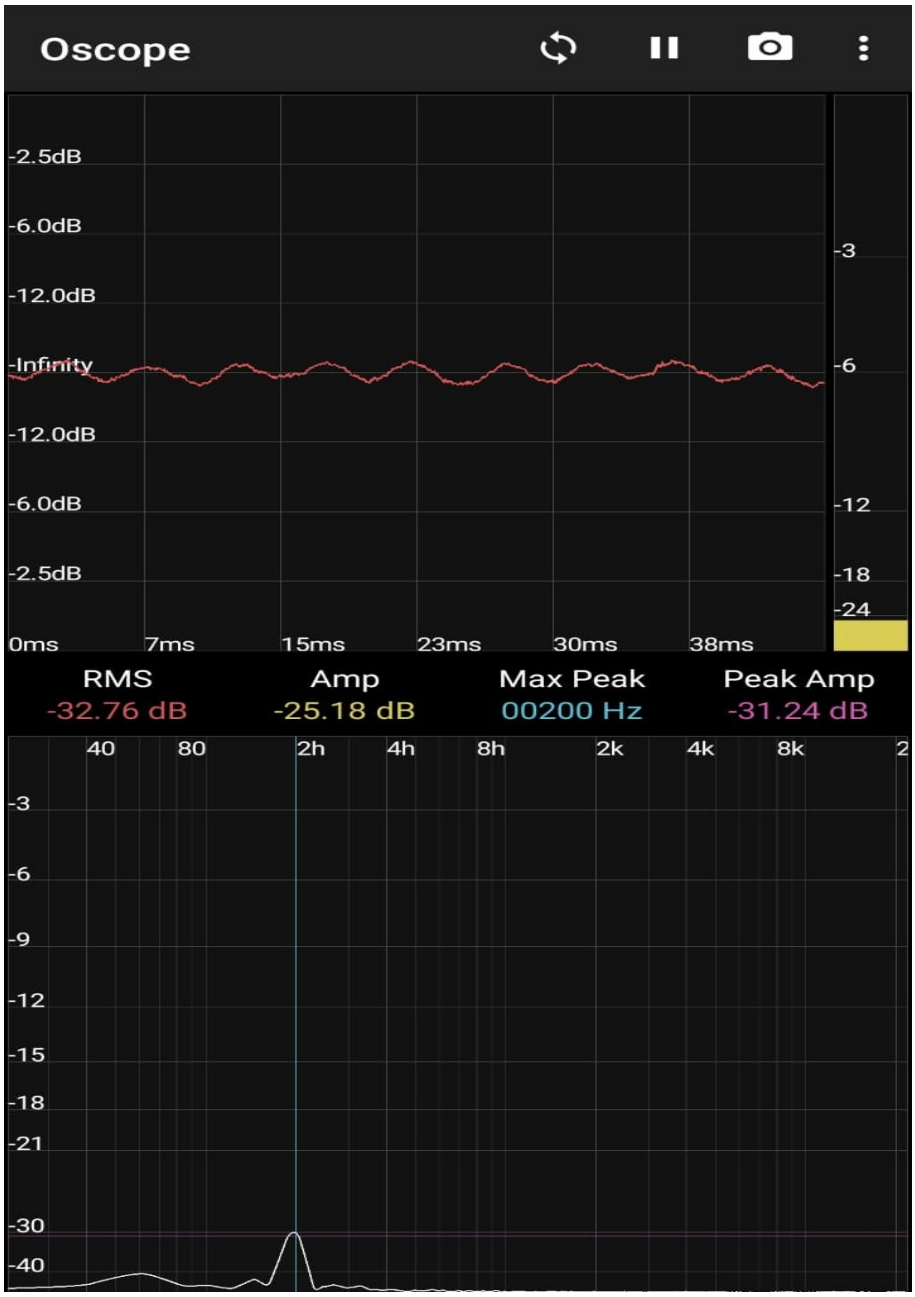



Figura 4.5: Teste do filtro no *Raspberry*
Fonte: Autor

Logo após foram feitos testes para validar as frequências acima de 1kHz.

Figura 4.6:

- Frequência central, f0 = 1000;
- Largura de banda, bf = 500;
- Frequência de amostragem, fa = 48kHz;
- Ganho em dB, GdB = 1;
- Senóide de entrada 1000Hz;
- Ordem 578;

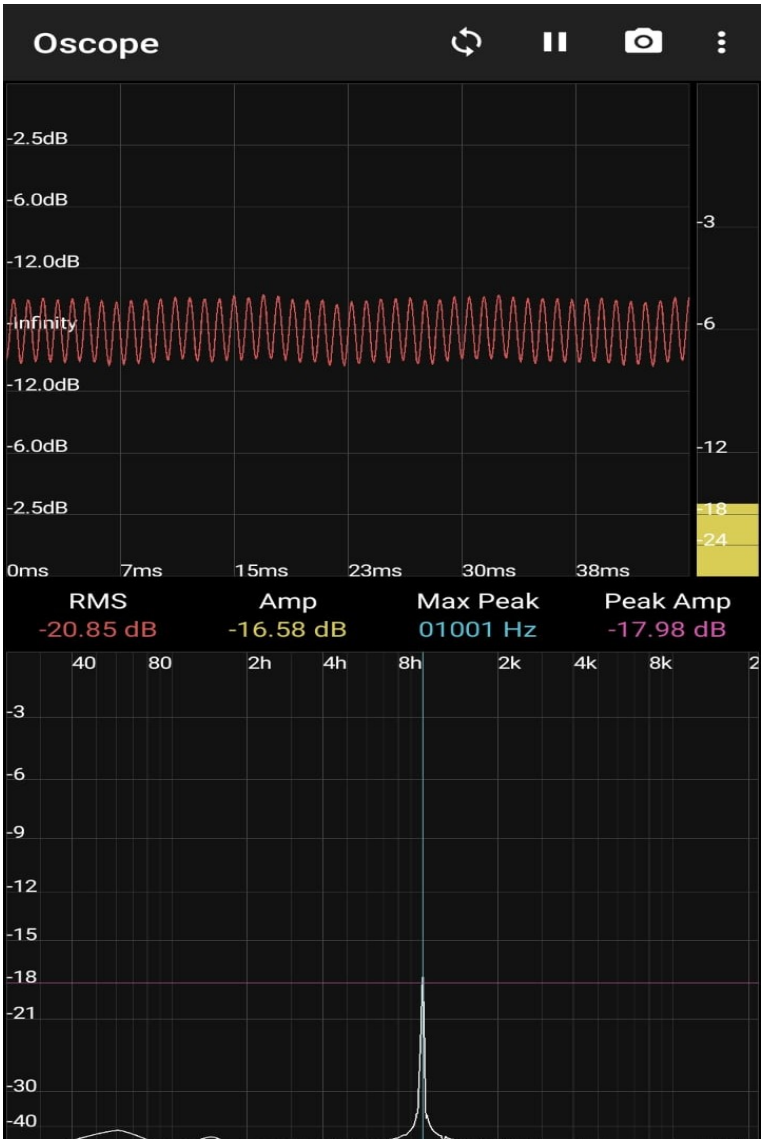



Figura 4.6: Teste do filtro no *Raspberry*
Fonte: Autor

Figura 4.7:

- Frequência central, f0 = 1000;
- Largura de banda, bf = 500;
- Frequência de amostragem, fa = 48kHz;
- Ganho em dB, GdB = 10;
- Senóide de entrada 1000Hz;
- Ordem 578;

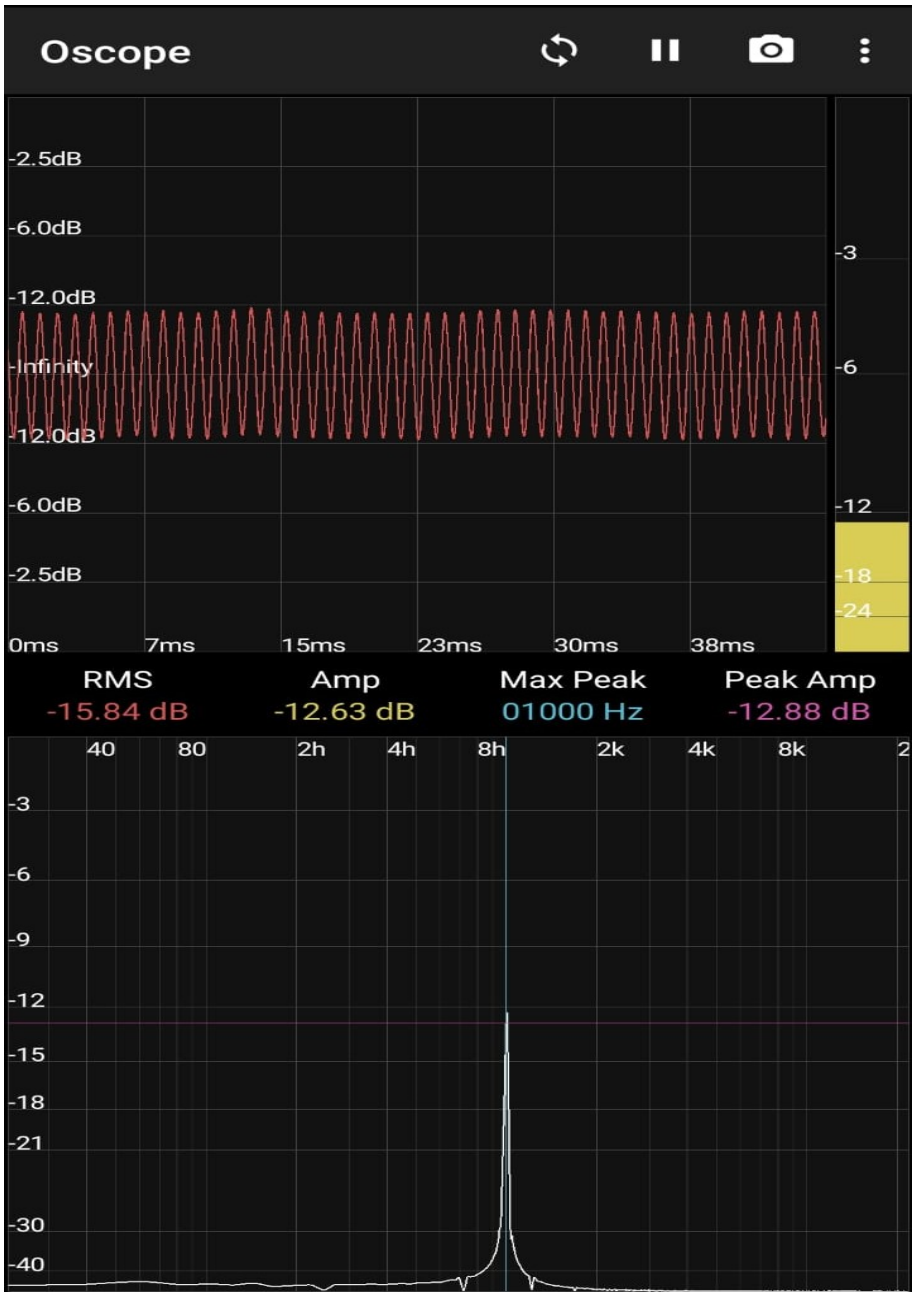



Figura 4.7: Teste do filtro no *Raspberry*
Fonte: Autor

Também foram feitos testes para validar a largura de banda:

- Frequência central, f0 = 1000;
- Largura de banda, bf = 500;
- Frequência de amostragem, fa = 48kHz;
- Ganho em dB, GdB = 10;
- Senóide de entrada 1200Hz;
- Ordem 578;

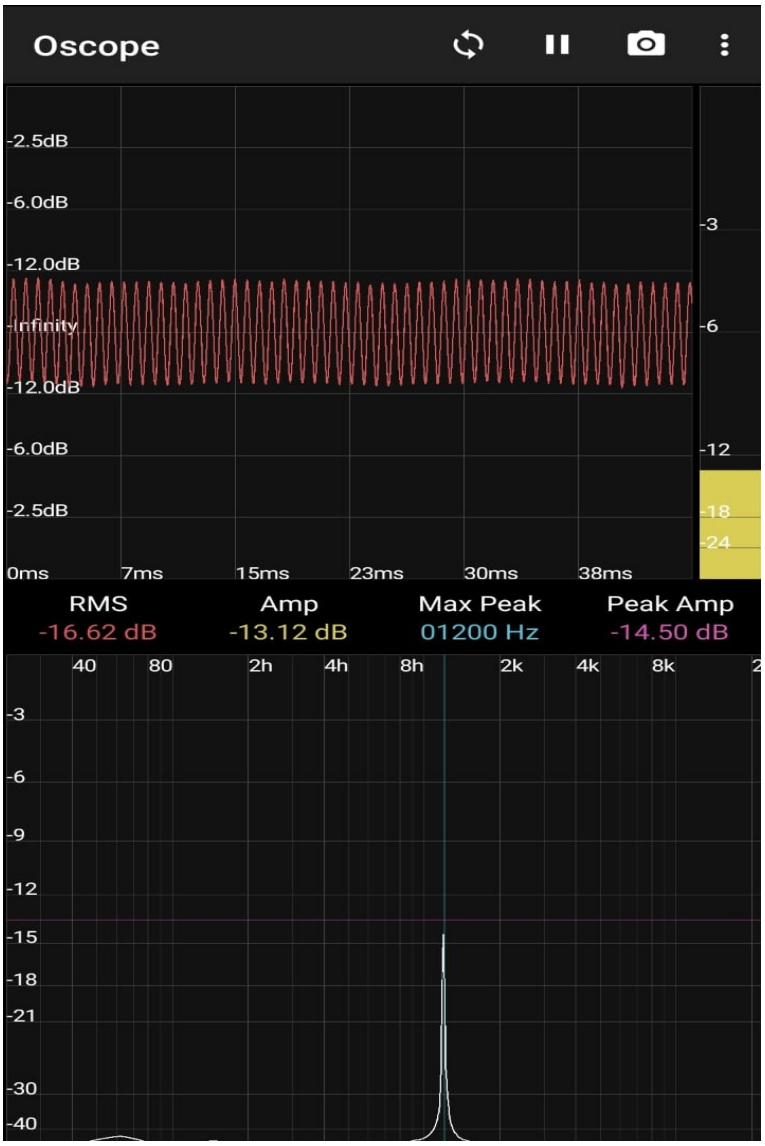



Figura 4.8: Teste do filtro no *Raspberry*
Fonte: Autor

Observamos na figura 4.8, que o ganho visto na figura 4.7 também é aplicado com uma frequência de entrada contida na largura de banda de 500Hz.

- Frequência central, f0 = 1000;
- Largura de banda, bf = 500;
- Frequência de amostragem, fa = 48kHz;
- Ganho em dB, GdB = 10;
- Senóide de entrada 1600Hz;
- Ordem 578;

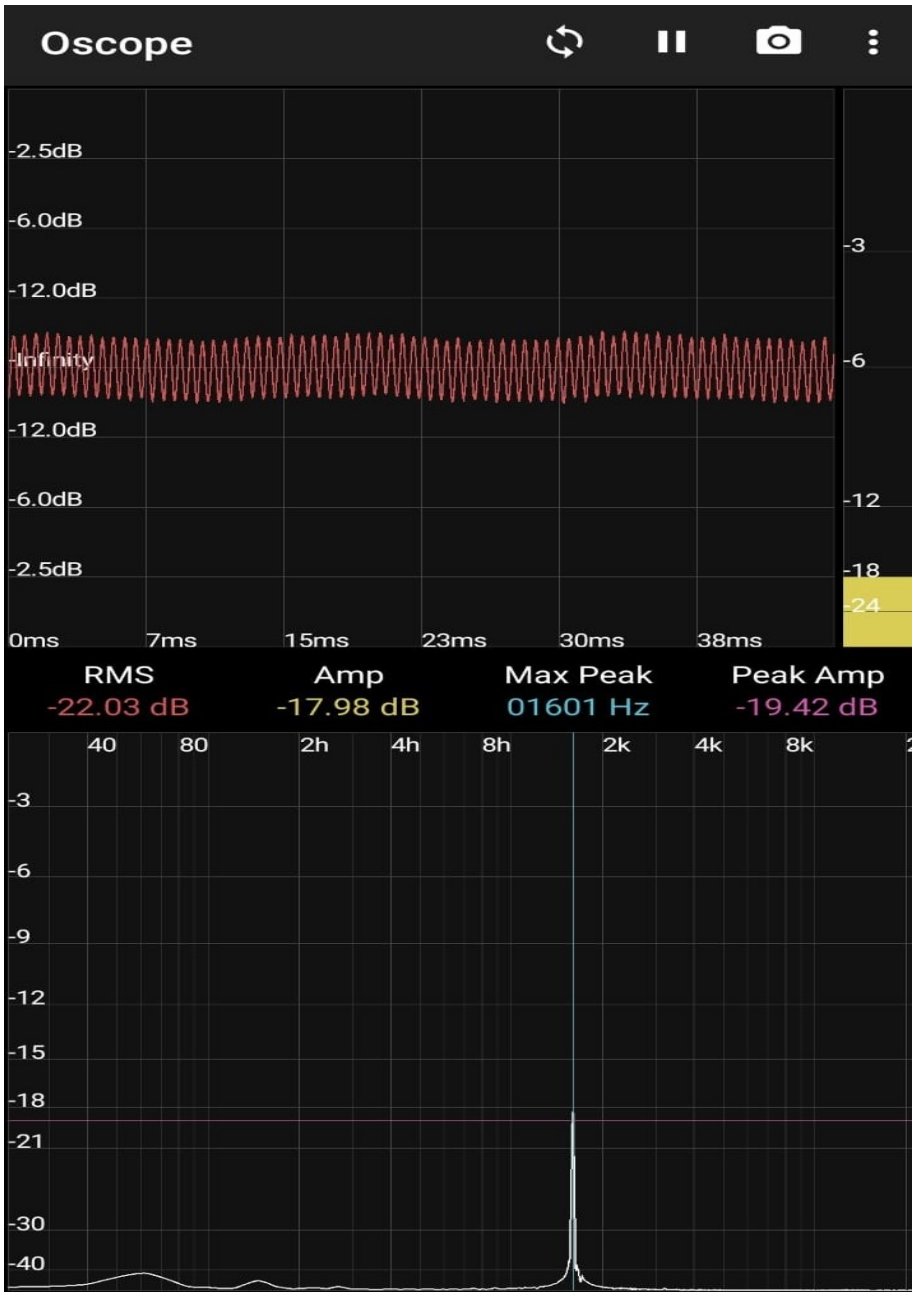



Figura 4.9: Teste do filtro no *Raspberry*
Fonte: Autor

Já na figura 4.9, é visto que o ganho não é alterado.

Utilizando um espectro de frequência em um celular ligado na saída da *Raspberry*, podemos ver um ganho de 10dB numa frequência de 5kHz, que sem a intervenção do equalizador estava em 30dB e após o uso do programa ele sobe para 40dB.

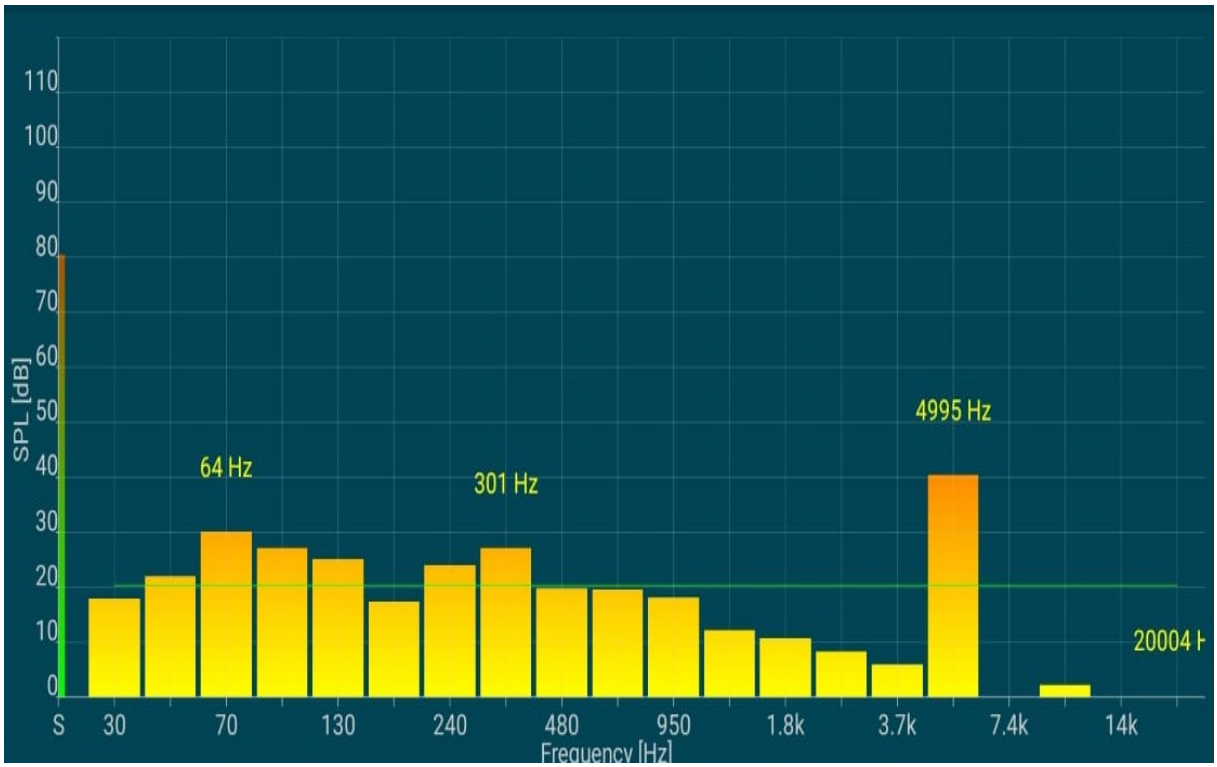



Figura 4.10: Teste do filtro no *Raspberry*
Fonte: Autor

Para testar o equalizador, na prática, foi feita a soma de uma música, com uma senoide de 1kHz, e no equalizador foram setados os parâmetros para eliminar essa senoide, com frequência central de 1kHz, largura de banda de 100Hz, e ganho de -30dB, a ordem do filtro foi de 576.

Na figura 4.11 vemos a música sem a atuação do equalizador, no espectro de frequência vemos a senoide observando o pico em 1kHz.

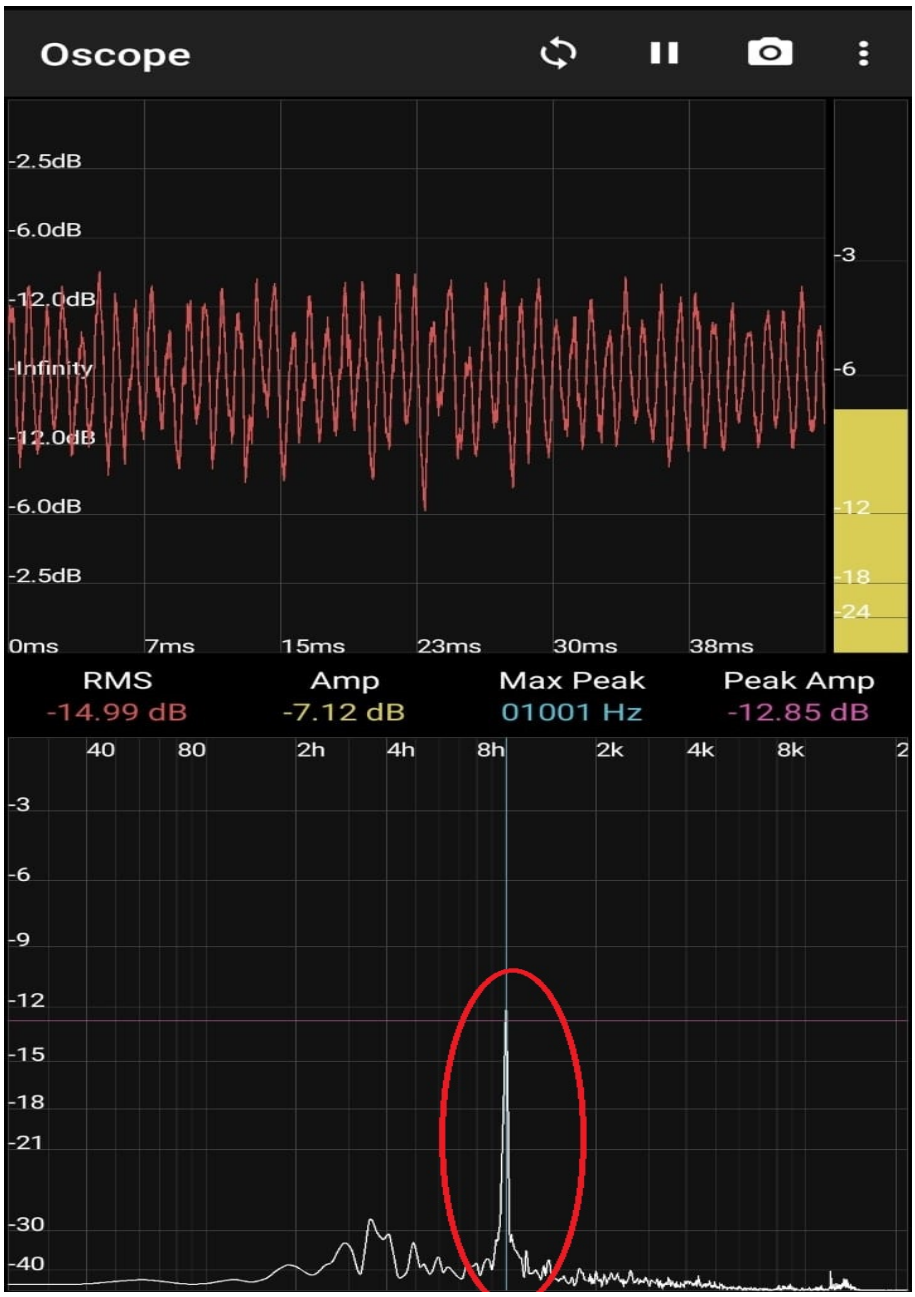



Figura 4.11: Música juntamente com uma senóide de 1kHz sem filtro
Fonte: Autor

Após o filtro aplicado vemos que o pico em 1kHz não é mais visível.

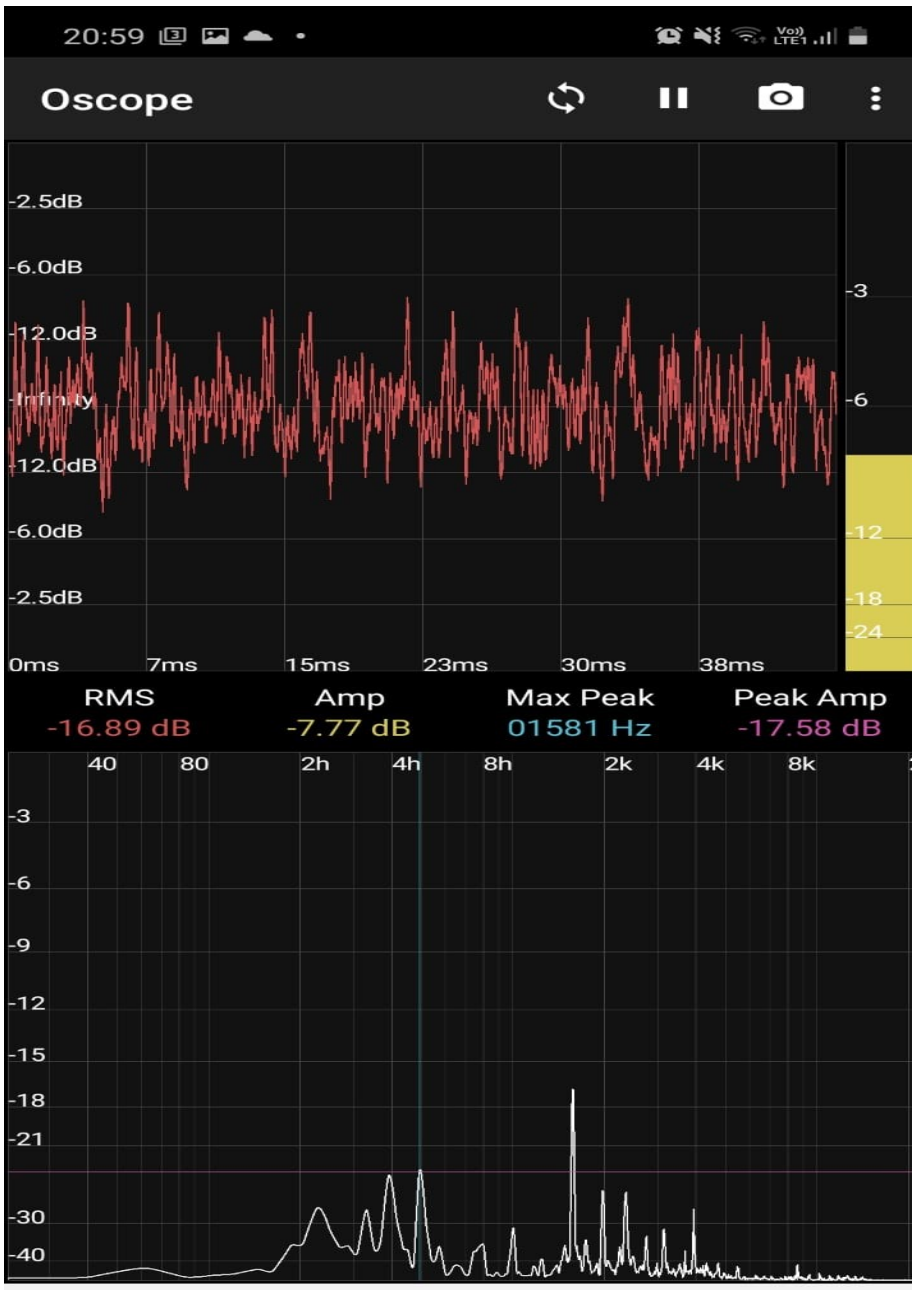



Figura 4.12: Música juntamente com uma senóide de 1kHz com filtro
Fonte: Autor

# 5. CONCLUSÃO

Com o desenvolvimento deste trabalho foi visto que é possível desenvolver um equalizador paramétrico de boa qualidade, com os parâmetros totalmente ajustáveis pelo usuário, com um valor de cerca de 500 reais, apenas utilizando um *Raspberry Pi 3* com uma placa de áudio.

Os resultados foram muito satisfatórios, pois o equalizador teve um desempenho ótimo tanto no *Desktop* que foi usado para desenvolvimento, utilizando filtros com ordens mais altas, como no *Raspberry Pi 3* que por mais que só suporte ordens de no máximo 1000, apresentou um *delay* praticamente imperceptível.

## 5.1 Melhorias Futuras

Como melhorias futuras, uma das principais coisas a se fazer é tentar compilar o código e criar um executável. O *Python* é uma linguagem de programação compilada em tempo de execução, assim num futuro, podendo utilizar alguma ferramente de compilação como o *Cython*, [Cyt11], para tentar algum ganho diminuindo o atraso em ordens maiores, pois como se trata de áudio, qualquer redução de milissegundos que seja aplicado ao projeto já é um grande ganho.

Outra possível melhoria é utilizar um *shield* de áudio específico para a *Raspberry Pi 3*, pois a interface de áudio utilizada nos testes além do seu preço ser mais elevado, não é o *hardware* ideal para a proposta do trabalho, podendo até diminuir o tamanho do equalizador.

Pode ser realizados testes também com algum sistema operacional mais leve, sem nenhuma outra aplicação instalada, assim também podendo gerar alguma melhoria para o sistema, como também a pesquisa de algum auxílio gráfico que não consuma tanto da CPU, não interferindo no atraso e gerando um auxílio gráfio para o usuário.

# REFERÊNCIAS BIBLIOGRÁFICAS


[BAU18] BAUER, M. “SISTEMA PARA EQUALIZAÇÃO PARAMÉTRICA DE ÁUDIO EM TEMPO REAL”. Capturado em: http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/12120/1/PB_COENC_2018_2_12.pdf, 2018.

[Beh] Behringer. “302USB”. Capturado em: https://www.behringer.com/product.html?modelCode=P0ADV.

[Cyt11] Cython. “Cython Documentation”. Capturado em: https://cython.org/#documentation, 2011.

[Han] Hanna, C. “Real-Time Control of DSP Parametric Equalizers”. Capturado em: http://www.thatcorp.com/datashts/AES13-041_Control_of_DSP_Parametric_EQs.pdf.

[Int] Intel. “Intel Core i7 9700”. Capturado em: https://www.intel.com.br/content/www/br/pt/products/sku/191792/intel-core-i79700-processor-12m-cache-up-to-4-70-ghz/specifications.html.

[MIT21] MIT. “python-sounddevice, version 0.4.3”. Capturado em: https://python-sounddevice.readthedocs.io/en/0.4.3/, 2021.

[Org21] Organization, S. “SciPy documentation”. Capturado em: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html?highlight=lfilter, 2021.

[Ram13] Ramírez, M. A. M. “DESIGN AND IMPLEMENTATION OF A PARAMETRIC EQUALIZER USING IIR AND FIR FILTERS”. Capturado em: https://marquetem.files.wordpress.com/2017/09/design-and-implementation-of-a-parametric-equalizer-using-iir-and-fir-filters-1.pdf, 2013.

[Ras] Raspberry. “Raspberry Pi 3 Model B”. Capturado em: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[VAR21] VARELA, J. N. “Equalizador Paramétrico de Áudio Digital com cinco Bandas, Interface GráficaTouch Screen e Operação em Tempo Real”. Capturado em: https://repositorio.ifsc.edu.br/bitstream/handle/123456789/2068/TCC_Jose_Nicolau_Varela_Eng_Eletronica.pdf?sequence=1, 2021.

[VKI12] Vinay K. Ingle, J. G. P. “Digital Signal Processing Using MATLAB®”. Third Edition; Cengage Learning, 2012.

[Zol95] Zolzer, U. “Digital Audio Signal Processing”. First Edition, 1995.

# APÊNDICE A – FILTRO DESENVOLVIDO NO MATLAB

A seguir é apresentado o filtro na linguagem .m.

```
clear;
clc;

fo = 1000; % frequencia central em Hz
bf = 500;  % banda para projeto do rejeita banda em Hz
gdB = -5; % ganho em dB na frequencia fo

fa = 48000; % frequencia de amostragem em Hz

lbt = 500; % largura de banda de transicao para calculo da ordem do filtro
fsi = fo - bf/2 - lbt; % frequencia limite da banda de passagem inferior em Hz
fpi = fo - bf/2;       % frequencia inferior da banda de rejeicao em Hz
fps = fo + bf/2;       % frequencia superior da banda de rejeicao em Hz
fss = fo + bf/2 + lbt ; % frequencia limite da banda de passagem superior em Hz

% Janela de Blackman: ap = 0.002 dB e as = 74 dB
% Ver formulario de consulta disponível no Moodle

% Frequencias normalizadas (w = 2*pi*f/fa)
wsi = 2*pi*fsi/fa;
wpi = 2*pi*fpi/fa;
wps = 2*pi*fps/fa;
wss = 2*pi*fss/fa;

% Fequencia de corte do filtro passa-baixas
w1 = (wpi+wsi)/2;
w2 = (wss+wps)/2;
wc = (w2-w1)/2;

% Frequencia central do passa-banda
w0 = (w2+w1)/2;

% Largura da banda de transicao para projeto do rejeita banda
dw = min((wpi-wsi),(wss-wps));


N = 12*pi/dw;  % janela de Blackman
N = 2*ceil(N/2); % ordem deve ser par
wn = blackman(N+1);

n = 0:N;
hlp = sin(wc*(n+1e-8-N/2))./(n+1e-8-N/2)/pi; % passa-baixas
hbp = 2*cos(w0*(n-N/2)).*hlp; % passa-banda
```

```
44 hbr = -hbp;
45 hbr(N/2+1) =1+hbr(N/2+1); % rejeita-banda
46
47 hn = hbr + 10^(gdB/20)*hbp;
48
49 % Resposta a amostra unitaria (impulso) do filtro a ser implementado
50 hn = hn.*wn';
51
52 % Visualizacao da resposta do filtro
53 fvtool(hn, 'fs',fa);
```

# APÊNDICE B – EQUALIZADOR NO MATLAB

A seguir é apresentado o equalizador implementado no matlab.

```
close all; % fecha todas as figuras
clear;     % limpa variáveis do workspace
clc;       % limpa janela de comandos

TT = 300.0; % tempo total de aquisicao em segundos

FA_AD = 48000; % frequencia de amostragem
FA_DA = FA_AD; % frequencia de amostragem
CH_AD = 1;     % 1 mono, 2 estereo

NSAMPLES_AD = FA_AD/25; % bloco de amostras por aquisicao do A/D.
NSAMPLES_DA = FA_DA/25; % bloco de amostras para reproducao no D/A.

% plota oscilograma do sinal de entrada
subplot(221);
xh1 = (0:NSAMPLES_AD-1)/(FA_AD*0.001);
yh1 = xh1*0;
h1 = plot(xh1,yh1); hold off; grid on;
axis([[0 NSAMPLES_AD/10]/(FA_AD*0.001) -1.0 1.0]);
xlabel('Tempo (mseg.)'), ylabel('Amplitude (norm.)');
title('Audio In - Oscilograma');
set(h1,'YDataSource','yh1');
set(h1,'XDataSource','xh1');

% plota espectro do sinal de entrada
subplot(223);
yh2 = abs(fft(yh1,length(yh1)));
yh2 = yh2(1:length(yh2)/2);
yh2 = yh2/max(yh2);
xh2 = (0:(length(yh2)-1));
xh2 = xh2*FA_AD/2/length(xh2);
h2 = plot(xh2,yh2); hold off; grid on;
axis([xh2(1) FA_AD/2 -0.1 1.1])
xlabel('Frequencia (Hz)'), ylabel('Magnitude (norm.)'),
title('Audio In - Espectro de Frequências');
set(h2,'YDataSource','yh2');
set(h2,'XDataSource','xh2');

% plota oscilograma do sinal de saida
subplot(222);
xh3 = (0:NSAMPLES_DA-1)/(FA_DA*0.001);
yh3 = xh3*0;
h3 = plot(xh3,yh3); hold off; grid on;
```

```
44 axis([[0 NSAMPLES_DA/10]/(FA_DA*0.001) -1.0 1.0]);
45 xlabel('Tempo (mseg.)'), ylabel('Amplitude (norm.)');
46 title('Audio Out - Oscilograma');
47 set(h3,'YDataSource','yh3');
48 set(h3,'XDataSource','xh3');
49
50 % plota espectro do sinal de saida
51 subplot(224);
52 yh4 = abs(fft(yh3,length(yh3)));
53 yh4 = yh4(1:length(yh4)/2);
54 yh4 =  yh4/max(yh4);
55 xh4 = (0:(length(yh4)-1));
56 xh4 = xh4*FA_DA/2/length(xh4);
57 h4 = plot(xh4,yh4); hold off; grid on;
58 axis([xh4(1) FA_DA/2 -0.1 1.1])
59 xlabel('Frequencia (Hz)'), ylabel('Magnitude (norm.)'),
60 title('Audio Out - Espectro de Frequências');
61 set(h4,'YDataSource','yh4');
62 set(h4,'XDataSource','xh4');
63
64 % Para MATLAB R2020a
65 ARec = audioDeviceReader('SampleRate',FA_AD,'NumChannels',CH_AD,'SamplesPerFrame',NSAMPLES_AD);
66 APly = audioDeviceWriter('SampleRate',FA_DA,'BufferSize',NSAMPLES_DA);
67 % %
68
69 tic;
70 fprintf('Start recording.\n');
71
72 fo = 1000; % frequencia central em Hz
73 bf = 500;  % banda para projeto do rejeita banda em Hz
74 gdB = -25; % ganho em dB na frequencia fo
75
76 fa = 48000; % frequencia de amostragem em Hz
77
78 lbt = 500;              % largura de banda de transicao para calculo da ordem do filtro
79 fsi = fo - bf/2 - lbt;  % frequencia limite da banda de passagem inferior em Hz
80 fpi = fo - bf/2;        % frequencia inferior da banda de rejeicao em Hz
81 fps = fo + bf/2;        % frequencia superior da banda de rejeicao em Hz
82 fss = fo + bf/2 + lbt ; % frequencia limite da banda de passagem superior em Hz
83
84 % Frequencias normalizadas (w = 2*pi*f/fa)
85 wsi = 2*pi*fsi/fa;
86 wpi = 2*pi*fpi/fa;
87 wps = 2*pi*fps/fa;
88 wss = 2*pi*fss/fa;
89
90 % Fequencia de corte do filtro passa-baixas
91 w1 = (wpi+wsi)/2;
92 w2 = (wss+wps)/2;
```

```
93  wc = (w2-w1)/2;
94
95  % Frequencia central do passa-banda
96  w0 = (w2+w1)/2;
97
98  % Largura da banda de transicao para projeto do rejeita banda
99  dw = min((wpi-wsi),(wss-wps));
100
101 N = 12*pi/dw;  % janela de Blackman
102 N = 2*ceil(N/2); % ordem deve ser par
103 wn = blackman(N+1);
104
105 n = 0:N;
106 hlp = sin(wc*(n+1e-8-N/2))./(n+1e-8-N/2)/pi; % passa-baixas
107 hbp = 2*cos(w0*(n-N/2)).*hlp; % passa-banda
108 hbr = -hbp;
109 hbr(N/2+1) =1+hbr(N/2+1); % rejeita-banda
110 temp = 10^(gdB/20)
111 hn = hbr + 10^(gdB/20)*hbp;
112
113 % Resposta a amostra unitaria (impulso) do filtro a ser implementado
114
115 b = hn;
116
117 zi = zeros([1 N]); % condições iniciais para o filtro da Funcao
118
119 while(toc < TT),
120     % le dados do A/D
121     yh1 = step(ARec);
122
123     % processa dados
124     [yh3 zi] = Funcao(yh1, b, zi);
125
126     % escreve dados no D/A
127     step(APly, yh3);
128
129     % atualiza oscilogramas
130     refreshdata(h1);
131     refreshdata(h3);
132
133     % atualiza espectro entrada
134     yh2=abs(fft(yh1,length(yh1)));
135     yh2=yh2(1:length(yh2)/2);
136     yh2=yh2/max(yh2);
137     refreshdata(h2);
138
139     % atualiza espectro saida
140     yh4=abs(fft(yh3,length(yh3)));
141     yh4=yh4(1:length(yh3)/2);
```

```
142     yh4=yh4/max(yh4);
143     refreshdata(h4);
144
145     % força atualização dos gráficos
146     drawnow;
147
148 end;
149 fprintf('End recording.\n');
150
151 release(ARec); % fecha dispositivo de entrada de áudio
152 clear ARec;    % remove ARec do workspace
153 release(APly); % fecha dispositivo de saída de áudio
154 clear APly;    % remove APly do workspace
```

Função para aplicar o filtro.

```
1  function [y, zf] = Funcao(x, b, zi)
2  % x - sinal a ser filtrado
3  % zi - condicoes iniciais do filtro
4  % zi - condicoes finais do filtro
5  % y - sinal filtrado
6
7  % Filtragem
8  %*****************************************
9  y = x*0;
10 [y zf] = filter(b,1,x,zi);
11 %*****************************************
12
13 end
```

# APÊNDICE C – EQUALIZADOR EM PYTHON COM AUXÍLIO GRÁFICO

```
#!/usr/bin/env python3

import argparse
import queue
import sys

from matplotlib.animation import FuncAnimation
import matplotlib.pyplot as plt
import numpy as np
import sounddevice as sd
import numpy  # Make sure NumPy is loaded before it is used in the callback
assert numpy  # avoid "imported but unused" message (W0611)

import math

from array import array

from scipy import signal

from scipy.signal import lfilter, lfilter_zi, butter, freqz, lfiltic

from numpy import array, ones

#inicialização e configuração do dispositivo de audio
############################################################
def int_or_str(text):
    """Helper function for argument parsing."""
    try:
        return int(text)
    except ValueError:
        return text


parser = argparse.ArgumentParser(add_help=False)
parser.add_argument(
    '-l', '--list-devices', action='store_true',
    help='show list of audio devices and exit')
args, remaining = parser.parse_known_args()
if args.list_devices:
    print(sd.query_devices())
    parser.exit(0)
parser = argparse.ArgumentParser(
    description=__doc__,
    formatter_class=argparse.RawDescriptionHelpFormatter,
    parents=[parser])
parser.add_argument(
```

```
    'channels', type=int, default=[1], nargs='*', metavar='CHANNEL',
    help='input channels to plot (default: the first)')
parser.add_argument(
    '-d', '--device', type=int_or_str,
    help='input device (numeric ID or substring)')
parser.add_argument(
    '-w', '--window', type=float, default=200, metavar='DURATION',
    help='visible time slot (default: %(default)s ms)')
parser.add_argument(
    '-i', '--interval', type=float, default=30,
    help='minimum time between plot updates (default: %(default)s ms)')
parser.add_argument(
    '-b', '--blocksize', type=int, help='block size (in samples)')
parser.add_argument(
    '-r', '--samplerate', type=float, help='sampling rate of audio device')
parser.add_argument(
    '-n', '--downsample', type=int, default=10, metavar='N',
    help='display every Nth sample (default: %(default)s)')
args = parser.parse_args(remaining)
if any(c < 1 for c in args.channels):
    parser.error('argument CHANNEL: must be >= 1')
mapping = [c - 1 for c in args.channels]  # Channel numbers start with 1
q = queue.Queue()
############################################################

fo = 1000 # frequencia central em Hz
bf = 500  # banda para projeto do rejeita banda em Hz
gdB = 5 # ganho em dB na frequencia fo

fa = 48000 # frequencia de amostragem em Hz

lbt = 50                # largura de banda de transicao para calculo da ordem do filtro
fsi = fo - bf/2 - lbt   # frequencia limite da banda de passagem inferior em Hz
fpi = fo - bf/2         # frequencia inferior da banda de rejeicao em Hz
fps = fo + bf/2         # frequencia superior da banda de rejeicao em Hz
fss = fo + bf/2 + lbt   # frequencia limite da banda de passagem superior em Hz



# Frequencias normalizadas (w = 2*pi*f/fa)
wsi = 2*math.pi*fsi/fa
wpi = 2*math.pi*fpi/fa
wps = 2*math.pi*fps/fa
wss = 2*math.pi*fss/fa

# Fequencia de corte do filtro passa-baixas
w1 = (wpi+wsi)/2
w2 = (wss+wps)/2
wc = (w2-w1)/2
```

```
96
97  # Frequencia central do passa-banda
98  w0 = (w2+w1)/2
99
100 # Largura da banda de transicao para projeto do rejeita banda
101 dw = min((wpi-wsi),(wss-wps))
102
103 N = (12*math.pi/dw)  # janela de Blackman
104 N = 2*math.ceil(N/2) # ordem deve ser par
105 wn = np.blackman(N+1)
106
107
108 hlp = list(range(N))
109 hbp = list(range(N))
110 hbr = list(range(N))
111 hn = list(range(N))
112
113 for n in range(N):
114     hlp[n] = math.sin(wc*(n+1e-8-N/2))/(n+1e-8-N/2)/math.pi # passa-baixas
115     hbp[n] = 2*math.cos(w0*(n-N/2))*hlp[n] # passa-banda
116     hbr[n] = hbp[n]*(-1)
117
118
119 hbr[math.ceil((N/2)+1)] = 1 + hbr[math.ceil((N/2)+1)] # rejeita-banda
120 gdb = 10**(gdB/20)
121
122 for n in range(1,N):
123     hn[n] = hbr[n] + gdb*hbp[n]
124
125 # Resposta a amostra unitaria (impulso) do filtro a ser implementado
126 for n in range(0,N):
127     hn[n] = hn[n]*wn[n]
128
129 zi1 = list(range(N-1))
130 #iniciando zi com 0
131 for n in range(1, (N-1)):
132     zi1[n] = 0
133
134 #Transformando de lista para array
135 b = np.array(hn)
136 zi = np.array(zi1)
137
138 a = 1
139
140 #Funcao loop principal
141 def audio_callback(indata, frames, time, status):
142
143     if status:
144         print(status, file=sys.stderr)
```

```
145
146         global zi
147         #Transformacao das amostras de entrada de lista para array
148         ############################################################
149         concat_list = [j for i in indata for j in i]
150         i = np.array(concat_list)
151         ############################################################
152
153         #Chamada da funcao de filtro
154         y, zf = lfilter(b, a, i, zi=zi*1)
155         #Novas condicoes iniciais
156         zi = zf
157
158        #transformacao de array para lista
159        ############################################################
160         o = []
161         for var in y:
162             aux = []
163             aux.append(var)
164             o.append(aux)
165         o1 = np.array(o)
166         ############################################################
167
168         q.put(o1[::args.downsample, mapping])
169
170     def update_plot(frame):
171         """This is called by matplotlib for each plot update.
172
173         Typically, audio callbacks happen more frequently than plot updates,
174         therefore the queue tends to contain multiple blocks of audio data.
175
176         """
177         global plotdata
178         while True:
179             try:
180                 data = q.get_nowait()
181             except queue.Empty:
182                 break
183             shift = len(data)
184             plotdata = np.roll(plotdata, -shift, axis=0)
185             plotdata[-shift:, :] = data
186         for column, line in enumerate(lines):
187             line.set_ydata(plotdata[:, column])
188         return lines
189
190     try:
191         if args.samplerate is None:
192             device_info = sd.query_devices(args.device, 'input')
193             args.samplerate = device_info['default_samplerate']
```

```
194
195     length = int(args.window * args.samplerate / (1000 * args.downsample))
196     plotdata = np.zeros((length, len(args.channels)))
197
198     fig, ax = plt.subplots()
199     lines = ax.plot(plotdata)
200     if len(args.channels) > 1:
201         ax.legend(['channel {}'.format(c) for c in args.channels],
202                   loc='lower left', ncol=len(args.channels))
203     ax.axis((0, len(plotdata), -1, 1))
204     ax.set_yticks([0])
205     ax.yaxis.grid(True)
206     ax.tick_params(bottom=False, top=False, labelbottom=False,
207                    right=False, left=False, labelleft=False)
208     fig.tight_layout(pad=0)
209     flag = 0
210     #Chamada do loop principal
211     stream = sd.InputStream(
212         device=args.device, channels=max(args.channels),
213         samplerate=48000, callback=audio_callback)
214
215     ani = FuncAnimation(fig, update_plot, interval=args.interval, blit=True)
216     with stream:
217         plt.show()
218
219
220 except Exception as e:
221     parser.exit(type(e).__name__ + ': ' + str(e))
```

# APÊNDICE D – EQUALIZADOR EM PYTHON SEM AUXÍLIO GRÁFICO

A seguir é mostrado o código do equalizador que foi testado na *Raspberry*, sem o auxílio visual, somente colocando o áudio equalizado na sáida.

```
#!/usr/bin/env python3

import argparse
import queue
import sys

from matplotlib.animation import FuncAnimation
import matplotlib.pyplot as plt
import numpy as np
import sounddevice as sd
import numpy  # Make sure NumPy is loaded before it is used in the callback
assert numpy  # avoid "imported but unused" message (W0611)

import math

from array import array

from scipy import signal


from scipy.signal import lfilter, lfilter_zi, butter, freqz

from numpy import array, ones

#inicialização e configuração do dispositivo de audio
############################################################
def int_or_str(text):
    """Helper function for argument parsing."""
    try:
        return int(text)
    except ValueError:
        return text


parser = argparse.ArgumentParser(add_help=False)
parser.add_argument(
    '-l', '--list-devices', action='store_true',
    help='show list of audio devices and exit')
args, remaining = parser.parse_known_args()
if args.list_devices:
    print(sd.query_devices())
    parser.exit(0)
```

```
43 parser = argparse.ArgumentParser(
44     description=__doc__,
45     formatter_class=argparse.RawDescriptionHelpFormatter,
46     parents=[parser])
47 parser.add_argument(
48     '-i', '--input-device', type=int_or_str,
49     help='input device (numeric ID or substring)')
50 parser.add_argument(
51     '-o', '--output-device', type=int_or_str,
52     help='output device (numeric ID or substring)')
53 parser.add_argument(
54     '-c', '--channels', type=int, default=1,
55     help='number of channels')
56 parser.add_argument('--dtype', help='audio data type')
57 parser.add_argument('--samplerate', type=float, help='sampling rate')
58 parser.add_argument('--blocksize', type=int, help='block size')
59 parser.add_argument('--latency', type=float, help='latency in seconds')
60 args = parser.parse_args(remaining)
61 ############################################################
62
63 fo = 5000   # frequencia central em Hz
64 bf = 50     # banda para projeto do rejeita banda em Hz
65 gdB = -100  # ganho em dB na frequencia fo
66
67 fa = 48000  # frequencia de amostragem em Hz
68
69 lbt = 500               # largura de banda de transicao para calculo da ordem do filtro
70 fsi = fo - bf/2 - lbt   # frequencia limite da banda de passagem inferior em Hz
71 fpi = fo - bf/2         # frequencia inferior da banda de rejeicao em Hz
72 fps = fo + bf/2         # frequencia superior da banda de rejeicao em Hz
73 fss = fo + bf/2 + lbt   # frequencia limite da banda de passagem superior em Hz
74
75 # Frequencias normalizadas (w = 2*pi*f/fa)
76 wsi = 2*math.pi*fsi/fa
77 wpi = 2*math.pi*fpi/fa
78 wps = 2*math.pi*fps/fa
79 wss = 2*math.pi*fss/fa
80
81 # Fequencia de corte do filtro passa-baixas
82 w1 = (wpi+wsi)/2
83 w2 = (wss+wps)/2
84 wc = (w2-w1)/2
85
86 # Frequencia central do passa-banda
87 w0 = (w2+w1)/2
88
89 # Largura da banda de transicao para projeto do rejeita banda
90 dw = min((wpi-wsi),(wss-wps))
91
```

```
92  N = (12*math.pi/dw)  # janela de Blackman
93  N = 2*math.ceil(N/2) # ordem deve ser par
94  wn = np.blackman(N+1)
95
96  hlp = list(range(N))
97  hbp = list(range(N))
98  hbr = list(range(N))
99  hn = list(range(N))
100
101 for n in range(N):
102     hlp[n] = math.sin(wc*(n+1e-8-N/2))/(n+1e-8-N/2)/math.pi # passa-baixas
103     hbp[n] = 2*math.cos(w0*(n-N/2))*hlp[n] # passa-banda
104     hbr[n] = hbp[n]*(-1)
105
106 hbr[math.ceil((N/2)+1)] = 1 + hbr[math.ceil((N/2)+1)] # rejeita-banda
107 gdb = 10**(gdB/20)
108 for n in range(1,N):
109     hn[n] = hbr[n] + gdb*hbp[n]
110
111 # Resposta a amostra unitaria (impulso) do filtro a ser implementado
112 for n in range(0,N):
113     hn[n] = hn[n]*wn[n]
114
115 zi1 = list(range(N-1))
116 #iniciando zi com 0
117 for n in range(1, (N-1)):
118     zi1[n] = 0
119
120 #Transformando de lista para array
121 b = np.array(hn)
122 zi = np.array(zi1)
123
124 a = 1
125
126 #Funcao loop principal
127 def callback(indata, outdata, frames, time, status):
128     if status:
129         print(status)
130
131     global zi
132     #Transformacao das amostras de entrada de lista para array
133     ###########################################################
134     concat_list = [j for i in indata for j in i]
135     i = np.array(concat_list)
136     ###########################################################
137     #Chamada da funcao de filtro
138     y, zf = lfilter(b, 1, i, zi=zi*1)
139     #Novas condicoes iniciais
140     zi = zf
```

```
141
142     #transformacao de array para lista
143     ############################################################
144      o = []
145      for var in y:
146          aux = []
147          aux.append(var)
148          o.append(aux)
149
150      o1 = np.array(o)
151      ############################################################
152      #Mandando o áudio para a saida
153      outdata[:] = o1
154
155
156 try:
157     with sd.Stream(device=(args.input_device, args.output_device),
158                    samplerate=48000, blocksize=args.blocksize,
159                    dtype=args.dtype, latency=args.latency,
160                    channels=args.channels, callback=callback):
161         print('#' * 80)
162         print('press Return to quit')
163         print('#' * 80)
164         input()
165 except KeyboardInterrupt:
166     parser.exit('')
167 except Exception as e:
168     parser.exit(type(e).__name__ + ': ' + str(e))
```